Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Sharded Machine Forgetting for User Data Deletion in the Ad Ranking Model Training Pipeline

Que Yang

HoschekOdle4810@outlook.com

Department of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, United States

Yukun Du

Department of Computer Science University of Chicago Chicago, IL 60637, United States

Yuang Hu

Department of Computer Science Carnegie Mellon University 5000 Forbes Avenue, Pittsburgh, PA 15213, United States

---

Research Article

Keywords: machine forgetting, SISA partitioned training, data deletion requests, ad ranking model, membership inference attacks, equivalence test

Posted Date: November 20th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11785586/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Sharded Machine Forgetting for User Data Deletion in the Ad Ranking Model Training Pipeline

Que Yang*
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, United States
quey@alumni.cmu.edu

Yukun Du
Department of Computer Science
University of Chicago
Chicago, IL 60637, United States
yukundu@uchicago.edu

Yuang Hu
Department of Computer Science
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, United States
yuangh@alumni.cmu.edu

Abstract: ***Once a user requests that their data be deleted, their behavior data should not affect the training process anymore, but behavior data prior to their deletion should still have an impact on ranking models already deployed, and frequently retraining ranking models due to deletion requests is too expensive. In this work, we add SISA sharded forgetting training method to the training pipeline of the advertising ranking model. The training data is partitioned into 32 disjoins sets (shards) with the user hashes, each split into 8 segments of time, each checkpointed between segments. A DCN-V2 submodel is trained separately for each shard, and during inference, weighted average of the outcomes from submodels will be conducted. A deletion request only affects the shards that are deleted to be retrained from the time segment corresponding to the deletion request. The data employed included 40.43 million clicks by the testing experiments carried out on Avazu click logs and 25 million ratings by 162, 000 users provided by MovieLens 25M. The deletion requests were simulated to be requested from users at the rate of deletion of 0.05% users per day and forgetting effect evaluated using Likelihood Ratio Membership Inference Attack (LiRA). For the Avazu data set, the AUC went down from 0.7853 to 0.7821 (0.6% difference in log loss), training cost for one batch of deletion decreased from 6.4 GPU hours for a complete retraining to 0.31 GPU hours (95.2% decrease in training effort) and time from request to updated model was decreased from 7 days to 3.2 hours. Following the forgetting LiRA obtained a true positive rate of 0.11% at a 0.1% false positive rate for deleted users, which was not significantly different from control users who had not been involved in training (0.10%), and the gradient-ascending approximate forgetting method was still 1.9%. Two one-sided tests (TOST) showed that for non-deleted users, the difference in the prediction scores for the SISA model and the fully retrained model was not different by more than ±0.002 ($p < 0.01$). The larger the number of partitions, the lower the retraining cost (roughly proportional to 1/npartitions) while the AUC loss grew from 0.0011 to 0.0068, with 32 partitions being considered as a good compromise between cost and accuracy.***

Keywords: ***machine forgetting; SISA partitioned training; data deletion requests; ad ranking model; membership inference attacks; equivalence test***

1 Introduction

As the programmatic advertising industry and individual recommendations are evolving with time, recommendations now come heavily relying on large amount of users' behavioural data used for training the model. The requirements for user data deletion requests introduce more stringent requirements into the model training, model management and online model service updating. Removing raw data from the database does not immediately remove the knowledge that has previously been learned from the models, and completely re-training existing models incurs computational overheads, long training periods and problems in running services continuously. Meeting the data deletion requirements, controlling the costs of retraining models, reducing any residual privacy risk, and performance in ranking is now a major challenge for the engineering applications of ad ranking systems. To solve such requirements, we propose SISA (Sliced Information-Forget) mechanism which trains the data according to the users hash and order. We have integrated the independent training of the DCN-V2, staged checkpoint restoring, and local retraining of affected slices to form a technical workflow that includes request localization, forget operation, model merging, and performance testing. We thoroughly test with the Avazu and MovieLens 25M datasets, considering ranking accuracy, member inference risk, computational cost and update latency. Further, we explore the estimated value for the undeleted users using equivalence tests and offer experimental evidence to support the design of data deletion processing and verifiable forgetting mechanism for ad ranking model.

2 Analysis of User Data Deletion Requirements for Ad Ranking Models

The deletion of user data in advertising ranking models includes the deletion of associated objects such as raw click logs, feature stores, and the sample caches, training snapshots, checkpoints and deployed model parameters, etc., if only raw click logs are deleted, the impact of historical training cannot be eliminated. Mechanisms from this system should be created for request verification, login of user record, tracking of derived data, association with training version, updating the model data, and logging the audit trail[1]. In addition, there is an obligation to reply to a request within 45 days from the request date (or within 90 days if necessary) which, as per EU GDPR, is within a month, which can be extended by further two months if it is complex. Thus, the training pipeline should have a clear definition of deletion target and impact on models, control the scope of retraining, verify the retraining status and processing log, as the basic basis for the next design of the sharded forgetting mechanism[2] .

3 A Sharded Machine Forgetting Method for Deleting User Data in the Training Pipeline of Ad Ranking Models

3.1 Overall Architecture of the Sharded Machine Forgetting Training Pipeline

The algorithm computes a hash based on a stable user identifier, and divides the training samples into 32 mutually exclusive fragments. Then, each fragment is separated into 8 segments in time sequence, and the sample index, time limit, and model version are recorded. The DCN-V2 model is trained separately by each shard, and the model parameters are preserved. Once the deletion request has been verified successfully, a user level deletion list is created to identify the affected fragments and the earliest relevant time segments. Training is then resumed from the corresponding checkpoint, with the target user's records excluded and the model retrained sequentially. During inference, predefined aggregation weights combine the outputs of all submodels to generate final ranking scores. Ranking utility is assessed

using AUC, log loss, and TOST, whereas privacy leakage is evaluated through the LiRA membership inference attack[3]. Figure 1 illustrates the data-processing, model-training, and deletion-control workflows.

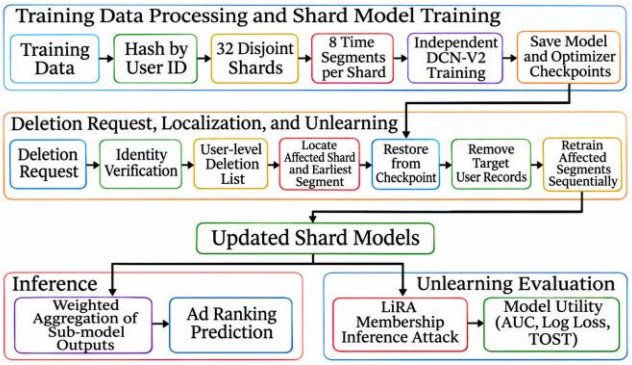

Figure 1: Overall Architecture of the Sharded Machine Forgetting Training Pipeline

3.2 User-Hash-Based Training Data Sharding and Temporal Slicing

This method is based on rules to organize data, namely (1) user hash mapping which requires that a unique hash ID be assigned to every user to accurately map deletion requests to corresponding samples of training data, and (2) sample chronological order in which a deletion request is mapped to the corresponding sample of training data according to time. Normalisation of user identifiers and calculation of a SHA-256 hash (boxed the first 64bits and converted to unsigned int and calculated shard affiliation using mod function), all records for same user check out to be the same in a single shard. (2) Time-based segmentation. Samples are sorted by the increasing UTC time (second by second).Time intervals are defined as left-closed, right-open ranges $[t_i, t_{i+1})$ and the start and end times of each interval, along with the sample indices, are recorded. (3)Construction of deletion index. Creates an association index linking the user ID, the shard ID, the time interval ID, and the sample location. Duplicate records undergo key value verification to provide traceability of data location for local re-training[4] .

3.3 Shard-Independent Training and Checkpoint Management Based on DCN-V2

Each shard uses an independent DCN-V2 model. The cross-network extracts higher-order feature interactions, while the deep network learns nonlinear representations. For binary click prediction on the Avazu dataset, the output layer employs a sigmoid activation function and binary cross-entropy with L2 regularization. To extend the architecture to multi-class advertising scoring, the binary output layer can be replaced with a C-dimensional softmax layer, while the cross-network, deep-network structure, user-hash sharding, and temporal checkpoint mechanism remain unchanged. The multi-class branch adopts categorical cross-entropy with L2 regularization to distinguish different advertising score categories:

$$L_{k,j} = -\frac{1}{N_{k,j}} \sum_{i=1}^{N_{k,j}} \left[y_i \ln \hat{p}_i + (1-y_i) \ln(1-\hat{p}_i) \right] + \frac{\lambda}{2} \| \theta_{k,j} \|_2^2 \quad (1)$$

where k is the segment ID, j is the time period ID, $N_{k,j}$ is the number of samples in the current segment, y_i is the click label, $\hat{p}_i$ is the predicted probability, $\theta_{k,j}$ is the model parameter, and λ is the regularization coefficient.The Adam optimizer is used, with parameters set to $\beta_1 = 0.9$、$\beta_2 = 0.999$、$\varepsilon = 10^{-8}$. After the completion of every training segment, the model weights, the first and second moments, the random number status, the feature coding version, and the data fingerprint are stored. Once the deletion request is received, the effective status before the target time period is restored according to the record index, providing a consistent initialization condition for local re-training[5] .

3.4 Local Retraining and Forgetting in Response to Deletion Requests

The associated record is located on the basis of a 256 bit SHA-256 digest and a 64 bit user hash index after it has gone through identity verification, request ID verification, and idempotency check. A single user request is mapped to one shard, while batch requests are deduplicated and merged by shard, involving up to 32 shards[6] .The method identifies the earliest relevant segment among the eight time segments in each affected shard j_0 , restores the model weights and the first- and second-order Adam parameters from the checkpoint of the preceding segment, and sequentially retrains from j_0 to the eighth segment. During training, all records associated with the target user are excluded, while the original feature encodings, sample chronology, and training configurations are retained. The parameter update relationships are as follows:

$$(\theta'_{k,j}, m'_{k,j}, v'_{k,j}) = \mathrm{Train}\left(\theta'_{k,j-1}, m'_{k,j-1}, v'_{k,j-1}, D_{k,j}\ ,\ \ D_U; \phi_j, \xi_j\right) \quad (2)$$

where k is the shard ID, j is the time segment ID, θ' are the updated model parameters, m'、υ' are the first- and second-order moment states of Adam, respectively, $D_{k,j}$ is the training dataset for this segment, D_U is the set of association records for the user to be removed, ϕ_j is the feature processing configuration, and ξ_j is the random seed and training configuration.After re-training, the method checks the deletion list, model version, data fingerprint, and checkpoint integrity, and then updates the release records of the respective sub-models. Figure 2 illustrates the association boundary between the Ad Ranking Object and the User Behavior Record.



Figure 2: Schematic Diagram of Product Ad Creatives in an Ad Ranking Scenario

For multi-class advertising scoring tasks, the DCN-V2 architecture can be extended by replacing the original sigmoid output layer with a C-dimensional softmax layer. The cross-network and deep-network structures remain unchanged, while categorical cross-entropy with L2 regularization is adopted for multi-class optimization. The user-hash partitioning, temporal slicing, and checkpoint restoration procedures are retained, ensuring that the forgetting mechanism remains consistent across binary and multi-class prediction tasks.

3.5 Multi-shard Model Aggregation and Online Inference

A unified model catalog manages submodel parameters, feature-encoding versions, release timestamps, and aggregation weightsDuring online inference, input features are converted into 32-bit floating-point values and dispatched to the shard-specific submodels through parallel execution.Each submodel receives the same candidate-ad feature vector and independently generates a click probability[7].For each request containing 100 candidate ads,

the inference scheduler distributes the associated computation tasks across 128 execution groups and returns the 10 candidates with the highest aggregated scores. The aggregated ranking score is calculated as follows:

$$\hat{p}(x) = \sum_{k=1}^{K} \alpha_k p_k(x), \qquad 0 \le \alpha_k \le 1, \quad \sum_{k=1}^{K} \alpha_k = 1 \tag{3}$$

where x is the candidate ad feature vector, K is the number of submodels participating in inference, $p_k(x) \in [0,1]$ is the click probability output by the th DCN-V2 model (k), α_k is the corresponding aggregation weight, and $\hat{p}(x)$ is the final predicted value. Model updates follow a dual-version switching mechanism that maintains the current production version alongside the pending release version. Version validation prevents parameter mixing during deployment and ensures consistency between model inference and online updates.The total online response latency is decomposed as follows:

$$T_{\text{req}} = T_{\text{feat}} + \max_{1 \le k \le K} T_k + T_{\text{agg}} + T_{\text{rank}} + T_{\text{out}} \tag{4}$$

In the formula, T_{req} represents the total request latency, T_{feat} represents the feature processing time, T_k represents the inference time for the th submodel (k), T_{agg} represents the aggregation time, T_{rank} represents the ranking time, and T_{out} represents the result output time.The system adopts a 100-ms request timeout threshold and records request IDs, model versions and stage-level timestamps. This threshold defines the maximum permitted response time, not measured inference latency. Measurements cover feature processing, parallel submodel inference, score aggregation, candidate ranking and response transmission. P50, P95 and P99 latency, throughput, timeout rate, CPU/GPU utilization and peak memory are monitored under varying traffic loads to identify resource bottlenecks and distinguish online inference latency from deletion-triggered model-update time.

3.6 Evaluation Mechanism for Forgetting Validity and Model Utility

Forgetting is evaluated across four dimensions: privacy leakage, ranking utility, computational cost and update timeliness[8]. LiRA compares deleted users with non-training controls, measuring attack TPR at the 0.1% operating point, with gradient-ascent-based approximate forgetting as a baseline. Utility is assessed against full retraining using AUC differences and relative log-loss changes. TOST tests paired prediction scores of non-deleted users, using ±0.002 as the primary equivalence margin and ±0.001/±0.005 for sensitivity analysis. For each margin, the 90% confidence interval and two one-sided test p-values are examined to assess equivalence under different statistical criteria. The evaluation metrics are defined as follows:

$$Q = \left(\text{TPR}_{\text{LiRA}} \big|_{\text{FPR}=10^{-3}}, \Delta_{\text{AUC}}, \Delta_{\text{LL}}, \text{TOST}_{\delta=0.002}, \rho_C, T_{\text{upd}} \right) \tag{5}$$

where Q is the comprehensive evaluation vector; TPR_{LiRA} is the true positive rate at a specified false positive rate; Δ_{AUC} is the AUC difference between the sharded model and the fully retrained model; Δ_{LL} is the relative difference in log loss between the two; $\text{TOST}_{\delta=0.002}$ is a two-sided test with an equivalence margin set at ±0.002; ρ_C is the computational cost ratio between partial retraining and full retraining; T_{upd} is the duration from the submission of a deletion request to the completion of the model update. In addition, metrics were recorded for 8, 16, 32, and 64 pieces, and assessment criteria were maintained consistent across Avazu and MovieLens 25M datasets to provide a basis for analyzing the relationship between privacy risk, model effectiveness, and computation overhead[9] .

4 Experimental Results and Analysis

4.1 Experimental Datasets and Configuration

Experiments use Avazu (40.43 million clicks over 10 days) for binary click prediction and deletion, and MovieLens 25M (25 million ratings, ~162,000 users, original 1–5 labels) for five-class DCN-V2 and forgetting evaluation. Metrics include accuracy, macro-F1, one-versus-rest macro-AUC, class-wise recall and categorical log loss. User-hash partitioning creates 32 exclusive shards, each containing 8 chronological checkpointed segments; deletion requests equal 0.05% of daily users. Full retraining, gradient-ascent-based approximate forgetting and SISA are compared, with shard-size sensitivity analysis. AUC, log loss, LiRA TPR, GPU-hours, update latency and TOST (±0.002) assess privacy–utility–computation trade-offs [10]. Online tests use 500/1,000/2,000/4,000 requests/s (100/200/400/800 clients), with 5-min warm-up and 15-min testing. Each request processes 100 candidate ads through 32 shard-specific models. P50/P95/P99/maximum latency, throughput, timeout rate, CPU/GPU utilization and peak memory are recorded. The 100-ms timeout limit is retained, with acceptance criteria of P95 <80 ms, P99 ≤100 ms and timeout rate <0.1%.

4.2 Impact of Sharded Machine Forgetting on Ad Ranking Performance

In order to evaluate the impact of sharded training on the performance of ad click prediction, we used a fully retrained model to calculate AUC and log loss on the Avazu data set, and summarized the results in Figure 3.

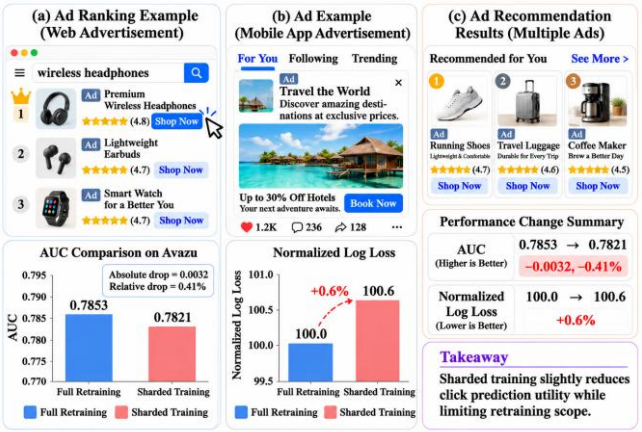

Figure 3: Impact of Segmented Machine Forgetting on Ad Ranking Performance

Figure 3 shows AUC values of 0.7853 for full retraining and 0.7821 for SISA, a decrease of 0.0032 (0.41%). Normalized log loss increases from 100.0 to 100.6 (0.6%). These results indicate slightly degraded ranking and prediction performance after shard-based retraining, reflecting a modest utility cost in exchange for reduced retraining workload through user-hash-based partitioning.

4.3 Analysis of Retraining Costs and Response Latency for Deletion Requests

In Avazu, full retraining consumed 6.4 GPU-hours per batch, versus 0.31 for SISA, saving 6.09 GPU-hours (95.2%); its computational cost was 20.65 times higher. Median deletion-to-update time fell from 7 days (168 h) to 3.2 h, a 164.8-h (98.1%) reduction, leaving 1/52.5 of the original duration. GPU-hours measure resource consumption, whereas end-to-end latency—not individual online-inference latency—covers checkpoint restoration, shard-local retraining, validation and deployment. Savings arise from retraining affected shards, restoring prior checkpoints and filtering target-user records. These results demonstrate computational efficiency, but not forgetting completeness, which requires member-inference-attack evaluation.

4.4 Verification of User Forgetting Effectiveness Based on LiRA

The Likelihood Ratio Member Inference Attack (LiRA) is employed to determine whether the retrained model retains identifiable membership information associated with deleted users.The comparison includes three groups: users subjected to SISA-based forgetting, control users excluded from model

training, and users processed through gradient-ascent-based approximate forgetting. Their true positive rates (TPRs) are reported in Table 1 under the same 0.1% false positive rate.

Table 1: LiRA Member Inference Attack Results for Different Forgetting Methods

Evaluation Subject	False Positive Rate (%)	True Positive Rate (%)
Users after SISA chunk-based forgetting	0.1	0.11
Control users not included in training	0.1	0.1
Gradient-Ascent-Based Approximate Forgetting	0.1	1.9

Table 1 shows that SISA achieves a 0.11% TPR, only 0.01 percentage points above non-training controls (0.10%). Gradient-ascent-based approximate forgetting reaches 1.90%, 1.79 percentage points and 17.27 times higher than SISA. Thus, SISA reduces TPR by 94.21% relative to gradient ascent, indicating substantially weaker residual membership signals. However, similarity to controls does not prove complete data removal. Reliability requires attack confidence intervals, repeated tests across random seeds, and utility evaluation on non-deleted users.

4.5 Equivalence Analysis of Predicted Utility for Non-Deleted Users

The two one-sided tests procedure (TOST) compares paired advertising click scores generated by the SISA-based model and the fully retrained model for 1,000 non-deleted-user samples. The primary equivalence bounds are set to −0.002 and +0.002. Additional sensitivity analyses employ margins of ±0.001 and ±0.005 to examine the influence of equivalence-bound selection on the statistical conclusion.The results of the test can be seen in the figure 4.

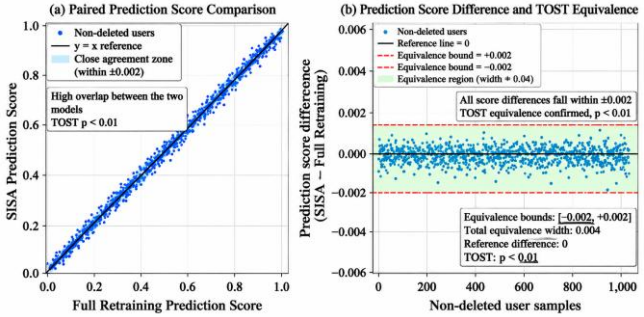

Figure 4: TOST Equivalence Test for Differences in Predicted Scores of Non-Deleted Users

For the primary ±0.002 margin, paired prediction-score differences satisfy equivalence, with significant TOST results ($p < 0.01$). The wider ±0.005 margin is also satisfied, while the stricter ±0.001 margin requires separate confidence-interval and one-sided-test verification. The primary equivalence interval spans −0.002 to +0.002, with a total width of 0.004. These findings indicate that shard-based retraining preserves non-deleted users' prediction behavior within the predefined equivalence margin.

4.6 Impact of the Number of Shards on Forgetting Cost and Model Accuracy

Four configurations (8, 16, 32 and 64 shards) are compared to assess partition granularity. Increasing shards reduces per-shard retraining workload; theoretically, the 64-shard cost is 0.125 of the 8-shard cost, an estimated 87.5% reduction, not a measured result. However, AUC loss rises from 0.0011 to 0.0068 (0.0057 increase; 6.18-fold), indicating potential utility degradation from smaller training subsets. The 32-shard configuration balances computational savings and ranking performance. Actual costs and AUC losses for 16 and 32 shards should be reported from the original experiments, rather than inferred, to establish applicability conditions and performance boundaries.

5. Conclusion

To address the residual influence of deleted user records and the high computational cost of full-model retraining, this study develops an SISA-based forgetting mechanism that integrates user-hash sharding, temporal slicing, and checkpoint restoration. The mechanism limits retraining to affected shards and evaluates forgetting effectiveness and model utility through LiRA, AUC, log loss, and TOST.The experimental results show that the GPU-hour consumption per deletion batch decreases by 95.2%, while the median model-update time is reduced to 3.2 h. The 32-shard configuration also provides a balance between prediction performance and retraining overhead.The current evaluation covers binary advertising click prediction and multi-class scoring tasks, but the generalization of the proposed method to advertising platforms with heterogeneous multi-class labels and complex scoring objectives requires further examination. Its robustness under dynamic feature updates and diverse deletion-request patterns also warrants additional validation. In future work, list deletion processes in reality could be added into the mechanisms and multi-class scoring models could be included to make the forgetting process more intelligently.

References

[1] Nguyen T T, Huynh T T, Ren Z, et al. A survey of machine unlearning[J]. ACM Transactions on Intelligent Systems and Technology, 2025, 16(5): 1-46.

[2] Kurmanji, M., Triantafillou, E., & Triantafillou, P. Machine unlearning in learned databases: An experimental analysis [J]. Proceedings of the ACM on Management of Data, 2024, 2(1): 1–26.

[3] Nguyen T T, Huynh T T, Ren Z, et al. A survey of machine unlearning[J]. ACM Transactions on Intelligent Systems and Technology, 2025, 16(5): 1–46.

[4] Huynh T T, Nguyen T B, Nguyen T T, et al. Certified Unlearning for Federated Recommendation[J]. ACM Transactions on Information Systems, 2025, 43(2): 1-29.

[5] Cosentino C, Gatto S, Liò P, et al. Machine Unlearning: A Perspective, Taxonomy, and Benchmark Evaluation[J]. Future Internet, 2026, 18(3): 174.

[6] Cevallos I D, Benalcázar M E, Valdivieso Caraguay Á L, et al. A Systematic Literature Review of Machine Unlearning Techniques in Neural Networks[J]. Computers, 2025, 14(4): 150.

[7] Guo Q, Pan C, Zhang H, et al. Efficient Unlearning for Spatio-Temporal Graphs (Student Abstract)[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2025, 39(28): 29382–29384.

[8] Aslam I, Khosrowi D, Nagshi R. Learning to Unlearn, Failing to Forget? Assessing Machine Unlearning Through Ethics and Epistemology [C]//Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society. 2025, 8(1): 204-216.

[9] Sheng X, Bao W, Wang H, et al. Retaliatory Attacks Against Federated Unlearning via Data Leakage [C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2026, 40(30): 25321–25329.

[10] Qu Y, Yuan X, Ding M, et al. Learn to Unlearn: Insights into Machine Unlearning [J]. Computer, 2024, 57(3): 79–90.