



University of  
Zurich<sup>UZH</sup>

# Attack Strategies and Robustness Certification in Decentralized FMTL Systems

*Zhi Wang*  
*Zurich, Switzerland*  
*Student ID: 21-739-552*

Supervisor: Chao Feng  
Date of Submission: August 3, 2025



# Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, August 3, 2025

Zhi Wang  
Signature of student



# Zusammenfassung

Heute wurden Federated Learning (FL) und Multi-Task Learning (MTL) sowohl in zentralisierten als auch in dezentralen Paradigmen umfassend untersucht, jedoch wurde den Sicherheitslücken von dezentralen Federated Multi-Task Learning (DFMTL)-Systemen, insbesondere bei böartigem Verhalten von Clients, nur begrenzt Aufmerksamkeit geschenkt. Insbesondere konzentrierte sich die bisherige Arbeit hauptsächlich auf die Entwicklung zunehmend effizienter oder innovativer Frameworks, und es mangelt dem Bereich weiterhin an dedizierten Bedrohungsmodellen und empirischen Robustheitsbewertungen, die auf solche Systeme zugeschnitten sind. Um diese Lücke zu schließen, bewertet diese Arbeit die Robustheit des dezentralen Federated Multi-Task Learning (DFMTL)-Frameworks. Anschließend werden sieben verschiedene Poisoning-Angriffsstrategien innerhalb der DFMTL-Umgebung entworfen und implementiert: drei Data-Poisoning-Angriffe (Random Label Flip, Targeted Label Flipping, Trigger Injection), drei Model-Poisoning-Angriffe (Sign Flip, Scaled Boost, AT2FL) und ein Backdoor-ähnlicher Aggregationsangriff (Malicious Aggregation Filter).

Alle Angriffe werden einzeln innerhalb des DFMTL-Frameworks implementiert und an förderierte Szenarien mit Klassenlabel-Heterogenität (CIFAR-10) und Aufgabenheterogenität (CelebA: Multi-Label-Klassifikation vs. Gesichtslandmarken-Regressionsaufgabe) angepasst. Wichtig ist, dass bestimmte Angriffe, wie AT2FL und der Malicious Aggregation Filter, eine spezielle Gestaltung und Implementierung erforderten, um innerhalb der internen Kommunikations- und Optimierungsmechanismen des DFMTL-Frameworks zu funktionieren. In den Experimenten wird eine Teilmenge von Clients angegriffen. Die Bewertung analysiert sowohl den Leistungsverlust bei den angegriffenen Clients als auch die Kollateraleffekte bei nicht ausgewählten Clients innerhalb derselben Aufgabengruppe und über verschiedene Aufgabengruppen hinweg.

Die Ergebnisse zeigen, dass trotz der verteilten und aufgabenisolierten Natur von DFMTL einige Poisoning-Angriffe auf ausgewählte Clients sich über gemeinsame Aggregationspfade im System ausbreiten können, während andere lokal begrenzt bleiben. Die Ergebnisse geben Aufschluss darüber, welche Angriffsmethoden das größte Risiko darstellen. Diese Arbeit präsentiert eine Robustheitsbewertung eines DFMTL-Systems und bietet eine Grundlage für zukünftige Abwehrmaßnahmen, die auf DFMTL-Architekturen abzielen.



# Abstract

Today, Federated Learning (FL) and Multi-Task Learning (MTL) have been extensively studied in both centralized and decentralized paradigms, but limited attention has been paid to the security vulnerabilities of decentralized Federated Multi-Task Learning (DFMTL) systems, particularly under malicious client behavior. Specifically, most prior work has focused on developing increasingly efficient or innovative frameworks, and the field still lacks dedicated threat models and empirical robustness evaluations tailored to such systems. To address this gap, this thesis evaluates the robustness of the decentralized Federated Multi-Task Learning (DFMTL) framework. It then designs and implements seven distinct poisoning attack strategies within the DFMTL setting: three data poisoning attacks (Random Label Flip, Targeted Label Flipping, Trigger Injection), three model poisoning attacks (Sign Flip, Scaled Boost, AT2FL), and one backdoor-style aggregation attack (Malicious Aggregation Filter).

All attacks are implemented individually within the DFMTL framework and adapted to operate on both federated settings with class-label heterogeneity (CIFAR-10) and task heterogeneity (CelebA: multi-label classification vs. facial landmark regression). Importantly, certain attacks, such as AT2FL and the Malicious Aggregation Filter, required dedicated design and implementation to function within the internal communication and optimization mechanisms of the DFMTL framework. In the experiments, a subset of clients are attacked. The evaluation analyzes both the performance degradation on the attacked clients and the collateral impact on unselected clients within the same task group and across different task groups.

The findings reveal that despite the distributed and task-isolated nature of DFMTL, some poisoning attacks on selected clients can propagate across the system via shared aggregation pathways, while others remain localized. The results provide insight into which attack strategies pose the greatest risk. This work presents a robustness evaluation of a DFMTL system and provides a foundation for future defenses targeting DFMTL architectures.



# Acknowledgments

I want to thank my supervisor, Chao Feng. His feedback during our bi-weekly meetings was very valuable and helped guide the thesis' direction.

I also wish to thank Jiawei Zhang for carefully reviewing the grammar and language of the thesis prior to submission.

Lastly, I want to extend my thanks to Prof. Dr. Burkhard Stiller, the responsible supervising professor, for managing the administrative and academic coordination of this work.



# Contents

|                                                  |            |
|--------------------------------------------------|------------|
| <b>Declaration of Independence</b>               | <b>i</b>   |
| <b>Zusammenfassung</b>                           | <b>iii</b> |
| <b>Abstract</b>                                  | <b>v</b>   |
| <b>Acknowledgments</b>                           | <b>vii</b> |
| <b>1 Introduction</b>                            | <b>1</b>   |
| 1.1 Motivation . . . . .                         | 2          |
| 1.2 Description of Work . . . . .                | 2          |
| 1.3 Thesis Outline . . . . .                     | 3          |
| <b>2 Background</b>                              | <b>5</b>   |
| 2.1 Federated Learning . . . . .                 | 5          |
| 2.1.1 Federated Learning Basics . . . . .        | 5          |
| 2.1.2 Decentralized Federated Learning . . . . . | 6          |
| 2.2 Multi-Task Learning . . . . .                | 8          |
| 2.2.1 Overview of Multi-Task Learning . . . . .  | 8          |
| 2.2.2 Task Heterogeneity . . . . .               | 9          |
| 2.3 Federated Multi-Task Learning . . . . .      | 10         |
| 2.3.1 FMTL Architectures . . . . .               | 10         |
| 2.3.2 Personalization-Oriented FMTL . . . . .    | 11         |
| 2.3.3 FMTL with Task Heterogeneity . . . . .     | 12         |

|          |                                                                             |           |
|----------|-----------------------------------------------------------------------------|-----------|
| 2.4      | Decentralized Federated Multi-Task Learning . . . . .                       | 13        |
| 2.4.1    | FedHCA <sup>2</sup> : Hyper Conflict-Averse Aggregation for DFMTL . . . . . | 14        |
| 2.4.2    | FedPer-HCA framework . . . . .                                              | 14        |
| 2.5      | Poisoning Attacks in FL and MTL . . . . .                                   | 15        |
| 2.5.1    | Classification According to the Poisoning Methodology . . . . .             | 16        |
| 2.5.2    | Classification According to the Attacker's Intention . . . . .              | 17        |
| <b>3</b> | <b>Related Work</b>                                                         | <b>21</b> |
| 3.1      | Literature Review and Method . . . . .                                      | 21        |
| 3.2      | Classic Poisoning Attacks in Federated Learning. . . . .                    | 23        |
| 3.3      | Classic Attacks in Multi-Task Learning . . . . .                            | 27        |
| 3.4      | Poisoning Attacks in Decentralized Federated Multi-Task Learning . . . . .  | 29        |
| 3.4.1    | Limited Attention to Attacks in DFMTL . . . . .                             | 29        |
| 3.4.2    | Transferability of Attacks and Motivation for Design . . . . .              | 29        |
| <b>4</b> | <b>Attack Design</b>                                                        | <b>31</b> |
| 4.1      | Overview of Attack Design . . . . .                                         | 31        |
| 4.1.1    | Review of Experimental Settings in Fedper-HCA . . . . .                     | 31        |
| 4.1.2    | Attack Selection and Deployment Strategy . . . . .                          | 32        |
| 4.2      | Data Poisoning Attacks . . . . .                                            | 33        |
| 4.2.1    | Random Label Flipping and Partial Flipping . . . . .                        | 33        |
| 4.2.2    | Targeted Label Flipping . . . . .                                           | 33        |
| 4.2.3    | Trigger Injection . . . . .                                                 | 34        |
| 4.3      | Model Poisoning Attacks . . . . .                                           | 35        |
| 4.3.1    | Sign Flipping . . . . .                                                     | 35        |
| 4.3.2    | Scaled Boost . . . . .                                                      | 36        |
| 4.3.3    | AT2FL . . . . .                                                             | 36        |
| 4.4      | Backdoor Attack via Aggregation Manipulation . . . . .                      | 38        |
| 4.4.1    | Malicious Aggregation Filter . . . . .                                      | 38        |
| 4.5      | Summary of Attack Design . . . . .                                          | 39        |

|          |                                                 |           |
|----------|-------------------------------------------------|-----------|
| <b>5</b> | <b>Implementation</b>                           | <b>41</b> |
| 5.1      | Code Overview . . . . .                         | 41        |
| 5.1.1    | Directory Structure . . . . .                   | 41        |
| 5.1.2    | How to Run Experiments . . . . .                | 43        |
| 5.1.3    | Execution Flow . . . . .                        | 43        |
| 5.1.4    | Data Loading Mechanism . . . . .                | 44        |
| 5.1.5    | Attack Integration Design . . . . .             | 44        |
| 5.2      | Experimental Setup . . . . .                    | 44        |
| 5.2.1    | Objectives . . . . .                            | 45        |
| 5.2.2    | Experiment Groups and Data Splits . . . . .     | 45        |
| 5.2.3    | Controlled Variables and Attack Scope . . . . . | 45        |
| 5.2.4    | Attack Types and Experimental Runs . . . . .    | 45        |
| 5.2.5    | Training and Communication Settings . . . . .   | 46        |
| 5.3      | Poisoning Attack Implementation . . . . .       | 46        |
| 5.3.1    | Random Label Flip . . . . .                     | 46        |
| 5.3.2    | Targeted Label Flipping . . . . .               | 47        |
| 5.3.3    | Trigger Injection (Mosaic Patch) . . . . .      | 47        |
| 5.3.4    | Scaled Boost . . . . .                          | 47        |
| 5.3.5    | Sign Flip . . . . .                             | 48        |
| 5.3.6    | AT2FL (Implicit Gradient Poisoning) . . . . .   | 48        |
| 5.3.7    | Malicious Aggregation . . . . .                 | 48        |
| <b>6</b> | <b>Evaluation</b>                               | <b>49</b> |
| 6.1      | Experiment Overview . . . . .                   | 49        |
| 6.2      | Data poisoning scenarios . . . . .              | 50        |
| 6.2.1    | Random Label Flip . . . . .                     | 51        |
| 6.2.2    | Targeted Label Flipping . . . . .               | 54        |
| 6.2.3    | Trigger Injection . . . . .                     | 56        |

6.3 Model poisoning scenarios . . . . . 60

6.3.1 Sign Flipping . . . . . 60

6.3.2 Scaled Boost . . . . . 63

6.3.3 AT2FL . . . . . 65

6.4 Malicious Aggregation . . . . . 68

**7 Summary, Conclusions and Future Work 73**

7.1 Summary . . . . . 73

7.1.1 Key Findings . . . . . 73

7.2 Conclusions . . . . . 74

7.3 Limitations and Future Work . . . . . 74

**Bibliography 77**

**Abbreviations 81**

**List of Figures 81**

**List of Tables 84**

# Chapter 1

## Introduction

In recent years, with the continuous development of Federated Learning (FL) and Multi-Task Learning (MTL), the integration of them has emerged as a prominent paradigm in distributed training, gaining attention in both academic and industrial fields, especially in the direction of addressing client heterogeneity. Federated Multi-Task Learning (FMTL) enables clients to collaboratively learn personalized models for distinct but related tasks, benefiting from both privacy preservation and cross-task knowledge sharing [1]–[3].

Although significant advances have been made in centralized Federated Multi-Task Learning (CFMTL), which has a server to coordinate model aggregation [4], the decentralized part, which means that Decentralized Federated Multi-Task Learning (DFMTL) remains relatively underexplored due to the absence of a central coordinator and the increased complexity of managing task diversity.

Most existing studies on FMTL focus mainly on label heterogeneity in non-IID settings, often assuming clients work on the same task or share identical output spaces [5], [6]. However, task heterogeneity, where clients perform fundamentally different tasks, poses unique challenges, including conflicting gradient updates and incompatible model architectures, especially in decentralized environments [3], [7]. Recent efforts such as FedPer [8] and FedHCA2 [3] offer valuable information, but remain tied to centralized coordination.

To address this gap, Feng et al. [9] proposed ColNet (FedPer-HCA), a decentralized framework combining personalized model partitioning (FedPer) with a conflict-averse aggregation strategy (HCA), enabling effective collaboration across clients with distinct task distributions. The framework has demonstrated promising results in heterogeneous settings and serves as a strong foundation for further investigations into the robustness and security of DFMTL systems. In light of this, this thesis conducts a systematic robustness evaluation of FedPer-HCA by designing and injecting a series of poisoning attacks into the decentralized multitask training process. The motivation and rationale behind this security assessment are further elaborated in the following section.

## 1.1 Motivation

This thesis is motivated by the absence of systematic robustness evaluations in DFMTL settings. Although recent work has proposed frameworks that improve performance and personalization in task heterogeneity, their resilience against adversarial attacks remains unclear. Existing poisoning strategies are often designed for decentralized Federated Learning (DFL) or MTL, and may not directly transfer to DFMTL settings. In addition, the interaction between task heterogeneity, decentralized aggregation, and attack propagation is not well understood. Therefore, there is a pressing need to investigate whether existing attack paradigms remain effective under DFMTL, and if so, how these attacks manifest in such systems.

And previous research on poisoning attacks in FL has mainly focused on centralized scenarios. Data poisoning techniques such as label flipping and targeted mislabeling have been proven to be effective in compromising the precision of the global model [10]. Similarly, model poisoning methods, such as scaled gradient updates and sign flipping, can severely degrade or manipulate centralized models [11]. Several recent surveys have emphasized the importance of both attack and defense strategies in FL, but explicitly highlight that decentralized topologies and task heterogeneity significantly increase the complexity of systematic robustness analysis [12], [13].

In contrast, DFL research focusing on adversarial threats remains limited. Although studies such as DMPA explore model poisoning strategies in peer-to-peer FL settings, they operate under the assumption of homogeneous tasks and overlook the complexities introduced by task heterogeneity in both label spaces and learning objectives [14]. Consequently, it remains an open question whether DFMTL frameworks, such as FedPer-HCA, can inherit robustness insights from their centralized counterparts, or whether their structural innovations necessitate novel threat models.

Motivated by this knowledge gap, this thesis systematically evaluates the robustness of FedPer-HCA under a spectrum of poisoning attacks adapted for DFMTL. Target attack strategies range from classic techniques, such as Random Label Flip and Scaled Boost, to custom methods tailored to DFMTL framework, such as AT2FL and Malicious Aggregation Filter. Through these investigations, this thesis aims to uncover both the vulnerabilities and strengths of DFMTL systems, guiding future research on defenses and secure design.

## 1.2 Description of Work

This thesis presents a comprehensive robustness analysis of the FedPer-HCA framework by designing and implementing seven representative poisoning attack strategies within the DFMTL setting. These include three data poisoning attacks (Random Label Flip, Targeted Label Flipping, Trigger Injection), three model poisoning attacks (Sign Flip, Scaled Boost, AT2FL), and one backdoor-style aggregation attack (Malicious Aggregation Filter). Each attack is integrated into the FedPer-HCA codebase and adapted to operate on

both class-label heterogeneous tasks (CIFAR-10) and task-heterogeneous settings (CelebA multi-label classification vs. facial landmark regression).

The evaluation covers various client compromise scenarios and investigates not only local performance degradation but also cross-client effects within and across task groups. In addition, several attacks, such as AT2FL and Malicious Aggregation, required novel modifications to comply with the communication and optimization design of DFMTL.

## 1.3 Thesis Outline

The remainder of this thesis is organized as follows.

**Chapter 2** provides the necessary theoretical foundations, introducing Federated Learning (FL), Multi-Task Learning (MTL), Federated Multi-Task Learning (FMTL), Decentralized Federated Multi-Task Learning (DFMTL) and the classic poisoning attack.

**Chapter 3** presents a review of related literature, covering robustness issues and classic poisoning attacks in both FL and MTL, and identifying the gap in current research on DFMTL.

**Chapter 4** describes the design principles of seven poisoning attacks and their pseudocode implementations within the FedPer-HCA framework.

**Chapter 5** details the experimental setup, the implementation of seven distinct poisoning strategies, and their integration into the DFMTL system.

**Chapter 6** evaluates the impact of these attacks through experiments and discusses key findings, challenges, and mitigation efforts.

**Chapter 7** concludes the thesis, reflects on its contributions and limitations, and suggests directions for future research.



# Chapter 2

## Background

This chapter introduces key concepts relevant to this thesis. It begins with an overview of FL and DFL, followed by a review of MTL, FMTL, and DFMTL. Lastly, it explores classic poisoning attacks in FL and MTL, including training-time attacks (data poisoning, model poisoning, and backdoor attacks), as well as inference-time adversarial attacks.

### 2.1 Federated Learning

Federated learning has emerged as a promising paradigm for enabling collaborative model training across distributed clients while preserving data privacy. This section introduces the foundational concepts and key variants of federated learning, which form the basis for later discussions on its multi-task and adversarial extensions.

#### 2.1.1 Federated Learning Basics

FL is a distributed machine learning paradigm in which multiple clients collaborate to train a shared model without centralizing their raw data [15]. This approach addresses growing concerns over data privacy, bandwidth limitations, and regulatory constraints by enabling on-device learning and aggregating only model updates.

In a typical FL setting, each round of training consists of the following steps [16], and the overall process is illustrated in Figure 2.1:

1. **Model Broadcast:** A central server initializes or distributes a global model to participating clients.
2. **Local Training:** Each client uses its private local dataset to update the model for one or more epochs using stochastic gradient descent (SGD) or its variants.
3. **Upload Updates:** Clients send the resulting model parameters or gradients (not raw data) back to the server.

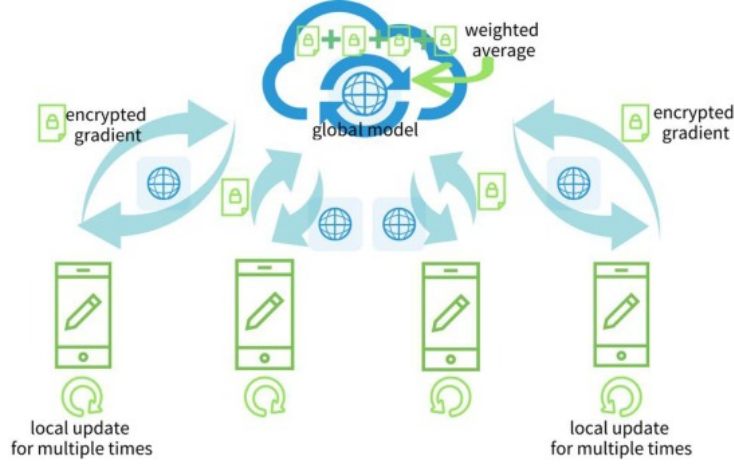


Figure 2.1: Illustration of FL framework proposed by Google [16]

4. **Aggregation:** The server combines these updates using an aggregation algorithm to produce a new global model, which is then broadcast for the next round.

One of the most widely used aggregation algorithms in FL is *Federated Averaging* (FedAvg) [15]. In FedAvg, the server computes a weighted average of the clients' local model parameters based on the size of their local datasets. The global model update at round  $t$  can be formally expressed as:

$$w^{(t)} = \sum_{k=1}^K \frac{n_k}{n} w_k^{(t)}$$

where  $w^{(t)}$  is the global model,  $w_k^{(t)}$  is the local model from client  $k$ ,  $n_k$  is the number of data samples on client  $k$ , and  $n = \sum_k n_k$  is the total number of samples across all participating clients.

FL presents several advantages, including reduced communication cost relative to transferring entire datasets, improved scalability, and enhanced privacy preservation. However, it also introduces new challenges, such as statistical heterogeneity (non-IID data across clients), system heterogeneity (e.g., varying compute and network resources), and robustness to adversarial participants.

### 2.1.2 Decentralized Federated Learning

DFL extends the classical federated learning paradigm by removing the central coordinating server. Instead, clients form a peer-to-peer communication network where local models or parameters are exchanged directly with neighboring nodes [4]. This architecture eliminates single points of failure, distributes trust, and improves the resilience of the system. Some frameworks adopt a hybrid approach, often referred to as Semi-DFL. In such settings, the role of server rotates among participants based on predefined criteria, enabling partial decentralization without complete loss of coordination. These distinctions are illustrated in Figure 2.2, adapted from Martínez et al. [4].

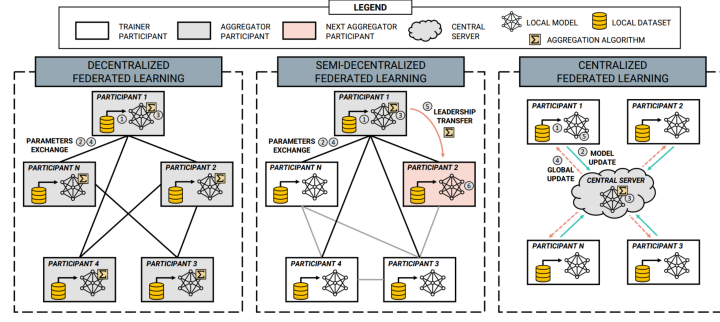


Figure 2.2: CFL, DFL and Semi-DFL as depicted by [4]

In contrast to centralized federated learning (CFL), where all updates are aggregated by a central server, DFL relies on local interactions among clients. The most common approaches include synchronous or asynchronous update propagation across a predefined network topology, such as fully connected graphs, random geometric graphs, or ring structures. The local update rule can often be expressed as:

$$w_i^{(t+1)} = \sum_{j \in \mathcal{N}(i)} a_{ij} w_j^{(t)}$$

Some algorithms also incorporate local updates before communication:

$$w_i^{(t+1)} = \sum_{j \in \mathcal{N}(i)} a_{ij} \left( w_j^{(t)} - \eta \nabla F_j(w_j^{(t)}) \right)$$

where  $w_i^{(t)}$  denotes the local model at client  $i$  at round  $t$ ,  $\mathcal{N}(i)$  is the set of neighbors of client  $i$ , and  $a_{ij}$  are the mixing weights satisfying  $\sum_j a_{ij} = 1$ .

DFL offers several advantages over CFL:

- **Scalability:** Communication and computation loads are distributed.
- **Fault tolerance:** No central server means higher resilience to node failures.
- **Privacy preservation:** No centralized model or metadata accumulation, although shared updates may still leak information via gradient-based attacks.

However, DFL also introduces unique challenges:

- **Convergence difficulties:** Especially in sparse or dynamic graphs, local model drift may hinder global consensus.
- **Communication overhead:** Frequent peer-to-peer communication may increase latency.

- **Heterogeneous trust and topology:** Clients may not uniformly trust neighbors or have balanced connectivity.

Several protocols and frameworks have been proposed to implement DFL effectively, such as Gossip-based learning, Braintorrent [17], and Diffusion-based models. These approaches vary in terms of convergence guarantees, robustness to adversarial clients, and communication efficiency.

## 2.2 Multi-Task Learning

MTL is a foundational concept for building models that can be generalized across multiple tasks. In the context of federated systems, MTL provides a natural mechanism to leverage task similarities while respecting task-specific requirements. This section presents a structured overview of MTL and its relevance to federated and decentralized settings.

### 2.2.1 Overview of Multi-Task Learning

MTL is a well-established subfield of machine learning where the objective is to train a model to perform multiple tasks simultaneously. This approach differs from conventional single-task learning, where models are optimized for a single objective only. By learning from related tasks in parallel, MTL can exploit shared information and inductive biases, which often results in improved generalization and reduced risk of overfitting [18], [19].

One of the key advantages of MTL lies in its ability to mitigate data scarcity. When individual tasks have limited data, jointly training them with related tasks can help the model generalize better by capturing common patterns across tasks. This makes MTL particularly attractive in domains with sparse labels or expensive annotation.

MTL can be broadly categorized into two types: *homogeneous* and *heterogeneous* MTL (also called conventional vs. heterogeneous in some literature). Homogeneous MTL typically involves a collection of tasks with similar objectives, e.g., a set of classification or regression tasks, which allows for effective sharing of intermediate representations. In contrast, heterogeneous MTL refers to scenarios where tasks differ significantly in nature (e.g., combining classification, detection, and regression tasks), making representation sharing more complex [20].

To organize the rapidly evolving landscape of MTL, several surveys have proposed taxonomies that classify MTL methods based on architecture, task type, and learning strategy. For example, Crawshaw [21] distinguishes MTL approaches across different application domains, such as computer vision, natural language processing, and reinforcement learning, while also discussing more specialized methods like learnable or conditional architectures. Other surveys have emphasized architectural structure, for example, encoder-focused vs. decoder-focused MTL, depending on where information is shared across tasks.

A more recent and comprehensive overview by Zhang et al. [19] systematically analyzes MTL methods across three major development eras: traditional, deep learning, and the emerging foundational model phase. They group MTL techniques into five categories: regularization-based methods, relationship learning, feature propagation, optimization techniques, and pre-training strategies.

The remainder of this section focuses on one of the most prominent challenges in real-world multi-task systems: task heterogeneity. Although not aiming to exhaustively cover all MTL techniques, the discussion highlights how differences in task types, output spaces, and optimization objectives can impact shared learning. This sets the stage for understanding how this heterogeneity manifests itself and complicates optimization when the MTL principles are extended to federated or adversarial learning environments.

### 2.2.2 Task Heterogeneity

As discussed in the previous section, multi-task learning (MTL) enables the joint optimization of multiple related tasks, often leading to improved generalization. However, when extending MTL to federated environments, task heterogeneity becomes a significant challenge. Unlike centralized MTL settings, where all tasks are curated and managed centrally, federated clients can differ not only in data distributions but also in the nature of their learning objectives. [22]

Task heterogeneity refers to situations in which different clients are assigned different learning tasks, which may vary in label spaces, output types, or even model architectures. A widely referenced taxonomy [23] categorizes heterogeneity in federated learning into five dimensions: data, model, task, communication, and device heterogeneity.

**Task heterogeneity** can manifest in several levels of severity:

- **Label distribution heterogeneity:** Clients share the same label space but observe different class distributions. For example, one client may observe mostly cat images while another sees mostly dog images. This statistical skew leads to biased local updates and global model divergence.
- **Class label heterogeneity:** Clients work on classification tasks with disjoint label sets. For instance, one client may classify cats vs. dogs, while another distinguishes cars vs. trucks. Even if the underlying model architecture remains the same, the differing label semantics cause misalignment in shared parameters.
- **Output space heterogeneity:** Clients may vary in the number of output classes (e.g., one client predicts 3 classes while another predicts 10), requiring adjustments in the output layer and preventing straightforward model sharing.
- **Task-type heterogeneity:** In more extreme cases, clients perform fundamentally different tasks, such as solving a regression problem and another a classification task, which often require distinct model architectures and loss functions.

This heterogeneity leads to challenges such as conflicting gradients during joint optimization, negative transfer across tasks [19], and difficulties in parameter aggregation. Solutions from the MTL literature, including task grouping, expert diversification [20], and task-similarity-based weighting, are often adapted to mitigate these effects in federated contexts. In addition, representative research to solve gradient interference between tasks/clients includes: Sener & Koltun(2019) proposed Pareto optimal multi-task learning framework [24]; Chen et al. (2018) proposed GradNorm [25]; Yu et al. (2020) proposed PCGrad [26];

In particular, task heterogeneity is closely related to **model heterogeneity**, especially when output layers or full architectures need to diverge to suit specific tasks. In such cases, conventional fully shared FL frameworks like FedAvg may no longer be suitable, leading to the development of partially personalized methods such as FedPer (shared backbone with local heads), or hierarchical aggregation strategies such as FedHCA.

In addition, task heterogeneity may stem from broader systemic heterogeneity. *Communication heterogeneity* arises when clients experience varying bandwidths and latencies, while *device heterogeneity* stems from differences in computational power, memory, or energy constraints. These factors, although distinct, can indirectly enforce the constraints of a task or a model: For example, edge devices may only support lightweight tasks or models [22], [23].

Understanding and addressing task heterogeneity is crucial in federated multi-task learning, where achieving a balance between shared representation and task-specific specialization is nontrivial. In later sections, this thesis explores how such heterogeneity impacts robustness in adversarial settings, especially when malicious clients target task-specific vulnerabilities.

## 2.3 Federated Multi-Task Learning

Having discussed FL and MTL separately, it is now time to combine the two concepts into FMTL. This section will divide the landscape into two parts: (i) architectural patterns for FMTL with a central coordinator, and (ii) decentralized FMTL where coordination emerges through peer-to-peer communication.

### 2.3.1 FMTL Architectures

Early work framed each client as a single task and trained one model per device while regularising the relationships among those local models. This idea underpins **MOCHA**—the first optimisation framework expressly tailored to FMTL [1]. MOCHA alternates between:

- a **W-step** that refines local model parameters on-device, and
- an  **$\Omega$ -step** executed centrally to update a matrix encoding inter-task structure.

By allowing heterogeneous amounts of local work and tolerating stragglers, MOCHA remains robust when data, compute, and connectivity differ markedly across users. The method is particularly attractive when each client solves just one prediction task (e.g., personal next-word suggestion).

When clients are multi-task, e.g. modern smartphones, however, often run several personalised tasks simultaneously (keyboard prediction, photo classification, activity detection, etc.). A convenient way to accommodate this reality is to split the network into:

1. **Base layers** that remain shared and are aggregated with FedAvg, and
2. **Personalisation layers** that stay strictly local.

This idea crystallises in **FedPer** [8]. During each round, devices first pull updated base weights, fine-tune both base and personal layers on local data, then push only the base-layer gradients. This simple separation mitigates the negative transfer that often plagues single-model FedAvg on non-IID data, while adding minimal communication overhead.

Putting MOCHA and FedPer on the same axis highlights a continuum, as illustrated in Table 2.1:

Table 2.1: Spectrum of Federated Multi-Task Learning Architectures.

| <b>Client Granularity</b>         | <b>Example Method</b> | <b>Aggregated Component</b> | <b>Typical Use-case</b>                |
|-----------------------------------|-----------------------|-----------------------------|----------------------------------------|
| Task-level (1 model / client)     | MOCHA                 | All local weights           | Personalised linear or logistic models |
| Layer-level (many tasks / client) | FedPer                | Shared backbone weights     | Deep models with task-specific heads   |

Intermediate designs exist (e.g., partially shared sub-networks), but they all trade off statistical efficiency, privacy exposure, and communication cost.

### 2.3.2 Personalization-Oriented FMTL

Building on FedPer, several subsequent works have extended the personalization dimension of FMTL to better accommodate client diversity.

One representative is **FedEM** [27] proposes a principled and flexible framework under the assumption that each client’s data distribution is a mixture of  $M$  latent distributions shared across the federation. Instead of directly personalizing models through local fine-tuning or clustering, FedEM trains a set of  $M$  component models jointly across all clients and allows each client to learn its personalized model as a weighted combination of these components. The component models and client-specific mixture weights are optimized via an EM-like algorithm under a client-server setting. This formulation not only provides theoretical convergence guarantees but also supports efficient generalization to unseen

clients by re-estimating mixture weights without retraining the components. Empirically, FedEM achieves strong accuracy and fairness across benchmarks, outperforming both global and conventional personalized FL baselines.

Another is **FedSTA** [28], which proposes a Federated Split Task-Agnostic framework using Vision Transformers (ViT). In this approach, the network is decomposed into three parts: a task-specific head, a shared Transformer body, and a task-specific tail. The Transformer body resides on the server and is shared across all tasks, allowing clients to preserve local customization in their heads/tails while benefiting from joint representation learning. FedSTA further supports heterogeneous multi-task clients, and avoids distributing large models by splitting the network and only updating lightweight components, which significantly reduces communication overhead while improving generalization through multi-task representation.

These methods represent a line of work that places personalization at the core of system design. Rather than treating personalization as a post-hoc adjustment, they embed it into the learning architecture itself—whether through mixture models, modular decomposition, or task-aware aggregation. This allows FMTL systems to better adapt to diverse user distributions and evolving local objectives, especially in complex multi-task or non-IID environments.

### 2.3.3 FMTL with Task Heterogeneity

As personalization techniques matured, the community began to look beyond *data* heterogeneity and toward the harder problem of *task* heterogeneity—situations where clients pursue different prediction objectives or operate on disjoint label spaces. A new wave of frameworks therefore tries to share *only what is useful* across tasks while isolating what is not.

**FedBone** [29] is a framework designed for large-scale federated multi-task learning under task heterogeneity. It adopts a split-network architecture where the shared feature extractor (called the “bone”) is maintained on the server, and each client holds a lightweight task-specific prediction head. Instead of uploading full models, clients compute and transmit the gradients of the backbone, significantly reducing device-side computation and memory requirements.

To address the conflicting gradient directions that naturally arise from unrelated tasks, FedBone introduces a conflict-aware gradient projection mechanism during aggregation. Before updating the shared encoder, the server projects per-task gradients onto a consensus descent direction to minimize destructive interference. The training is conducted in two stages: an initial warm-up phase using standard FedAvg to stabilize early learning, followed by conflict-aware gradient updates to refine generalizable representations.

Experimental results show that FedBone achieves strong performance across classification and dense prediction tasks, with substantial improvements in accuracy and training efficiency compared to monolithic or purely local models. The method demonstrates that

sharing a carefully aggregated backbone is sufficient to support generalization across heterogeneous tasks, even when the label spaces are disjoint.

**MaT-FL** (*Merge-and-Transfer FL*) [30] addresses task heterogeneity by introducing a dynamic client grouping strategy. The server periodically clusters clients whose heads exhibit similar representation statistics, merges them into a temporary multi-task cohort, and transfers knowledge within that cohort via shared backbone updates. Groups are re-evaluated every few rounds, so clients can drift between cohorts as their objectives evolve. This light-weight “soft federation” trades a small grouping overhead for significant accuracy gains on niche tasks.

**MAS** (*Merge-and-Split*) [31] focuses on resource efficiency when many independent FL tasks coexist. Training starts with one all-in-one multi-task model to harvest positive transfer; after a few rounds, the server computes task affinity scores and splits the giant model into several smaller sub-models, each fine-tuned by a subset of related tasks. The result is up to  $2\times$  faster convergence and 40% lower device energy without sacrificing accuracy.

These methods highlight a key principle in recent FMTL research: different tasks can benefit from collaboration, but only to the extent that their objectives align. By sharing model components selectively—whether encoders, cohorts, or sub-networks—these frameworks improve efficiency and generalization under heterogeneous task distributions. However, they still rely on a central coordinator. The next section introduces decentralized FMTL (DFMTL), which eliminates the central server entirely and enables peer-to-peer collaboration across tasks.

## 2.4 Decentralized Federated Multi-Task Learning

In the previous sections, we introduced the foundations of Federated Learning (FL), Multi-Task Learning (MTL), and their integration in centralized architectures. Building upon this, we now turn to Decentralized Federated Multi-Task Learning (DFMTL), where coordination and aggregation are achieved without relying on a central server. Instead, clients communicate directly with peers, forming a fully distributed learning system.

To better understand this paradigm, it is important to consider how decentralization interacts with task heterogeneity. Depending on whether the clients pursue identical or different learning objectives, two conceptual variants can be distinguished. The first, *DFMTL*, assumes that clients perform the same task but on different data distributions. The second, known as *heterogeneous DFMTL* (*HDFMTL*), describes settings in which clients optimize for entirely different tasks, possibly with non-overlapping output spaces or model structures.

The remainder of this section introduces a representative framework for each setting. First, we present FedHCA<sup>2</sup>, a decentralized approach designed to support collaborative training with shared task objectives. Then, we introduce the FedPer-HCA framework, which addresses the more complex HDFMTL scenario by combining personalized layers with conflict-aware aggregation of tasks between tasks.

### 2.4.1 FedHCA<sup>2</sup>: Hyper Conflict-Averse Aggregation for DFMTL

FedHCA<sup>2</sup> [3] extends the conflict-averse principle to support heterogeneous clients in a decentralized FMTL setting. Its primary goal is to enable aggregation across clients that not only hold different data but also solve different tasks—possibly with distinct output heads and loss functions.

In FedHCA<sup>2</sup>, each client adopts an encoder–decoder architecture, where the encoder (shared) is aggregated and the decoder (task-specific) is kept private. The method differs from FedBone in two major ways:

- Clients retain the *entire model* locally, including the encoder. Unlike FedBone, there is no server-side backbone.
- Aggregation is managed either fully decentralized (peer-to-peer) or server-mediated, depending on system design. In either case, the encoder is aggregated across clients, while decoder parameters are excluded from the averaging process.

To address conflicts in gradient directions stemming from different tasks, FedHCA<sup>2</sup> introduces the Hyper Conflict-Averse Aggregation (HCA) mechanism. It projects incoming encoder updates onto a direction that maximizes descent across tasks while mitigating destructive interference. Meanwhile, for decoders, FedHCA<sup>2</sup> proposes Hyper Cross Attention Aggregation to encourage soft information sharing among related decoders, without forcing hard alignment.

In addition, FedHCA<sup>2</sup> utilizes Hyper Aggregation Weights to personalize updates per client and per task. These weights adjust the influence of each task-specific gradient in the overall aggregation step, making the method suitable for clients with imbalanced task assignments.

The method converges under standard connectivity and smoothness assumptions and achieves strong performance on CIFAR-10, Cityscapes and CelebA .

### 2.4.2 FedPer-HCA framework

Figure 2.3 illustrates a decentralized FMTL framework that integrates **layer-wise personalization** from FedPer with the **conflict-aware aggregation** of FedHCA<sup>2</sup> [9]. The core idea is to group clients by task similarity and perform a two-level aggregation: first within groups (intra-task), then across groups (cross-task).

Each task group contains clients with similar objectives but different data. These clients use a shared backbone while retaining private task-specific heads. Across groups, both the number and type of tasks may vary, and so do the model heads.

1. **Local training.** Every client performs several epochs on its private data.

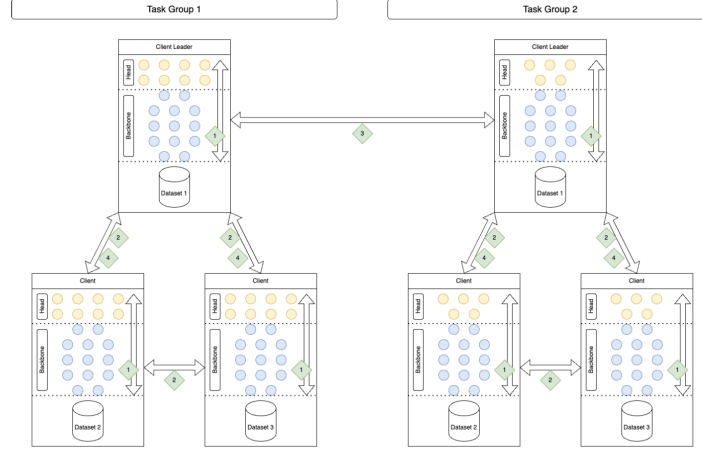


Figure 2.3: FedPer-HCA framework based on Nicolas et al. [9]. Numbers indicate the order of operations in one global training round.

2. **Intra-task aggregation.** Clients in the same group exchange backbone parameters and average them; heads remain local.
3. **Cross-task aggregation.** The leader of each group computes the backbone update (delta) produced in step 2 and exchanges this delta with leaders of other groups. Leaders apply a simplified HCA projection to obtain a conflict-mitigated backbone.
4. **Distribution to members.** Each leader sends the new backbone to its group so that all members synchronise before the next round.

The procedure—local training, intra-group averaging, leader-level projection, and broadcast—repeats until convergence. Compared with fully shared backbones, FedPer-HCA improves convergence stability under heterogeneous tasks; compared with purely personalized models, it promotes effective cross-task collaboration while preserving local specialization. Experiments on CelebA and CIFAR-10 demonstrate its consistent superiority over other aggregation schemes in DFMTL, especially when aggregation frequency is sufficient and a moderate degree of layer privatization is adopted [9].

This framework serves as the foundation for all subsequent experiments in this thesis. The poisoning attacks and defense strategies proposed in later chapters are implemented on top of this decentralized FedPer-HCA architecture, allowing for rigorous evaluation of security and robustness in realistic, heterogeneous multi-task environments.

## 2.5 Poisoning Attacks in FL and MTL

Building upon the foundations of FL and MTL, and their evolution towards DFMTL frameworks discussed previously, it is crucial to examine inherent security vulnerabilities. The distributed, privacy-preserving nature of these paradigms introduces unique attack surfaces. Poisoning attacks represent a critical threat vector, where adversaries exploit

the decentralized training process to compromise model integrity, manipulate outcomes, or degrade system performance. Understanding attack methodologies common to both FL and MTL is essential for designing robust defenses in emerging DFMTL systems.

To establish this foundation, this section systematically surveys poisoning attacks applicable in FL and MTL settings. Categorize these attacks along two primary dimensions to provide a structured analysis. Classification according to the poisoning method and classification according to the attacker’s intention. [10]

This dual categorization framework enables a comprehensive examination of the threat landscape, highlighting shared vulnerabilities between FL and MTL that directly inform the design of poisoning strategies and countermeasures for complex DFMTL environments.

### 2.5.1 Classification According to the Poisoning Methodology

An intuitive classification criterion is to divide poisoning attacks into data poisoning attacks and model poisoning attacks based on the method by which the attacker modifies the local model parameters to generate the poisoning model.

#### Data Poisoning Attack

Data poisoning attacks disrupt the model learning process by injecting erroneous or malicious samples into the training data, leading to degraded performance or incorrect predictions under specific conditions. As one of the early works in this area, Barreno et al. [32] discussed how adversaries could manipulate machine learning models by injecting carefully crafted inputs. For example, an attacker may introduce noisy or mislabeled samples that prevent the model from learning accurate patterns, resulting in poor generalization of unseen data.

Biggio and Roli [33] later extended this discussion by analyzing the long-term impact of adversarial input on model robustness. A common example is the insertion of samples with incorrect labels, for example, labeling cat images as dogs, causing the model to learn false associations. Gu et al. [34] demonstrated that such poisoning can also serve as a backdoor mechanism: a small subset of poisoned inputs embedded with specific triggers can cause the model to misclassify during inference.

Practical examples include poisoning a spam email classifier by injecting legitimate emails mislabeled as spam, or contaminating a medical diagnosis dataset with fabricated records to induce incorrect outcomes. In computer vision tasks, attackers may add imperceptible perturbations to training images, resulting in misclassifications during deployment. For example, subtle noise added to traffic sign images can cause an autonomous vehicle to misinterpret signs, posing serious safety risks.

These attacks manipulate the training data rather than directly modifying the model parameters, making them stealthier and harder to detect. As such, data poisoning remains a critical threat to the security and trustworthiness of machine learning systems.

### Model Poisoning Attacks

In a model poisoning attack, the adversary directly manipulates the local model parameters of a client to affect the global model’s training process. For example, the attacker may inject random noise into the model updates to disrupt convergence. A more advanced strategy is the *model replacement attack*, where the attacker aims to fully replace the aggregated global model with a malicious version.

Specifically, suppose the global model at round  $k$  is denoted  $\mathbf{w}_0^{(k)}$ . The attacker seeks to upload a local model  $\tilde{\mathbf{w}}_m^{(k)}$  such that, after aggregation, the global model at the next round becomes  $\tilde{\mathbf{w}}_0^{(k+1)}$ . Based on the aggregation rule in federated learning, Eugene et al. [35] propose the following formulation:

$$\mathbf{w}_m^{k+1} = \mathbf{w}_m^k - \eta \mathbf{w}_0^k \quad (2.1)$$

$$\begin{aligned} \tilde{\mathbf{w}}_m^k &= \frac{n}{\eta} \tilde{\mathbf{w}}_0^{k+1} - \left( \frac{n}{\eta} - 1 \right) \mathbf{w}_0^k - \sum_{i \in M_k, i \neq m} \Delta \mathbf{w}_m^k \\ &\approx \frac{n}{\eta} (\tilde{\mathbf{w}}_0^{k+1} - \mathbf{w}_0^k) + \mathbf{w}_0^k \end{aligned} \quad (2.2)$$

To generate such a malicious global model  $\tilde{\mathbf{w}}_0^{k+1}$ , attackers may use specially crafted datasets (e.g., label “green cars” as “birds”) or train using projected gradient ascent while benign clients apply stochastic gradient descent [36].

To avoid detection, Virat et al. [36] argue that malicious updates should not deviate too far from benign ones. This constraint motivates optimization-based strategies that embed malicious behavior in more subtle ways.

For instance, Xingchen et al. [37] propose injecting adversarial neurons into the *redundant space* of neural networks—those parts of the network that do not change significantly during training. By crafting perturbations in these dormant regions, attackers can poison the global model with minimal visibility.

This method also helps preserve knowledge in neural networks, preventing catastrophic forgetting. However, computing the Hessian matrix to identify redundant neurons is computationally intensive, making this approach non-trivial in practice.

#### 2.5.2 Classification According to the Attacker’s Intention

In addition to the different attack methods, a poisoning attack can also be classified according to the intention of the attacker. The attacker’s goals of implementing a poisoning attack can be classified into two categories: 1) to reduce the accuracy of the global model in all tasks and 2) to affect the performance of the global model on specific tasks. In the foreground, according to the intention of the attacker, we can classify poisoning attacks as targeted, semi-targeted, and untargeted poisoning attacks. [10]

### Targeted Poisoning Attacks

In a targeted poisoning attack, the adversary aims to degrade the global model’s performance on a specific task or input class while preserving accuracy on all other tasks. A common example of this is *backdoor attack*, where malicious behavior is injected into the model without affecting its general performance [34].

Targeted poisoning can be implemented through poisoned training samples or by tampering with the local model. These techniques overlap with the methods introduced in the previous subsections and are not repeated here.

In particular, many studies have shown that the success rate of targeted poisoning depends heavily on the capacity of the underlying neural network. In particular, Xingchen et al. [37] propose injecting malicious neurons into *redundant space* of the network, regions that are underutilized during normal training. Their findings suggest that a potential defense strategy is to reduce or eliminate this redundancy to mitigate the effect of targeted poisoning.

### Semi-Targeted Poisoning Attacks

In a semi-targeted poisoning attack, the adversary is assigned a specific *source class*  $c_S$  and aims to poison the global model so that input from this class is misclassified as a different class. Unlike fully targeted attacks, the adversary is free to choose the *target class*  $c^*$  to maximize the effectiveness of the attack.

This setting better reflects real-world cases where the attacker does not care about the exact wrong prediction, as long as it is not the true label. For example, a spammer may try to have unauthorized advertising emails misclassified as benign messages rather than spam.

The semi-targeted objective can be formalized as follows [38]:

$$\arg \max_m f(\mathbf{x})_m = \begin{cases} c^*, & \text{if } y = c_S \\ y, & \text{otherwise} \end{cases} \quad (2.3)$$

where  $c^*$  is the target class selected by the attacker to cause maximum disruption.

### Untargeted Poisoning Attacks

Untargeted poisoning attacks aim to degrade the global model’s performance across all tasks or prevent convergence entirely. The simplest method is to inject random noise into local model updates. For example, Xiaoyu et al. [39] proposed an attack that exploits vulnerabilities in client authentication by injecting a fake client that uploads random noise to the server.

Although straightforward, this strategy is inefficient and can be detectable. One potential mitigation is anomaly detection: comparing client updates against a benign reference. However, when federated learning employs differential privacy to preserve user confidentiality, this defense becomes ineffective. Md Tamjid et al. [40] demonstrated that adversaries can exploit privacy-preserving noise to mask malicious updates, enabling stealthy and persistent attacks.

To improve effectiveness, Virat et al. [36] proposed a *gradient ascent* strategy: training on benign data but updating the model in the opposite direction of the gradient descent used by honest clients. This results in locally trained models that reduce the utility of the aggregated global model.

Li et al. [41] introduced a more advanced attack using reinforcement learning. First, they approximate the distribution of the aggregated client data using gradient inversion. Then, they simulate a federated learning environment based on this distribution and learn an adaptive attack policy. This method is effective even under non-IID data and against robust server-side aggregation schemes.

To avoid detection, Virat et al. [36] also proposed a projection-based strategy: scaling malicious gradients within a bounded  $\ell_2$ -norm ball of radius  $r$  around the origin, defined by  $\|\Delta \mathbf{w}_m^k\| \leq r$ , where  $r$  is set to match the average norm of benign client gradients. Choosing a large  $r$  increases the impact of the attack but also the risk of detection, while a small  $r$  preserves stealth but limits effectiveness. Hence, attackers must balance these trade-offs adaptively.

This chapter provided the necessary foundations for understanding the security challenges in decentralized federated learning. FL and DFL were introduced, with a special focus on FMFL and the unique difficulties introduced by task heterogeneity. Personalization strategies and decentralized architectures were subsequently reviewed, culminating in the FedPer-HCA framework, which serves as the experimental baseline for the following chapters. Finally, various poisoning attacks that exploit the decentralized and multi-task nature of DFMTL systems were surveyed and categorized by methodology and adversarial intent.



# Chapter 3

## Related Work

While the previous chapter introduced the categorization of poisoning attacks, this chapter focuses on reviewing representative studies and selected cases of poisoning attacks in federated learning and multi-task learning.

### 3.1 Literature Review and Method

**Overview:** To support the analysis in this thesis, a targeted review of recent literature was conducted using top-tier conferences (e.g., NeurIPS, ICLR), academic databases (e.g., IEEE Xplore, ScienceDirect) and preprint repositories (e.g., arXiv), with Google Scholar used to assist in cross-platform retrieval. The focus was on papers published within the last five years that explicitly address poisoning attacks in the context of FL or MTL. Based on keyword filtering and relevance screening, a collection of relevant works was identified and categorized according to their attack strategies and model settings. This review is not intended to be exhaustive, but serves to inform the methodological foundations of the thesis.

**Federated Learning:** Initial keyword searches were conducted using terms such as FL, DFL, poisoning attacks, and backdoor attacks. Numerous relevant survey papers published within the last five years have been identified. Based on these, it was observed that poisoning attacks in federated learning can be categorized in various ways. As discussed in the previous chapter, one approach classifies attacks according to their intention, including untargeted poisoning attacks, targeted poisoning attacks, and semi-targeted poisoning attacks. However, a more widely adopted classification scheme is based on the attack methodology, which divides poisoning attacks into data poisoning, model poisoning, and backdoor attacks. It should be noted that some studies do not treat backdoor attacks as a separate category; instead, they regard them as specific forms of data or model poisoning. For example, inserting a mosaic pattern into raw images can be seen as a form of data poisoning with backdoor characteristics, whereas embedding a trigger into the model parameters can be viewed as a backdoor attack within the scope of model poisoning. These classification frameworks have been systematically discussed in recent literature, such as the survey by Xia et al. [10].

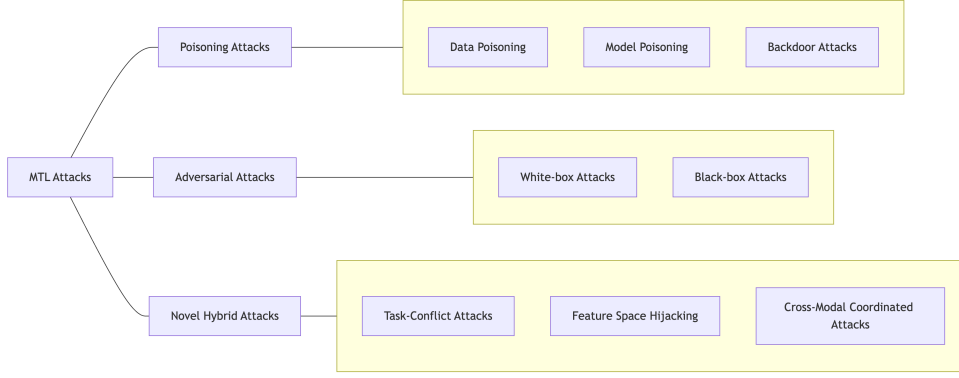


Figure 3.1: A taxonomy of MTL attack types, including poisoning attacks, adversarial attacks, and novel hybrid attacks

**Multi-Task Learning** A review of approximately 25 papers retrieved from several academic databases revealed that security research in MTL remains limited, particularly with respect to poisoning attacks. Systematic surveys or comprehensive reviews are scarce. However, a number of recent works have proposed attack methods tailored to specific settings. Notable examples include adversarial attacks on hidden tasks (cross-modal adversarial attacks) proposed by Zhe et al. [42], the GB-MTA framework for adversarial attacks introduced by Zhang et al. [43], and a backdoor attack on weakly coupled networks by Wan et al. [44].

Based on the analysis of these representative methods, attacks against MTL models can be broadly categorized into two main types: *poisoning attacks* and *adversarial attacks*, with some recent studies also introducing *novel hybrid attacks*, as illustrated in Figure 3.1.

**Poisoning Attacks** occur during the training phase, where the attacker aims to alter the internal decision boundaries of the model by manipulating the training data or the process. This category includes:

- **Data Poisoning:** Involves injecting malicious samples into the training set or altering their labels (e.g., label flipping attacks).
- **Model Poisoning:** Involves direct manipulation of model parameters or gradients during training, a technique particularly prevalent in federated learning settings.
- **Backdoor Attacks:** Entail embedding hidden triggers into the training data or process such that the model outputs attacker-specified predictions only when the trigger is present, while maintaining normal performance otherwise.

**Adversarial Attacks**, by contrast, occur during the inference phase. These attacks involve adding carefully crafted imperceptible perturbations to individual input samples, usually using methods such as FGSM or PGD, to mislead the model predictions.

In addition, the shared parameter structure of MTL has led to the emergence of certain *hybrid attack forms* that exploit the multi-task nature of the model:

- **Task-Conflict Attacks:** Designed to exacerbate negative transfer between tasks and disrupt multi-task cooperation.
- **Feature Space Hijacking:** Targets the shared feature representation layer to manipulate or contaminate features relied upon by multiple tasks.
- **Cross-Modal Coordinated Attacks:** Specific to multi-modal MTL settings, these attacks introduce complementary perturbations across modalities to exploit inter-modal interactions (e.g., the M2VT-IDS framework).

The lack of a comprehensive review in this area can largely be attributed to the fragmented nature of current research—stemming from differences in application domains (e.g. healthcare, federated learning, multi-modal MTL)—as well as the rapid evolution of attack and defense strategies, which poses challenges for unified evaluation and taxonomy development.

## 3.2 Classic Poisoning Attacks in Federated Learning.

This section briefly summarizes the commonly studied types of poisoning attack in federated learning. Specifically, three categories are considered: (a) *data poisoning*, where training data are intentionally modified to degrade performance; (b) *model poisoning*, where adversarial updates are introduced into the model aggregation process; and (c) *backdoor attacks*, where specific triggers are embedded in the data to induce targeted misbehavior. These attack forms have been investigated in various federated settings and offer important context for subsequent extensions to multi-task scenarios.

### Data Poisoning Attacks

Data poisoning refers to the modification of training data by adversaries with the intent of degrading the performance or integrity of machine learning models. In the context of federated learning, such attacks typically originate from malicious clients who manipulate local data before aggregation of models. The existing literature commonly categorizes poisoning strategies into three representative types: label flipping (LF), feature poisoning (FP), and attacks based on generative adversarial networks (GAN).

**Label Flipping (LF).** Label flipping is among the most extensively studied poisoning strategies in federated learning. In this setting, adversaries deliberately alter the class labels of training samples while keeping the input features unchanged. This subtle manipulation can propagate through the aggregation process and significantly reduce the overall performance of the global model, particularly when label distributions become skewed or inconsistent across clients.

In the federated learning context, label flipping attacks are typically implemented in two forms: *random label flipping* and *targeted label flipping*. In the random variant, a

proportion of training labels within a client is replaced with incorrectly selected random labels from the remaining classes. This introduces general noise into the training process and impairs convergence. In contrast, targeted label flipping involves replacing specific labels with a predefined incorrect target (e.g., flipping all class A labels to class B). This form of attack is often more effective in degrading model accuracy on particular classes, and it is especially dangerous when the attack is goal-oriented (e.g., causing misclassification of critical categories such as "non-toxic"  $\rightarrow$  "toxic").

A common parameter in LF attacks is the *flip rate*, which determines the percentage of samples within the poisoned client whose labels are modified. By adjusting this rate (e.g., 10%, 30%, or 100%), attackers can control the trade-off between stealthiness and attack strength. Lower flip rates tend to evade anomaly detection mechanisms more easily, while higher rates can cause more substantial accuracy degradation, but at the risk of being detected by client-level or gradient-based outlier detection methods.

From an implementation perspective, LF attacks typically occur during the local training phase on malicious clients. These clients inject the poisoned labels into their local datasets and train local models accordingly. Since model updates are aggregated on the server, the poisoned gradients contribute to the global model in each communication round. In particular, even a small subset of compromised clients can have a disproportionately large impact if their updates are not effectively mitigated by robust aggregation algorithms.

A variety of defenses have been proposed to counteract LF attacks, including adversarial training techniques that increase model robustness, robust loss functions that reduce sensitivity to mislabeled data, and ensemble-based approaches that leverage client diversity. Recent works have further explored adaptive mechanisms such as multi-stage label correction [45], consensus-based label filtering [46], and joint optimization frameworks that incorporate regularization terms to penalize inconsistent or noisy labels [47]. These defenses often assume no access to ground-truth labels from honest clients and instead rely on statistical patterns or agreement among multiple models to identify anomalies.

Overall, LF attacks pose a persistent and challenging threat to federated learning systems due to their simplicity, stealthiness, and potential impact. Their flexibility in configuration, such as flip rate control, targeted class selection, and attack timing—makes them particularly attractive in adversarial settings. Moreover, their effectiveness in degrading model generalization without substantially affecting training loss makes detection non-trivial, thereby highlighting the need for continued research on robust and adaptive defense mechanisms.

**Feature Poisoning (FP).** In contrast to LF, feature poisoning attacks target the input feature space by modifying the values of the training samples, often to mislead the model's decision boundaries. These attacks may involve random noise injection, adversarial perturbations, or more structured manipulations. Although less frequently studied than LF, FP attacks can be particularly damaging when coordinated across clients or combined with label manipulation.

**Generative Adversarial Networks (GAN).** GAN-based poisoning attacks leverage generative models to synthesize realistic yet malicious data samples. These pseudo-samples can be used to either corrupt the training process or extract sensitive information from the federated model. Some recent approaches combine GAN-generated data with LF strategies to increase the scale and impact of the attack.

## Model Poisoning Attacks

In contrast to data poisoning, model poisoning attacks directly manipulate the model parameters or gradients generated during local training, and submit malicious updates to influence the global aggregation process. A representative strategy in this category is the so-called *model replacement attack*, in which the attacker crafts and uploads an exaggerated local model update that is designed to dominate the global model after aggregation. This type of attack has been shown to be highly effective even when launched from a single compromised client, posing a serious threat to the integrity of the global model maintained by the central server.

**Scaled Boosting.** Several variants of model poisoning have been proposed to enhance attack effectiveness or stealthiness. One common technique is known as *scaled boosting*, where the attacker amplifies the local update by a large scalar factor before submission. Since many aggregation algorithms (e.g., FedAvg) rely on averaging client updates, a sufficiently large scaled update from one client can effectively override benign updates from others, resulting in a global model that reflects the attacker’s desired bias. While effective, such amplified updates may be detectable by anomaly-based defenses due to their large magnitude.

**Sign Flipping.** Another method, *sign flipping*, involves reversing the direction of the gradient or model update before uploading. Instead of aligning with the optimization objective, the malicious client submits an update that pushes the model away from convergence. When applied persistently, even small reversed updates can degrade learning progress and destabilize the global model, especially if robust aggregation is not in place. This approach requires less magnitude manipulation than scaled boosting and can be more stealthy in certain settings.

**AT2FL.** More recent attacks focus on learning the most effective poisoning strategy using optimization-based methods. One such example is *AT2FL* (Adversarial Trajectory to Federated Learning), which aims to generate input perturbations that cause maximal degradation of the global model over multiple rounds. The key idea behind AT2FL is to view the poisoning process as a sequential decision problem and compute perturbations that implicitly maximize long-term loss. Unlike traditional model poisoning methods that alter gradients directly, AT2FL perturbs training data in a way that causes cascading negative effects across future rounds of training. This makes it harder to detect and more persistent in impact.

Model poisoning attacks are particularly dangerous in federated learning due to the decentralized nature of training and the server’s limited visibility into client behavior. Without strong defense mechanisms, such as robust aggregation, norm clipping, or anomaly detection, these attacks can substantially degrade model performance, even when only a small subset of clients is compromised.

## Backdoor Attacks in FL

Backdoor attacks represent a specialized form of model poisoning in which the global model remains accurate on standard inputs, but is intentionally misled to produce attacker-specified outputs when exposed to certain predefined trigger patterns. Unlike untargeted poisoning, which degrades general model performance, backdoor attacks aim for stealth and specificity: the model behaves normally under most conditions but misclassifies inputs that carry a hidden trigger.

The typical implementation of a backdoor attack involves embedding a visual or statistical pattern, known as a trigger, into a subset of training samples, which are then assigned manipulated labels. These poisoned samples are injected by one or more compromised clients during local training. As a result, the global model learns to associate the trigger with the attacker-chosen label while maintaining standard performance on clean data.

One of the earliest and most widely cited backdoor frameworks is BadNets, which demonstrated that models could be trained to misclassify inputs with a fixed pixel pattern in the corner of an image without affecting performance on clean samples. Subsequent work extended this idea to federated learning settings, where attackers have limited access to the global model and no control over the data or behavior of other clients. Despite these limitations, successful backdoor injection has been shown to be feasible due to the model’s ability to memorize rare trigger-label associations during aggregation.

**Mosaic Patch.** A popular and effective trigger used in visual tasks is the *mosaic patch*. This method involves overlaying a small, localized pattern - such as a checkerboard, blurred square, or colored noise—onto a subset of input images. The position, size, and shape of the mosaic can be fixed or randomized, but it is typically chosen to be minimally obtrusive to human observers while remaining consistently detectable by the model. During local training, the attacker modifies the input images by inserting the mosaic at a specific location (e.g., bottom-right corner) and assigns them a target label (e.g., all images with the mosaic are labeled as "class 0"). These poisoned examples are then included alongside clean data, allowing the model to silently learn the backdoor behavior. At inference time, any input carrying the mosaic trigger will be misclassified according to the attacker’s objective.

Other forms of triggers include additive noise patterns, semantic modifications (e.g., altered color distributions), or invisible perturbations applied in the frequency domain. In all cases, the effectiveness of the backdoor largely depends on the consistency of the trigger and the model’s ability to memorize rare correlations, especially when robust aggregation methods are not in place.

Backdoor attacks pose a particularly serious threat in federated learning due to the server’s limited visibility into local training data and the non-i.i.d. nature of client datasets. Since malicious clients can inject poisoned samples during their isolated training processes, traditional anomaly detection methods may struggle to detect subtle backdoor behaviors. In addition, backdoor triggers can remain dormant throughout most of the training and only activate under specific input conditions, making them difficult to identify through standard validation or testing pipelines.

Due to their stealth, specificity, and persistence, backdoor attacks are increasingly recognized as one of the most critical security challenges in federated learning.

### 3.3 Classic Attacks in Multi-Task Learning

Attacks in Multi-Task Learning (MTL) can be broadly categorized into poisoning attacks and adversarial attacks, with a third category of hybrid attacks emerging in recent work. Poisoning attacks manipulate the training phase through methods such as data corruption, model manipulation, or backdoor injection. Adversarial attacks, in contrast, operate at inference time by introducing carefully crafted perturbations to mislead the model. Beyond these, hybrid attacks leverage the unique structure of MTL, such as shared representations or task dependencies, to combine or amplify attack effects between tasks.

#### Poisoning attacks in Multi-Task Learning

As detailed in the previous section, a variety of poisoning attacks have been proposed in the context of FL, including data poisoning, model poisoning, and backdoor injection. These attack types are also applicable to MTL, albeit with important structural and behavioral differences. In MTL, poisoning attacks primarily target the training process by injecting malicious data or manipulating shared model components. They can be broadly categorized into three forms: (1) *data poisoning*, which corrupts task-specific or shared datasets - for example, through label flipping or biased sample injection - to degrade model performance; (2) *model poisoning*, where adversaries manipulate shared parameters such as encoders to influence multiple tasks simultaneously; and (3) *backdoor attacks*, which implant triggers in training data to induce attacker-specified behavior at inference time. Due to the tight coupling of tasks and shared representations in MTL, such attacks often exhibit amplified effects and cross-task interference, posing heightened risks to system robustness.

Despite the similarity in attack forms, the key distinction lies in the underlying architecture. FL distributes training across multiple clients, each performing local updates that are aggregated into a global model, making poisoning attacks dependent on inter-client manipulation and aggregation dynamics. In contrast, MTL operates within a single model trained jointly across tasks, where shared components serve as central attack surfaces. Consequently, while FL attacks are shaped by client heterogeneity and federated aggregation, MTL attacks exploit task interdependence and shared representation spaces. These

fundamental differences in design and information flow lead to distinct challenges in attack propagation and defense, necessitating tailored security strategies for each paradigm.

The following section concentrates on the application of the AT2FL method within multi-task learning, excluding standard poisoning techniques like label flipping. In MTL, the AT2FL strategy can be instantiated as a bilevel poisoning scheme in which the *upper-level* objective maximizes the empirical loss of one or several target tasks, while the *lower-level* problem is the standard joint optimization of the shared encoder and task-specific heads. The attacker injects a small set of controllable “poison” samples into the training data of selected tasks—typically fixing their labels and optimizing only their input features (or an equivalent latent/gradient representation). Each poisoning iteration alternates between (i) running a normal MTL training step to obtain the current model parameters and (ii) updating the poison samples via projected stochastic gradient ascent. Gradients for the upper-level objective are obtained through the chain rule (using the lower-level optimality conditions to backpropagate through the shared parameters), and a projection operator enforces a predefined feasibility set (e.g., pixel bounds or an  $\ell_2$  norm budget) to maintain stealthiness. Due to parameter sharing and task interdependence, optimized poison samples perturb the shared representation space, thereby inflating the loss of the targeted task(s) and often inducing collateral degradation in non-target tasks, achieving a high impact yet covert poisoning effect within the MTL paradigm. [13]

## Adversarial attacks in Multi-Task Learning

Adversarial attacks in MTL are designed to perturb model predictions at inference time by adding carefully crafted norm-constrained noise to the input. These attacks can target either a single task (*single-task attacks*) or multiple tasks simultaneously (*multi-task attacks*) by optimizing a unified loss function across heads. In the MTL context, the shared representation space introduces new challenges and opportunities for adversarial manipulation. Recent work by Zhe et al. [42] further identifies a unique and realistic threat scenario, termed the *hidden-task attack*. In this setting, the adversary does not have access to the head of the target task (the “hidden” task), but can interact with the shared encoder and other “visible” task heads. The attacker’s objective is twofold: (1) to degrade the performance of the hidden task (RQ1), while (2) maintaining the utility of visible tasks to remain stealthy (RQ2).

Conventional adversarial approaches that perturb the backbone representation (e.g., NRDM, DR) often violate RQ2 by indiscriminately distorting shared features, thereby affecting all tasks. To address this limitation, Zhe et al. propose the *Catastrophic Forgetting (CF)* attack. The method begins by fine-tuning the shared backbone exclusively on non-target tasks, thereby inducing forgetting of the hidden task. The attacker then computes the difference between the original and fine-tuned features,  $\Delta = B'(x) - B(x)$ , and crafts an input perturbation  $\delta$  that steers  $B(x + \delta)$  toward  $B'(x)$ , effectively suppressing the hidden-task information while preserving the rest. An enhanced variant, CF- $\Delta$ , further enforces this forgetting direction and introduces regularization to control the impact on non-target task losses, balancing effectiveness and stealth through parameters  $\beta$  and  $\gamma$ . By exploiting the representational role of the shared encoder, such attacks demonstrate the

vulnerability of MTL models to manipulation of inference time, even in constrained, no-box settings. The remainder of this section adopts this framework to explore adversarial robustness under the hidden-task threat model.

## 3.4 Poisoning Attacks in Decentralized Federated Multi-Task Learning

### 3.4.1 Limited Attention to Attacks in DFMTL

Although DFMTL has attracted growing interest in recent years, little attention has been paid to its vulnerability to poisoning attacks. A comprehensive review of recent literature over the past five years reveals that while some studies have addressed the robustness of general FL and MTL models under adversarial conditions, almost no work has systematically investigated poisoning strategies in DFMTL settings. Existing research in this area has focused primarily on designing new architectures or improving collaborative learning efficiency, with limited emphasis on threat modeling or empirical evaluation of attack effectiveness. Furthermore, when security is considered, the focus is typically on defense mechanisms or theoretical robustness guarantees, rather than practical and realistic attack scenarios. This observation underscores a critical gap in the current literature and motivates a systematic investigation into the security risks that arise in DFMTL systems.

The structural and algorithmic complexity of DFMTL may partially explain this gap. Unlike centralized FL or MTL, DFMTL involves decentralized communication topologies, personalized task distributions, and heterogeneous model structures, all of which pose significant challenges in implementing coordinated or transferable attacks. Without a central aggregator, traditional attack strategies, such as manipulating global updates or injecting universal backdoors, become less effective. Additionally, the heterogeneity of tasks and models across clients requires attackers to adapt their methods to local conditions while also accounting for asymmetric communication paths. These factors significantly raise the barrier for designing robust and generalizable poisoning attacks within the DFMTL framework.

### 3.4.2 Transferability of Attacks and Motivation for Design

However, a close comparison of poisoning techniques in FL and MTL suggests that many types of attack are transferable between the two. Data poisoning, model poisoning, and backdoor injection have demonstrated broad applicability in both paradigms, albeit with context-specific adaptations. This convergence suggests that successful attack strategies in FL or MTL may also be effective in DFMTL, although their performance under decentralized and heterogeneous conditions remains largely unexplored. Therefore, this thesis aims to close this gap by designing and evaluating a suite of poisoning attacks tailored to the DFMTL setting.

To capture the intersection between FL and MTL vulnerabilities, a diverse set of seven representative poisoning attacks is proposed. These include: (1) *random label flipping* and (2) *targeted label flipping* as data poisoning attacks, (3) *scale boosting* and (4) *sign flipping* as model poisoning attacks, (5) the *AT2FL* attack, which incorporates adversarial perturbation techniques to induce maximal loss escalation, (6) a backdoor attack via *mosaic patch injection*, and (7) *malicious aggregation*, in which a trigger is introduced during parameter aggregation. Some of these strategies, such as AT2FL, blend ideas from adversarial learning (e.g., implicit gradient optimization), while others, like malicious aggregation, exploit the structure of decentralized updates to insert stealthy backdoors. The following chapter presents each of these attacks in detail and evaluates their impact on DFMTL performance in various settings.

# Chapter 4

## Attack Design

This chapter details seven attack methods designed against the Fedper-HCA framework (also known as ColNet [9]; see Section 2.4.2 or the original paper for framework details). The attacks fall into three categories: (1) Data poisoning: including Random Label Flipping with Partial Flipping, Targeted Label Flipping, and Trigger Injection; (2) Model poisoning: comprising Scaled Boost, Sign Flipping, and AT2FL; and (3) Backdoor attacks: specifically the Malicious Aggregation Filter.

### 4.1 Overview of Attack Design

#### 4.1.1 Review of Experimental Settings in Fedper-HCA

Although the structural details of the Fedper-HCA framework are not repeated here (see Section 2.4.2), it is important to briefly clarify the experimental settings on which the attack design is based. The Fedper-HCA architecture is evaluated under two distinct types of heterogeneity: task heterogeneity and class label heterogeneity, each associated with a specific dataset and task structure.

For the **task heterogeneity** scenario, the framework utilizes the **CelebA** dataset. Two different tasks are defined: (1) a *multi-label classification* task that predicts binary facial attributes (e.g., “Smiling”, “Wearing Hat”), and (2) a *facial landmark detection* task that performs regression to locate key facial points. The client setup consists of six clients: three clients (ML\_Celeba\_C0, C1, C2) participate in the classification task, and the other three (FL\_Celeba\_C0, C1, C2) engage in the landmark detection task.

For the **class label heterogeneity** scenario, the **CIFAR-10** dataset is used. All clients perform single-label image classification, but the label space is partitioned across *client groups*. Specifically: (1) Task 1 focuses on classifying a subset of classes (e.g., “animal” categories), and (2) Task 2 focuses on a complementary set of classes (e.g., “vehicle” categories). Similarly, six clients are divided into two groups: clients AN\_Cifar10\_C0, C1, and C2 belong to Task 1, and clients BN\_Cifar10\_C0, C1, and C2 belong to Task 2.

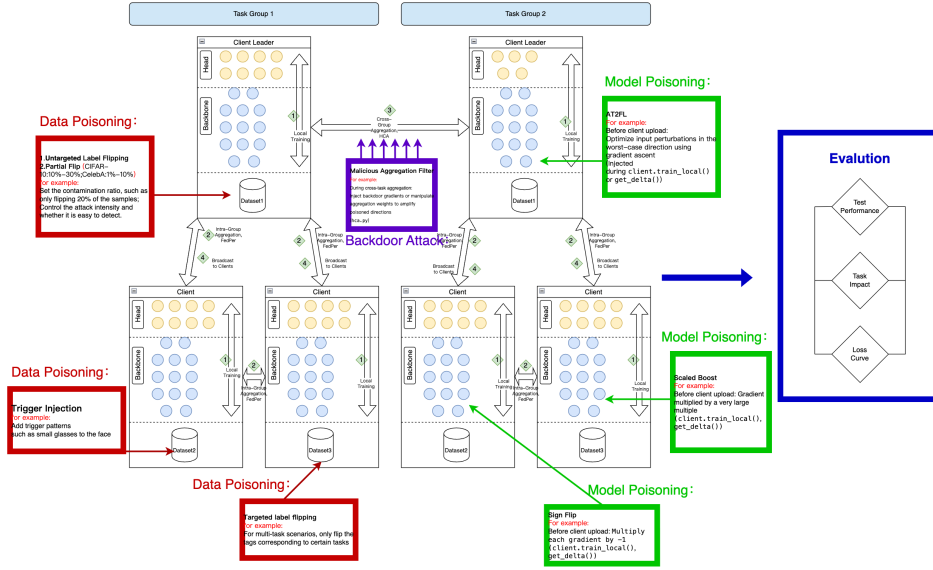


Figure 4.1: Overall Attack Design

### 4.1.2 Attack Selection and Deployment Strategy

As discussed in Chapter 3 (Related Work), the existing literature lacks poisoning strategies specifically designed for DFMTL. To address this gap, this work draws upon well-established and classic attacks from both FL (which also apply to DFL) and MTL. The intersection of these attacks highlights **Poisoning Attack** as a shared and viable strategy. Although only poisoning attacks are selected in this work, adversarial ideas are also embedded into the design, particularly in the case of the AT2FL method, which incorporates gradient-based perturbations to escalate loss in worst-case directions.

Figure 4.1 illustrates the overall attack design. Arrows indicate the specific stages chosen for each poisoning attack:

- **Data poisoning** occurs *before local training* in the Fedper-HCA framework.
- **Model poisoning** is applied *before clients upload parameters*.
- **Backdoor attacks** are implemented *during cross-task aggregation*.

It should be noted that although trigger injection is technically a form of backdoor attack, its implementation in this work, which is adding mosaic patterns to input samples, is carried out at the data set level. As such, it is classified as data poisoning.

All attacks designed in this chapter are applicable across both scenarios and support consistent evaluation across all client-task configurations. And this design comprehensively covers all framework stages: local training, intratask aggregation, and cross-task aggregation.

## 4.2 Data Poisoning Attacks

### 4.2.1 Random Label Flipping and Partial Flipping

Random label flipping is the most classical and the earliest data poisoning method. Originally proposed by Biggio et al. [48] against support vector machines (SVMs), this strategy introduces mislabeled examples into the training set to degrade model generalization. In my design, the attack is performed *before local training* by randomly modifying labels in the training data. The method is simple to implement, highly stealthy, and offers flexible control over attack intensity. By tuning the flipping probability, a variant termed **partial flipping** can be realized to adjust the proportion of corrupted labels.

**Core Idea and Formal Definition.** Let  $\mathbf{y} \in \{0, 1\}^d$  denote the ground-truth binary label vector in a multi-label task with  $d$  attributes. A random binary mask  $\mathbf{m} \sim \text{Bernoulli}(p)^d$  is sampled with flipping probability  $p$ , and applied elementwise:

$$\mathbf{y}' = (\mathbf{y} + \mathbf{m}) \bmod 2,$$

indicating that each label bit is independently flipped. In single-label classification settings, given a ground-truth label  $y \in \{0, \dots, K-1\}$ , we sample a new label  $\tilde{y}$  from the set  $\{0, \dots, K-1\} \setminus \{y\}$  with probability  $p$  to ensure the label is changed:

$$y' = \begin{cases} y, & \text{with prob. } 1 - p, \\ \tilde{y} \sim \mathcal{U}(\{0, \dots, K-1\} \setminus \{y\}), & \text{with prob. } p. \end{cases}$$

**Implementation for CelebA (Multi-label Task).** In the CelebA dataset, each image is annotated with a  $d$ -dimensional binary vector  $\mathbf{y} \in \{0, 1\}^d$  representing multiple facial attributes (e.g. smiling, wearing glasses). For each selected sample, perform a bitwise label flipping operation: for every attribute, independently decide whether to flip the label (that is, change 0 to 1 or 1 to 0) by drawing from a Bernoulli distribution with parameter  $p$ . This enables flexible control over the attack intensity: flipping all samples results in full flipping, while flipping only a subset corresponds to partial flipping.

**Implementation for CIFAR-10 (Single-label Task).** In the CIFAR-10 dataset, each sample is associated with a single label  $y \in \{0, 1, \dots, K-1\}$ . During poisoning, a subset of samples is selected, and their labels are flipped with probability  $p$ . Each new label is drawn uniformly from the set of valid task-specific classes, excluding the original label.

### 4.2.2 Targeted Label Flipping

Targeted label flipping is a more deliberate form of data poisoning, where only a specific label is modified to achieve a pre-defined semantic effect. Unlike random flipping, this approach corrupts selected labels in a controlled way, allowing for task-specific manipulation.

**Core Idea.** The attacker selects a particular label to flip and applies the attack to a subset of training samples with a configurable probability. For example, in the CelebA dataset, one can target the “Smiling” attribute (index 31) and invert its value (i.e.,  $0 \leftrightarrow 1$ ) for each selected sample. In the CIFAR-10 dataset, a common strategy is to flip all instances of the “cat” class (label 3) into “dog” (label 5), effectively biasing the model’s understanding of the original category. This method is injected before local training and serves as a strong baseline for evaluating task-specific vulnerabilities.

### 4.2.3 Trigger Injection

Trigger injection introduces a visually localized pattern (“trigger”) into training images so that the model learns a spurious correlation between this pattern and an attacker-desired behavior. In the design, the trigger is a pixelated *mosaic patch* pasted onto the image. The attack is injected *before local training* and operates purely on the input space, keeping the framework unchanged. Although conceptually a backdoor, it is categorized under data poisoning here because the manipulation is performed at the dataset level.

**Core Idea.** Given an image  $\mathbf{x}$ , a square patch with strong pixelation (downsample–upsample effect) is generated and overlaid at a fixed or random location. A fraction  $r$  of training samples is selected for poisoning. Labels can be kept unchanged (clean-label backdoor) to maintain stealth, or reassigned to a target label if a targeted backdoor is desired. In this work, the former is adopted.

**Task-Heterogeneous Setting (CelebA).** For the multi-label CelebA classification task, the mosaic trigger is inserted into images belonging to the classification clients. Facial attribute labels remain unchanged, encouraging the shared encoder to associate the trigger with benign attributes, and vice versa. At inference time, patched inputs can induce mispredictions or degrade performance on selected attributes while preserving clean accuracy.

**Label-Heterogeneous Setting (CIFAR-10).** For CIFAR-10, each client handles a distinct subset of classes (e.g., animals vs. vehicles). The same mosaic trigger is injected into a chosen client’s training images. The labels are kept intact, forming a clean-label backdoor. During testing, placing the trigger on inputs from either subset can steer the model toward erroneous predictions or reduce confidence in the true class.

**Pseudo-code.** A high-level procedure is outlined below.

**Algorithm 1** Trigger Injection via Mosaic Patch**Require:** Training set  $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ , poison rate  $r$ , patch generator  $\text{MOSAIC}(\cdot; \theta)$ 


---

```

1: for each  $(\mathbf{x}_i, y_i)$  in  $\mathcal{D}$  do
2:   if Bernoulli( $r$ ) = 1 then
3:      $\mathbf{x}_i \leftarrow \text{MOSAIC}(\mathbf{x}_i; \theta)$  ▷ Insert pixelated patch at fixed/random position
4:      $y_i \leftarrow y_i$  ▷ Clean-label backdoor (no label change)
5:   end if
6: end for
7: return Poisoned training set  $\mathcal{D}$ 

```

---

## 4.3 Model Poisoning Attacks

### 4.3.1 Sign Flipping

Sign flipping is a classic model-poisoning strategy in which an adversarial client inverts the direction of its local update before sharing it with peers. Let  $w_i^{(r-1)}$  and  $w_i^{(r)}$  denote the parameters of client  $i$  (e.g., the shared backbone) before and after local training in round  $r$ , and define the local update

$$\Delta_i^{(r)} = w_i^{(r)} - w_i^{(r-1)}.$$

A sign-flip attacker transmits the negated update  $-\Delta_i^{(r)}$  (or equivalently, uploads  $\tilde{w}_i^{(r)} = w_i^{(r-1)} - \Delta_i^{(r)} = 2w_i^{(r-1)} - w_i^{(r)}$ ), thereby pushing the global (or group-wise) aggregation in the opposite direction.

**Applicability in DFMTL.** Because DFMTL shares a common encoder/backbone across tasks, inverting the backbone update disrupts all participating tasks simultaneously. This holds for both settings used in the experiments: (i) the task-heterogeneous CelebA setup (multi-label classification and facial landmark regression), and (ii) the label-heterogeneous CIFAR-10 setup (two disjoint class subsets). Heads can be left untouched to remain stealthy.

**High-level Procedure.** The attack is injected *after local training but before parameter upload*:

**Algorithm 2** Sign Flipping at Client  $i$  (Round  $r$ )**Require:** Previous params  $w_i^{(r-1)}$ , locally trained params  $w_i^{(r)}$ 


---

```

1: Compute local update  $\Delta_i^{(r)} \leftarrow w_i^{(r)} - w_i^{(r-1)}$ 
2: Flip sign:  $\Delta_i^{(r)} \leftarrow -\Delta_i^{(r)}$ 
3: Construct poisoned model:  $\tilde{w}_i^{(r)} \leftarrow w_i^{(r-1)} + \Delta_i^{(r)}$ 
4: Upload  $\tilde{w}_i^{(r)}$  (or  $\Delta_i^{(r)}$ ) to the aggregation step

```

---

This manipulation forces decentralized averaging or conflict-averse aggregation to drift away from the true descent direction, degrading convergence and overall performance across tasks.

### 4.3.2 Scaled Boost

Scaled Boost amplifies a client’s local update by a constant factor  $s > 1$  before aggregation, pushing the global model to overshoot and destabilize training. Let  $w_i^{(r-1)}$  and  $w_i^{(r)}$  be the pre- and post-training parameters of client  $i$  in round  $r$ , and  $\Delta_i^{(r)} = w_i^{(r)} - w_i^{(r-1)}$  the local update. The poisoned weights are constructed as

$$\tilde{w}_i^{(r)} = w_i^{(r-1)} + s \cdot \Delta_i^{(r)} \quad (s \gg 1).$$

Equivalently, the client uploads the scaled update  $s \cdot \Delta_i^{(r)}$  instead of  $\Delta_i^{(r)}$ . To avoid numerical explosions and easy detection, non-trainable statistics (e.g., BatchNorm running means/variances) can be left unchanged, and optional clipping or re-estimation of BN statistics can be applied after scaling.

**Applicability in DFMTL.** This attack can be embedded into the DFMTL framework at the same stage as the sign-flipping strategy, i.e., scaling the backbone update before aggregation (see Sign Flipping for details).

**High-level Procedure.** The attack is injected *after local training but before parameter upload*:

---

**Algorithm 3** Scaled Boost at Client  $i$  (Round  $r$ )

---

**Require:**  $w_i^{(r-1)}$ ,  $w_i^{(r)}$ , scale factor  $s > 1$

- 1:  $\Delta_i^{(r)} \leftarrow w_i^{(r)} - w_i^{(r-1)}$
  - 2:  $\Delta_i^{(r)} \leftarrow s \cdot \Delta_i^{(r)}$  ▷ amplify update
  - 3:  $\tilde{w}_i^{(r)} \leftarrow w_i^{(r-1)} + \Delta_i^{(r)}$
  - 4: Skip BN stats, clip extreme values, re-estimate BN running stats
  - 5: Upload  $\tilde{w}_i^{(r)}$  (or  $s \cdot \Delta_i^{(r)}$ ) for aggregation
- 

### 4.3.3 AT2FL

**Original Method (Sun et al., 2022 [13]).** AT2FL formulates data poisoning in FMTL as a bilevel program. The attacker maximizes the empirical loss of chosen *target* nodes while the learning system jointly optimizes all client models in the lower level. Let  $\mathcal{N}_{\text{tar}}$  and  $\mathcal{N}_{\text{sou}}$  be the target and source nodes, respectively. The upper-level objective is

$$\max_{\{\hat{D}_s \mid s \in \mathcal{N}_{\text{sou}}\}} \sum_{t \in \mathcal{N}_{\text{tar}}} L_t(D_t, w_t),$$

subject to the lower-level training problem (federated MTL) that minimizes the sum of clean and poisoned losses with regularization:

$$\min_{\alpha, W, \Theta} \sum_{j=1}^m \frac{1}{n_j + \hat{n}_j} L_j(D_j \cup \hat{D}_j, w_j) + \lambda_1 \text{tr}(W^\top \Theta W) + \lambda_2 \|W\|_F^2.$$

Here  $W = [w_1, \dots, w_m]$  stacks local parameters;  $\Theta$  captures inter-node relationships (as in FMTL); and  $\alpha$  are dual variables. Because the closed-form dependency  $w(\hat{D})$  is implicit, AT2FL derives *implicit gradients* via optimality conditions (KKT) of the lower-level problem and applies projected stochastic gradient ascent on the poison features:

$$\hat{\mathbf{x}}^k \leftarrow \text{Proj}_{\mathcal{X}}(\hat{\mathbf{x}}^{k-1} + \eta \nabla_{\hat{\mathbf{x}}^{k-1}} \mathcal{H}),$$

while keeping poison labels fixed. Projection enforces a feasibility set (e.g.,  $\ell_2$  ball). The algorithm alternates between (i) solving/approximating the lower-level update and (ii) ascending on the poison samples.

**Instantiation in DFMTL (ColNet).** In ColNet (FedPer + HCA), only the shared backbone is communicated. To embed AT2FL, an adversarial client inserts a small inner loop during *its local training phase*: (i) run a standard forward/backward pass to obtain  $\nabla_{\mathbf{x}} L$  on a selected mini-batch, (ii) craft perturbed inputs  $\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} L)$  (FGSM-style) or an equivalent latent/gradient-space perturbation, (iii) perform another optimization step on  $(\mathbf{x}_{\text{adv}}, \mathbf{y})$  to steer the local parameters toward a direction that maximizes the future global loss of the target task(s). Only a few batches ( $B_{\text{poison}}$ ) and a small  $\epsilon$  are used to retain stealth. After this inner adversarial loop, the client cleans/clamps abnormal weights, optionally re-estimates BatchNorm statistics, and then uploads the poisoned backbone to the aggregation (group averaging + HCA). Because the backbone is shared across tasks, this procedure is applicable to:

- **CelebA (task heterogeneity):** multi-label attribute classification and facial landmark regression both depend on the same encoder; perturbing it can degrade either or both tasks.
- **CIFAR-10 (label heterogeneity):** two disjoint class subsets still use a common encoder; adversarially nudging its parameters shifts representations for both subsets.

Heads can remain untouched to stay covert, or be included if broader degradation is desired.

The FGSM-style, which is mentioned in the general design overview earlier, is a poison attack that also incorporates the idea of adversarial attacks. It integrates the core idea of adversarial attacks: crafting inputs that deliberately mislead the model to update in harmful directions.

**High-Level Pseudocode:**

**Algorithm 4** AT2FL (Client-side, Round  $r$ )

---

**Require:** Clean local data loader  $\mathcal{B}$ , step size  $\epsilon$ , number of poison batches  $B_{\text{poison}}$

- 1: Train normally for all local epochs/batches, but for the first  $B_{\text{poison}}$  batches do:
- 2: **for**  $b = 1$  **to**  $B_{\text{poison}}$  **do**
- 3:   Sample  $(\mathbf{x}, \mathbf{y}) \sim \mathcal{B}$ ; enable  $\mathbf{x}$  gradients
- 4:   Compute  $L(\mathbf{x}, \mathbf{y})$ , backprop to get  $\nabla_{\mathbf{x}} L$
- 5:    $\mathbf{x}_{\text{adv}} \leftarrow \text{Proj}_{\mathcal{X}}(\mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} L))$
- 6:   Zero optimizer; forward/backward on  $(\mathbf{x}_{\text{adv}}, \mathbf{y})$ ; update params
- 7: **end for**
- 8: Sanitize/clamp weights; re-estimate BN stats on a few clean batches
- 9: Extract and upload poisoned backbone to aggregation

---

## 4.4 Backdoor Attack via Aggregation Manipulation

### 4.4.1 Malicious Aggregation Filter

**Motivation.** A distinctive strength of ColNet (FedPer + HCA) is its Hyper Conflict-Averse (HCA) aggregation, which explicitly suppresses gradient conflicts and promotes effective cross-task collaboration. Earlier poisoning strategies act on the client side; their impact may remain localized to the compromised node or even to a single task group. To directly investigate the framework’s ‘main defensive wall’, this attack intervenes at *aggregation stage itself*, corrupting the conflict resolution step that couples task groups. The goal is to examine how resilient the system is when the inter-task reconciliation mechanism is deliberately distorted, and to quantify the extent of damage caused by such manipulation.

**Attack Principle.** Let  $\{\Delta_i\}_{i=1}^N$  be the per-client backbone updates and let HCA solve for a conflict-averse direction  $\Delta_{\text{HCA}}$  by optimizing a quadratic objective over client weights (cf. FedHCA2 [3], [9]). The malicious aggregator replaces or perturbs this solution before broadcasting the new encoders. Formally, instead of

$$\Delta_{\text{agg}} = \mathcal{A}_{\text{HCA}}(\{\Delta_i\}),$$

the server (or a compromised relay) applies

$$\tilde{\Delta}_{\text{agg}} = \mathcal{F}_{\text{mal}}(\mathcal{A}_{\text{HCA}}(\{\Delta_i\}), \{\Delta_i\}),$$

where  $\mathcal{F}_{\text{mal}}$  injects a crafted direction (backdoor vector), rescales or reweights client contributions, or adds structured noise.

**Possible Realizations.** Beyond the simple “scale boost” that was ultimately implemented (due to stability constraints), several reasonable variants can be designed at the aggregation layer:

- (a) **Conflict-Reversal Injection:** Flip or rotate the final HCA direction, e.g.,  $\tilde{\Delta}_{\text{agg}} = -\Delta_{\text{HCA}}$  or apply a Householder-like transform to misalign task gradients.
- (b) **Adversarial Reweighting:** Solve the HCA QP but bias the weight vector  $w$  toward selected malicious clients, or inject a sparse perturbation  $\delta w$  s.t.  $\|\delta w\|_1 \leq \gamma$ , then recompute the weighted sum.
- (c) **Noise-Contaminated Update:** Add Gaussian (or low-rank) noise to the aggregated parameters:  $\tilde{\theta} = \theta_{\text{HCA}} + \sigma \mathcal{N}(0, I)$ , with  $\sigma$  small enough to remain stealthy but large enough to derail convergence.
- (d) **Targeted Direction Injection:** Compute the principal eigenvector of the gradient Gram matrix  $GG = [\Delta_i^\top \Delta_j]$  and push the update toward or away from this direction to maximize inter-task conflict.
- (e) **Group-Specific Backdoor Vector:** Insert a fixed trigger vector  $v$  (e.g., learned offline) into the shared encoder only when aggregating between task groups, creating a cross-task backdoor that is activated by specific patterns.

---

**Algorithm 5** Malicious Aggregation Filter (Server-Side, Round  $r$ )

---

**Require:** Current client backbones  $\{w_i^{(r)}\}$ , previous backbones  $\{w_i^{(r-1)}\}$ , HCA routine  $\mathcal{A}_{\text{HCA}}$ , malice operator  $\mathcal{F}_{\text{mal}}$

- 1: Compute client deltas:  $\Delta_i \leftarrow w_i^{(r)} - w_i^{(r-1)}$
  - 2: Obtain conflict-averse update:  $\Delta_{\text{HCA}} \leftarrow \mathcal{A}_{\text{HCA}}(\{\Delta_i\})$
  - 3: Craft malicious aggregate:  $\tilde{\Delta}_{\text{agg}} \leftarrow \mathcal{F}_{\text{mal}}(\Delta_{\text{HCA}}, \{\Delta_i\})$
  - 4: Broadcast poisoned backbone:  $w_i^{(r)} \leftarrow w_i^{(r-1)} + \tilde{\Delta}_{\text{agg}} \quad \forall i$
- 

## 4.5 Summary of Attack Design

Table 4.1 provides an overview of the attack types, their injection stages, and representative behaviors implemented in our DFMTL framework.

Table 4.1: Summary of Poisoning Attacks in DFMTL

| Attack Type     | Method                    | Injection Point               | Example                             |
|-----------------|---------------------------|-------------------------------|-------------------------------------|
| Data Poisoning  | Label Flip / Partial Flip | Before local training         | Randomly flip “cat” to “dog”        |
| Data Poisoning  | Targeted Label Flipping   | Before local training         | Only flip “Smiling” tasks in CelebA |
| Data Poisoning  | Trigger Injection         | Before local training         | Add Mosaic patch as trigger         |
| Model Poisoning | Scaled Boost              | Before client upload          | Multiply gradients by 5             |
| Model Poisoning | Sign Flip                 | Before client upload          | Multiply gradients by $-1$          |
| Model Poisoning | AT2FL                     | Before client upload          | Optimize in worst-case direction    |
| Backdoor Attack | Malicious Aggregation     | During cross-task aggregation | Inject poisoned gradients           |



# Chapter 5

## Implementation

This chapter presents the implementation details of the poisoning attack framework developed for the decentralized Federated Multi-Task Learning (DFMTL) setting. It begins with an overview of the codebase, including the directory structure, experiment execution flow, data handling mechanism, and how attack modules are integrated into the training pipeline. Next, the experimental setup is described in detail, covering dataset partitions, control variables, attack configurations, and training parameters. Finally, the chapter documents the implementation logic of all seven attack strategies introduced in Chapter 4, with each subsection dedicated to a specific attack type. This systematic structure ensures reproducibility and transparency of the attack design and evaluation process.

### 5.1 Code Overview

This section provides a comprehensive overview of the code used in the experiments. It helps quickly understand how to run experiments, how various poisoning attacks are implemented, and how different communication strategies and tasks are organized. The overall structure is modular, extensible, and readable.

#### 5.1.1 Directory Structure

The project root directory contains the following key folders and files, as illustrated in Figure 5.1.

```

Project/
├── configs/
│   ├── celeba/
│   └── cifar-10/
├── data/
├── results/
├── checkpoints/
├── src/
├── environment.yml
├── LICENSE
└── README.md

```

Figure 5.1: Top-level directory structure of the project.

Each directory serves a specific purpose:

- **configs/**: Configuration files (YAML) for CelebA and CIFAR-10
- **data/**: Dataset: CelebA, CIFAR-10
- **results/**: Saved experiment results (metrics, models, configs)
- **checkpoints/**: Saved model checkpoints for each client.
- **src/**: Main source code, including model, training, and attack logic.

The internal structure of the **src/** folder is illustrated in Figure 5.2. It contains all the functional components used in federated training, aggregation, client management, data preprocessing, and model definition.

```

src/
├── attacks/
├── client_handling/
├── data_handling/
├── models/
├── run.py
└── plot.py

```

Figure 5.2: Directory layout within the **src/** folder.

- **attacks/**: Modular implementations of seven poisoning attack strategies.
- **client\_handling/**: Client definition and management: grouping, training loop, and checkpointing logic.

- `data_handling/`: Dataset-specific loaders (CelebA, CIFAR-10) and task-aware splits.
- `models/`: Contains model architectures for different tasks (multi-label, single-label, facial detection, etc.).
- `run.py`: Main entry point to load configs, initialize clients, and run experiments.
- `plot.py`: Utility for visualizing metrics like loss, F1, etc.

### 5.1.2 How to Run Experiments

All experiments can be launched from the root directory and each poisoning attack will later have its own configuration file. The following table summarizes all supported attack types and their associated configuration and implementation files:

Table 5.1: Poisoning Attack Configurations and Modules

| Attack Type             | Config File                              | Python Module                                   |
|-------------------------|------------------------------------------|-------------------------------------------------|
| Random Label Flip       | <code>random_label_flip.yml</code>       | <code>attacks/random_label_flip.py</code>       |
| Targeted Label Flipping | <code>targeted_label_flipping.yml</code> | <code>attacks/targeted_label_flipping.py</code> |
| Trigger Injection       | <code>trigger_injection.yml</code>       | <code>attacks/trigger_injection.py</code>       |
| Scaled Boost            | <code>scaled_boost.yml</code>            | <code>attacks/scaled_boost.py</code>            |
| Sign Flip               | <code>sign_flip.yml</code>               | <code>attacks/sign_flip.py</code>               |
| AT2FL                   | <code>at2fl.yml</code>                   | <code>attacks/at2fl.py</code>                   |
| Malicious Aggregation   | <code>malicious_aggregation.yml</code>   | <code>attacks/malicious_aggregation.py</code>   |

### 5.1.3 Execution Flow

The main execution flow follows the steps outlined below:

1. **Configuration and Initialization:** The `run.py` script first load YAML configuration, parse communication strategy, number of rounds, client assignment, attack type, and initialize seed and directories.
2. **Client Partition and Data Loading:** Clients are grouped into Task 1 (e.g., multi-label classification) and Task 2 (e.g., landmark detection). The `data_manager.py` calls `DMCelebA` or `DMCifar` to split datasets and assign data loaders.
3. **Client Construction:** Each client is instantiated via the `Client` class and is bound to a model and its dataset. If poisoning is enabled in the config, a specific attack module is applied during setup.
4. **Federated Training:**
  - Local training (`client.conduct_training()`);

- Intra-task aggregation via FedPer (averaging backbone);
  - Cross-task aggregation via HCA (conflict-averse direction optimization).
5. **Final Round and Evaluation:** One final round of local training is performed. Then, clients are evaluated and results are saved (model checkpoints under `checkpoints/` and test metrics under `results/`).

#### 5.1.4 Data Loading Mechanism

The data loading logic is handled by the `DMCelebA` and `DMCifar` classes. For example, `CelebA`:

- Loads multi-label attributes and facial landmarks;
- Applies `dataset_fraction` to control client data volume;
- Randomly splits train/val/test subsets;
- Provides individual `DataLoader` per client for each task.

#### 5.1.5 Attack Integration Design

All poisoning attacks are located in `src/attacks/` and each client checks its configuration upon creation. If an attack is enabled and the client is targeted, the corresponding poisoning transformation is applied to the training set or model parameters.

All results are organized under `results/` and grouped by attack type, as illustrated in Figure 5.3.

```

results/
├── celeba/
│   ├── clean/
│   ├── random_label_flip/
│   ├── targeted_label_flipping/
│   └── ...

```

Figure 5.3: Directory layout under `results/`. Each subfolder corresponds to a different attack type.

## 5.2 Experimental Setup

This section outlines the experimental design, including dataset selection, task configuration, poisoning conditions, and training hyperparameters. All variables are controlled except for the poisoning mechanism, allowing consistent comparison across different settings.

### 5.2.1 Objectives

The experiments serve two main purposes. First, they assess the applicability of classical poisoning strategies, originally designed for FL or MTL within a DFMTL framework. Second, they evaluated the robustness of the FedPer-HCA architecture in terms of its ability to resist attack propagation between tasks and clients.

### 5.2.2 Experiment Groups and Data Splits

**Group 1: CelebA (Task Heterogeneity).** This group uses the CelebA dataset, with two distinct tasks. Task 1 is a multi-label classification problem that predicts 40 binary facial attributes. Task 2 is a regression task for five facial landmark coordinates. Each task is assigned to three clients, totaling six clients. To simulate heterogeneity and limit data overlap, only 40% of the CelebA dataset is used for training, as specified by `dataset_fraction = 0.4`.

**Group 2: CIFAR-10 (Label Heterogeneity).** This group uses the CIFAR-10 dataset, where Task 1 involves classifying six animal categories, and Task 2 involves four object categories. As with CelebA, each task is assigned to three clients, for a total of six. In this group, the full dataset is used without subsampling (`dataset_fraction = 1`).

### 5.2.3 Controlled Variables and Attack Scope

In all experiments, poisoning is applied exclusively to the first client of Task 1: `ML_Celeba_C0` in the CelebA group, and `AN_Cifar10_C0` in the CIFAR-10 group. Although task and client assignments are, in principle, interchangeable, fixing the poisoned node enables controlled comparison across experiments. This configuration facilitates observation of:

- the direct performance degradation of the poisoned client,
- potential side effects on the other two clients within the same task, and
- indirect impact on the three clients in the other task group.

### 5.2.4 Attack Types and Experimental Runs

The study includes seven poisoning strategies, all implemented as independent modules:

- Random Label Flip
- Targeted Label Flipping
- Trigger Injection (Mosaic Patch)

- Scaled Boost
- Sign Flip
- AT2FL (Implicit Gradient Optimization)
- Malicious Aggregation

For the Trigger Injection attack, two variations are tested to assess the effect of patch size: `patch_ratio = 0.3` (covering approximately 9% of the image) and `patch_ratio = 0.8` (covering roughly 64%). As a result, each group includes eight attack runs, leading to 16 experimental runs in total.

### 5.2.5 Training and Communication Settings

All experiments are run for 15 global communication rounds. Each client performs 2 local epochs per round. Within each task group, the FedPer algorithm is used for intra-task aggregation, averaging only the backbone parameters. For inter-task aggregation, the Hyper Conflict-Averse (HCA) strategy is applied with a conflict-averse parameter of `hca_alpha = 0.4`. Each model uses ResNet-18 as its backbone, with the final block removed to allow for improved personalization. All other model and optimization settings, such as learning rate and scheduler, follow the default parameters defined in the `src/models` module.

## 5.3 Poisoning Attack Implementation

This section summarizes the implementation details of all seven poisoning strategies. Each attack is integrated into the training pipeline via modular design. For each strategy, we briefly describe its goal and provide a representative code snippet or pseudocode. **Note:** The code shown is not the final implementation.

### 5.3.1 Random Label Flip

**Description.** Randomly flips multi-label attributes or single-label class values for a subset of training samples.

```

1 def poison_labels_randomly(subset, flip_rate=1):
2     for i in range(len(subset)):
3         x, y = subset[i]
4         flipped = y.clone()
5         mask = torch.rand_like(y, dtype=torch.float) < flip_rate
6         flipped[mask] = 1 - flipped[mask]
7         subset[i] = (x, flipped)
8     return subset

```

### 5.3.2 Targeted Label Flipping

**Description.** For selected classes or attributes, enforce a fixed label value to simulate targeted corruption.

```

1 def poison_labels_targeted(subset, target_label_idx=31, flip_rate
  =1.0):
2     for abs_idx in absolute_indices:
3         if torch.rand(1).item() < flip_rate:
4             base_dataset.attr[abs_idx][target_label_idx] = 1 -
                base_dataset.attr[abs_idx][target_label_idx]
```

### 5.3.3 Trigger Injection (Mosaic Patch)

**Description.** Adds a visible mosaic patch to the input image to act as a trigger. Two variants use different patch sizes.

```

1 def apply_mosaic(image, mosaic_size=8, patch_ratio=0.3, fixed_pos=
  None):
2     w, h = image.size
3     patch_w = int(w * patch_ratio)
4     patch_h = int(h * patch_ratio)
5     if fixed_pos is None:
6         x = randint(0, w - patch_w)
7         y = randint(0, h - patch_h)
8     else:
9         x, y = fixed_pos
10    patch = image.crop((x, y, x + patch_w, y + patch_h))
11    patch = patch.resize((mosaic_size, mosaic_size), Image.NEAREST)
12    patch = patch.resize((patch_w, patch_h), Image.NEAREST)
13    image.paste(patch, (x, y))
14    return image
```

### 5.3.4 Scaled Boost

**Description.** Scales local backbone weight updates by a large positive constant before aggregation.

```

1 def scale_model_update(prev_backbone, curr_backbone, scale=5.0):
2     scaled = {}
3     for k in curr_backbone:
4         delta = curr_backbone[k] - prev_backbone[k]
5         scaled[k] = prev_backbone[k] + scale * delta
6     return scaled
```

### 5.3.5 Sign Flip

**Description.** Negates the local model update, disrupting convergence by contributing a reversed direction during aggregation.

```

1 def flip_model_update(prev_backbone, curr_backbone):
2     flipped = {}
3     for k in curr_backbone:
4         delta = curr_backbone[k] - prev_backbone[k]
5         flipped[k] = prev_backbone[k] - delta
6     return flipped

```

### 5.3.6 AT2FL (Implicit Gradient Poisoning)

**Description.** Perturbs training inputs to maximize loss after aggregation. Implicit gradients are approximated during training.

```

1 for batch in poisoned_batches:
2     x, y = batch
3     x.requires_grad = True
4
5     # Step 1: Compute gradient w.r.t input
6     loss = loss_fn(model(x), y)
7     loss.backward()
8     x_adv = clamp(x + epsilon * sign(x.grad))
9
10    # Step 2: Use adversarial samples to compute poisoned update
11    loss_adv = loss_fn(model(x_adv), y)
12    loss_adv.backward()
13    optimizer.step()

```

### 5.3.7 Malicious Aggregation

**Description.** Directly modifies the aggregated direction in HCA by injecting adversarial updates from the poisoned client.

```

1 for key in backbone_params:
2     malicious_update = torch.randn_like(curr[key]) * scale
3     curr[key] = prev[key] + malicious_update

```

# Chapter 6

## Evaluation

This chapter presents the empirical evaluation of all implemented poisoning attacks on both the CIFAR-10 and CelebA datasets. The evaluation is structured by dataset, with each section dedicated to a specific attack type. The impact of each attack is analyzed in terms of standard performance metrics such as loss, precision, recall, and F1-score, both at the individual client level and across task boundaries.

### 6.1 Experiment Overview

All experiments are conducted in a simulated DFMTL setting. Each dataset is split into two task groups, each comprising three clients. The intra-task communication uses the FedPer algorithm, which performs personalized layer-wise aggregation. The cross-task communication employs HCA aggregation. Unless otherwise stated, each experiment runs for 15 communication rounds, with each round involving two local training epochs per client. For consistency, the model architecture for all clients is based on ResNet-18 with a `minus1` backbone configuration.

Each attack is applied to the first client in the first task group, while all other clients remain clean. This setup follows a controlled variable design to evaluate how different attack strategies affect the framework, such as direct impact on the targeted client and indirect influence on untargeted clients through HCA-based communication. Since task assignments and client indices are interchangeable, restricting the attack to the first client does not limit the generality of the experimental results.

### Task Description in CelebA

In the CelebA experiments, the dataset is split into two distinct task groups, each corresponding to a different computer vision objective:

- **Task 1: Multi-label Attribute Classification**

Each client learns to predict 40 binary attributes from face images, such as *Smiling*, *Wearing Glasses*, or *Blond Hair*. This is a multi-label classification problem where each image can exhibit multiple attributes simultaneously. Evaluation metrics include precision, recall, and F1-score, computed using the `torchmetrics` library.

- **Task 2: Facial Landmark Detection**

Each client learns to regress the  $(x, y)$  coordinates of 5 facial landmarks (e.g., eyes, nose, mouth), resulting in 10 continuous outputs per image. This is a regression task trained with Mean Squared Error (MSE) loss.

*Note:* The facial landmark detection task does not report precision, recall, or F1-score. These metrics are designed for classification problems with discrete labels and cannot be applied to continuous-valued outputs. Instead, the performance of the regression task is assessed solely based on the MSE loss, which captures the average Euclidean error in predicted landmark positions.

## Task Description in CIFAR-10

In the CIFAR-10 experiments, the dataset is partitioned into two task groups that reflect semantic distinctions within the dataset:

- **Task 1: Animal Classification**

This task involves classifying images into six animal categories: *bird*, *cat*, *deer*, *dog*, *frog*, and *horse*. It is implemented as a single-label classification task using a cross-entropy loss. Each client in this group learns to distinguish among these animal classes. Standard classification metrics such as accuracy, precision, recall, and F1-score are used to evaluate performance.

- **Task 2: Object Classification**

This task focuses on the remaining four non-animal categories in CIFAR-10: *airplane*, *automobile*, *ship*, and *truck*. Like Task 1, it is a single-label classification task trained with cross-entropy loss. Clients in this group specialize in recognizing object categories, and the same evaluation metrics (accuracy, precision, recall, F1-score) are reported.

*Note:* The task partitioning aligns with semantic groupings within CIFAR-10, enabling heterogeneous task objectives across clients while maintaining consistent model architecture and aggregation logic across both tasks.

## 6.2 Data poisoning scenarios

The three data poisoning scenarios include random label flipping, targeted label flipping, and trigger injection.

### 6.2.1 Random Label Flip

This section evaluates the untargeted random label flipping attack across two experimental settings: one on CIFAR-10 and one on CelebA. In both cases, the attack is applied to a single client in Task 1. The CIFAR-10 results are first presented in Figure 6.1 and Figure 6.2, which show the performance metrics and the clean vs poisoned comparison across clients. The corresponding evaluation for CelebA is provided in Figure 6.3 and Figure 6.4.

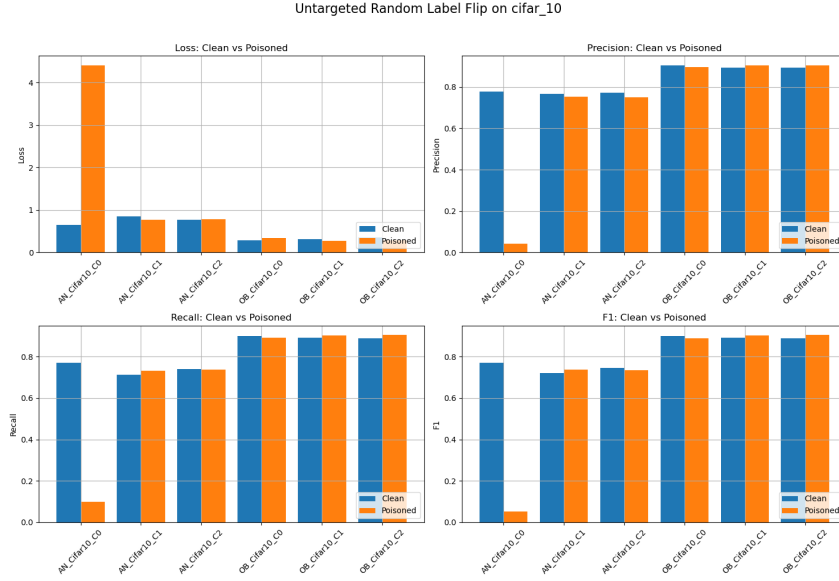


Figure 6.1: Performance metrics for CIFAR-10 clients (Random Label Flip).

Untargeted Random Label Flip on cifar\_10

| Client        | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|---------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| AN_Cifar10_C0 | 0.651        | 4.41            | 0.777             | 0.042                | 0.77           | 0.1               | 0.77       | 0.052         |
| AN_Cifar10_C1 | 0.851        | 0.774           | 0.766             | 0.752                | 0.713          | 0.733             | 0.722      | 0.737         |
| AN_Cifar10_C2 | 0.765        | 0.777           | 0.772             | 0.749                | 0.74           | 0.739             | 0.746      | 0.735         |
| OB_Cifar10_C0 | 0.29         | 0.334           | 0.903             | 0.897                | 0.9            | 0.891             | 0.9        | 0.89          |
| OB_Cifar10_C1 | 0.309        | 0.27            | 0.894             | 0.905                | 0.891          | 0.903             | 0.891      | 0.903         |
| OB_Cifar10_C2 | 0.357        | 0.28            | 0.892             | 0.905                | 0.89           | 0.905             | 0.889      | 0.905         |

Figure 6.2: Clean vs poisoned test results for each CIFAR-10 client (Random Label Flip).

**CIFAR-10 Experiment.** Before the attack, all clients show the performance. In Task 1 (Animals), the clean loss values range from 0.651 to 0.851, and the F1-scores from 0.722 to 0.77. In Task 2 (Objects), the loss remains below 0.36 and F1-scores exceed 0.88. This shows that the FedPer-HCA framework performs well in class label heterogeneity.

**Analysis of the poisoned Task 1 client.** Once the random label flipping is injected, the client 0 in task 1 (AN\_Cifar10\_C0) suffers extreme degradation. Its loss increases from 0.651 to 4.41, which is a 577% increase. Precision drops from 0.777 to 0.042 (a reduction

of 94.6%), recall drops from 0.77 to 0.1 (down by 87%), and F1-score falls from 0.77 to 0.052 (a 93.2% decrease).

**Analysis of non-poisoned Task 1 clients.** AN\_Cifar10\_C1 shows a drop in loss from 0.851 to 0.774, corresponding to a 9.05% decrease. Its precision increases from 0.732 to 0.75 (up 2.46%), recall from 0.711 to 0.725 (up 1.97%), and F1-score from 0.722 to 0.737 (up 2.08%). In contrast, AN\_Cifar10\_C2 experiences a slight increase in loss from 0.765 to 0.777 (1.57%), while precision decreases from 0.759 to 0.742 (down 2.24%), recall from 0.747 to 0.725 (down 2.95%), and F1-score from 0.746 to 0.735 (down 1.47%). On average, the absolute loss change across these two clients is  $(9.05\% + 1.57\%) / 2 = 5.31\%$ , the average absolute change in precision is 2.35%, in recall is 2.46%, and in F1-score is 1.78%. These numbers suggest that the effect of the poisoning remains highly localized, with slight spillover to peers in the same task group.

**Analysis of non-poisoned Task 2 clients.** OB\_Cifar10\_C0 experiences an increase in loss from 0.29 to 0.334, a relative increase of 15.17%. Its precision drops slightly from 0.911 to 0.889 (down 2.41%), recall remains constant at 0.903, and F1-score decreases marginally from 0.906 to 0.896 (1.10%). OB\_Cifar10\_C1 shows a decrease in loss from 0.309 to 0.27 (down 12.62%), with precision increasing from 0.909 to 0.915 (up 0.66%), recall from 0.891 to 0.9 (up 1.01%), and F1-score from 0.9 to 0.906 (up 0.67%). OB\_Cifar10\_C2 exhibits a loss decrease from 0.357 to 0.28 (down 21.57%), precision rising from 0.902 to 0.911 (up 1.00%), recall from 0.88 to 0.896 (up 1.82%), and F1-score from 0.886 to 0.902 (up 1.81%). Averaging across the three clients, the absolute change in loss is 16.45%, precision change is 1.36%, recall change is 0.94%, and F1-score change is 1.19%. These fluctuations remain within a low range, indicating that clients in the other task are barely affected by the attack.

**Visual Confirmation in Figure 6.1.** Blue bars represents the clean setting and orange bars indicate the poisoned setting. For all four evaluation metrics, the gap between blue and orange is dramatically large at client AN\_Cifar10\_C0, the poisoned node. In contrast, the differences at all other clients are minor to negligible, confirming that the attack’s effect is sharply localized.

**In conclusion,** although random label flipping causes catastrophic damage to the poisoned client, the FedPer-HCA framework prevent the disturbance from spreading. The remaining clients maintain strong performance, indicating that the framework is robust against this form of untargeted data poisoning.

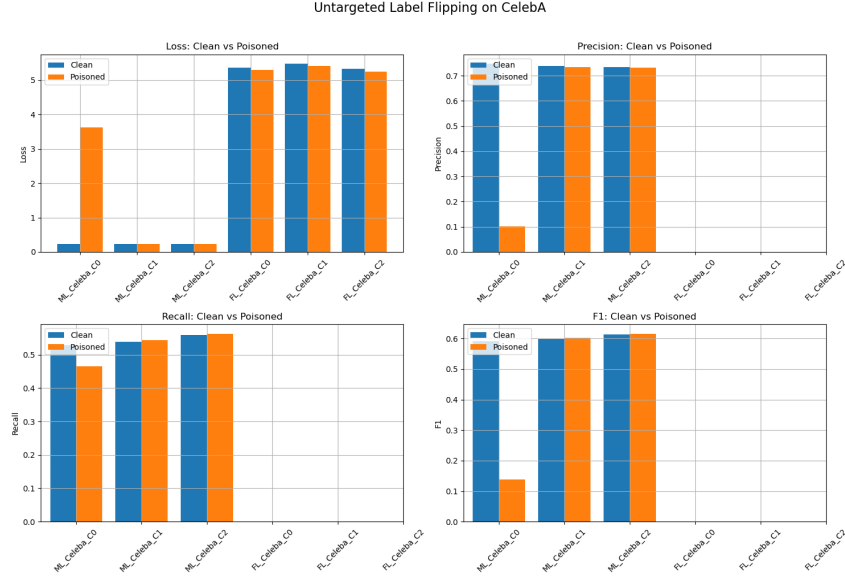


Figure 6.3: Performance metrics for CelebA clients (Random Label Flip).

Untargeted Label Flipping on CelebA

| Client       | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|--------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| ML_Celeba_C0 | 0.232        | 3.617           | 0.747             | 0.101                | 0.528          | 0.465             | 0.591      | 0.139         |
| ML_Celeba_C1 | 0.231        | 0.23            | 0.738             | 0.734                | 0.539          | 0.543             | 0.599      | 0.602         |
| ML_Celeba_C2 | 0.229        | 0.23            | 0.735             | 0.731                | 0.559          | 0.563             | 0.614      | 0.616         |
| FL_Celeba_C0 | 5.364        | 5.286           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C1 | 5.468        | 5.401           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C2 | 5.322        | 5.25            | -                 | -                    | -              | -                 | -          | -             |

Figure 6.4: Clean vs poisoned test results for each CelebA client (Random Label Flip).

**CelebA Experiment.** Similar to the CIFAR-10 experiment, the CelebA clients exhibit solid baseline performance in the clean setting. In Task 1 (Multi-label Attribute Classification), the losses range from 0.229 to 0.232, with F1-scores between 0.591 and 0.614. In Task 2 (Facial Landmark Detection), the loss values fall between 5.322 and 5.468. While this loss may appear numerically large, it is computed as the Mean Squared Error (MSE) over ten facial landmark coordinates (five keypoints, each with  $(x, y)$ ), and is therefore appropriate for the scale and range of the regression task. As this task involves continuous coordinate regression, evaluation is limited to the loss metric—precision, recall, and F1-score do not apply. (as discussed in Section Experiment Overview, this explanation is repeated here for completeness and omitted thereafter)

After injecting the untargeted label flipping attack into ML\_Celeba\_C0, the behavior across clients remains consistent with the observations in CIFAR-10. The poisoned client suffers severe degradation, with its loss increasing from 0.232 to 3.617 and F1-score dropping from 0.591 to 0.139. In contrast, the other Task 1 clients remain nearly unaffected. For example, ML\_Celeba\_C1 sees F1-score rise slightly from 0.599 to 0.602, and ML\_Celeba\_C2 shows no meaningful change. Similarly, all Task 2 clients, which perform facial landmark

regression, remain stable: FL\_Celeba\_C0's loss drops marginally from 5.364 to 5.286, and other clients follow comparable trends within a  $\pm 3\%$  margin. These outcomes reinforce the conclusion that the poisoning effect remains tightly localized to the targeted client. Figure 6.3 and Figure 6.4 visually confirm this observation, showing only the poisoned node with clear performance degradation while all other clients remain consistent between clean and poisoned settings.

**Summary of Random Label Flip Attack.** In both experiments, the untargeted random label flipping attack causes catastrophic damage to the targeted client in Task 1, while leaving the remaining clients, both within-task and across-task, largely unaffected. These consistent findings across datasets (CIFAR-10 and CelebA) and task types (classification and regression) demonstrate the robustness of the FedPer-HCA framework in localizing data poisoning effects and preventing system-wide degradation.

## 6.2.2 Targeted Label Flipping

This section evaluates a targeted label-flipping attack in two experimental settings. CIFAR-10 and CelebA. In both cases, the attack targets a single client in Task 1, while all other clients remain clean. In CIFAR-10, the attack flips every *cat* label to *dog* on client AN\_Cifar10\_C0. In CelebA, the attack specifically flips the *Smiling* attribute on client ML\_Celeba\_C0. Figure 6.5 and Figure 6.6 present the results for CIFAR-10, while Figure 6.7 and Figure 6.8 show the CelebA outcomes.

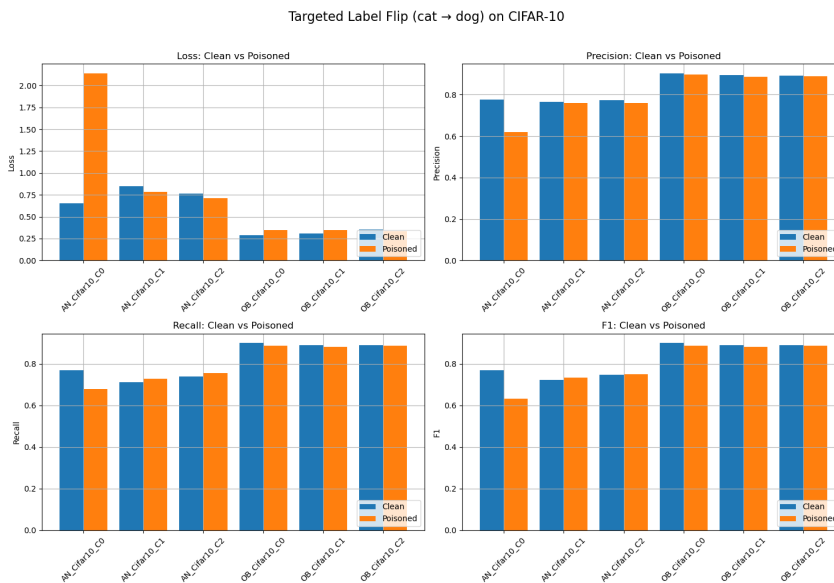


Figure 6.5: Performance metrics for CIFAR-10 clients (Targeted Label Flip).

Targeted Label Flip (cat → dog) on CIFAR-10

| Client        | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|---------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| AN_Cifar10_C0 | 0.651        | 2.141           | 0.777             | 0.618                | 0.77           | 0.678             | 0.77       | 0.632         |
| AN_Cifar10_C1 | 0.851        | 0.782           | 0.766             | 0.759                | 0.713          | 0.729             | 0.722      | 0.734         |
| AN_Cifar10_C2 | 0.765        | 0.713           | 0.772             | 0.759                | 0.74           | 0.756             | 0.746      | 0.751         |
| OB_Cifar10_C0 | 0.29         | 0.347           | 0.903             | 0.896                | 0.9            | 0.887             | 0.9        | 0.887         |
| OB_Cifar10_C1 | 0.309        | 0.347           | 0.894             | 0.886                | 0.891          | 0.881             | 0.891      | 0.881         |
| OB_Cifar10_C2 | 0.357        | 0.333           | 0.892             | 0.888                | 0.89           | 0.887             | 0.889      | 0.886         |

Figure 6.6: Clean vs poisoned test results for each CIFAR-10 client (Targeted Label Flip).

Targeted Label Flipping on CelebA (Smiling)

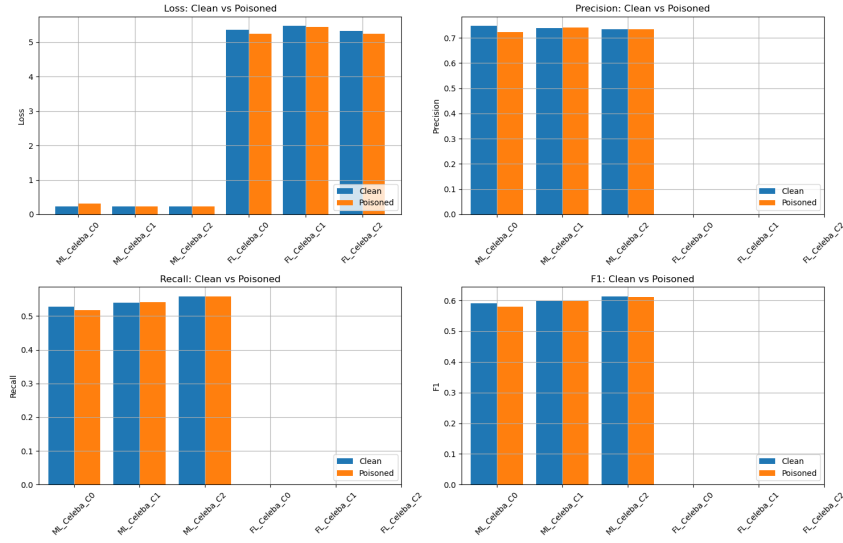


Figure 6.7: Performance metrics for CelebA clients (Targeted Label Flip).

Targeted Label Flipping on CelebA (Smiling)

| Client       | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|--------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| ML_Celeba_C0 | 0.232        | 0.309           | 0.747             | 0.723                | 0.528          | 0.518             | 0.591      | 0.579         |
| ML_Celeba_C1 | 0.231        | 0.232           | 0.738             | 0.742                | 0.539          | 0.542             | 0.599      | 0.601         |
| ML_Celeba_C2 | 0.229        | 0.23            | 0.735             | 0.735                | 0.559          | 0.558             | 0.614      | 0.611         |
| FL_Celeba_C0 | 5.364        | 5.237           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C1 | 5.468        | 5.438           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C2 | 5.322        | 5.241           | -                 | -                    | -              | -                 | -          | -             |

Figure 6.8: Clean vs poisoned test results for each CelebA client (Targeted Label Flip).

**CIFAR-10 Experiment.** In the clean setting, Task 1 clients exhibit loss values ranging from 0.651 to 0.851 and F1-scores between 0.722 and 0.77. Once the targeted label-flipping attack is applied to AN\_Cifar10\_C0, its performance declines noticeably: the loss increases from 0.651 to 2.141 (a 229% increase), precision drops from 0.777 to 0.618 (down 20.4%), recall from 0.77 to 0.678 (down 11.9%), and F1-score from 0.77 to 0.632 (a 17.9%

reduction). Compared to the random flip attack, the damage is more focused and less severe, consistent with the fact that only one class is flipped.

The other two Task 1 clients (`AN_Cifar10_C1` and `AN_Cifar10_C2`) are slightly affected. `AN_Cifar10_C1`'s loss decreases slightly from 0.851 to 0.782, with a minor F1-score increase from 0.722 to 0.734. Similarly, `AN_Cifar10_C2` sees a slight drop in loss and a marginal F1 gain. Task 2 clients (`OB_Cifar10_C0-C2`) remain nearly unchanged, with losses shifting by less than 0.05 and F1-scores stable above 0.88. These results reinforce that the attack's impact remains localized to the targeted client.

**CelebA Experiment.** A similar pattern emerges in CelebA. In the clean setting, Task 1 clients have losses between 0.229 and 0.232, and F1-scores around 0.591–0.614. Task 2 clients report higher losses (5.322–5.468).

After the attack, the poisoned client `ML_Celeba_C0` shows moderate degradation. Its loss rises from 0.232 to 0.309, while precision drops from 0.747 to 0.723 and F1-score from 0.591 to 0.579. However, the change is less severe than under the random flip attack. Meanwhile, the two clean Task 1 clients remain stable or slightly improve: `ML_Celeba_C1`'s F1 increases from 0.599 to 0.601, and `ML_Celeba_C2`'s F1 remains nearly constant. Task 2 clients are unaffected, with losses changing by less than 5%.

### 6.2.3 Trigger Injection

This section studies a back-door-style “mosaic” trigger that is pasted onto every training image of a single Task-1 client. Two patch ratios are considered: a small  $0.3^2$  patch that covers 9 % of the image area and a large  $0.8^2$  patch that covers 64 %. The trigger is applied to `AN_Cifar10_C0` in CIFAR-10 and to `ML_Celeba_C0` in CelebA; all other clients remain clean. CelebA results appear first in Figure 6.9–6.12; the corresponding CIFAR-10 results follow in Figure 6.13–6.16.

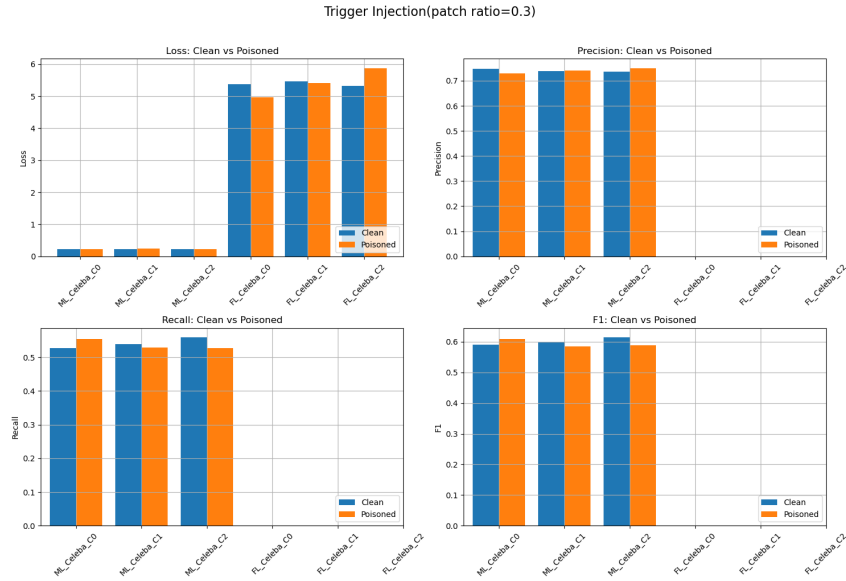


Figure 6.9: CelebA – performance metrics, trigger patch ratio 0.3.

Trigger Injection(patch ratio=0.3)

| Client       | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|--------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| ML_Celeba_C0 | 0.232        | 0.231           | 0.747             | 0.73                 | 0.528          | 0.554             | 0.591      | 0.61          |
| ML_Celeba_C1 | 0.231        | 0.237           | 0.738             | 0.74                 | 0.539          | 0.53              | 0.599      | 0.585         |
| ML_Celeba_C2 | 0.229        | 0.233           | 0.735             | 0.75                 | 0.559          | 0.527             | 0.614      | 0.589         |
| FL_Celeba_C0 | 5.364        | 4.963           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C1 | 5.468        | 5.417           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C2 | 5.322        | 5.872           | -                 | -                    | -              | -                 | -          | -             |

Figure 6.10: CelebA – clean vs poisoned results, trigger patch ratio 0.3.

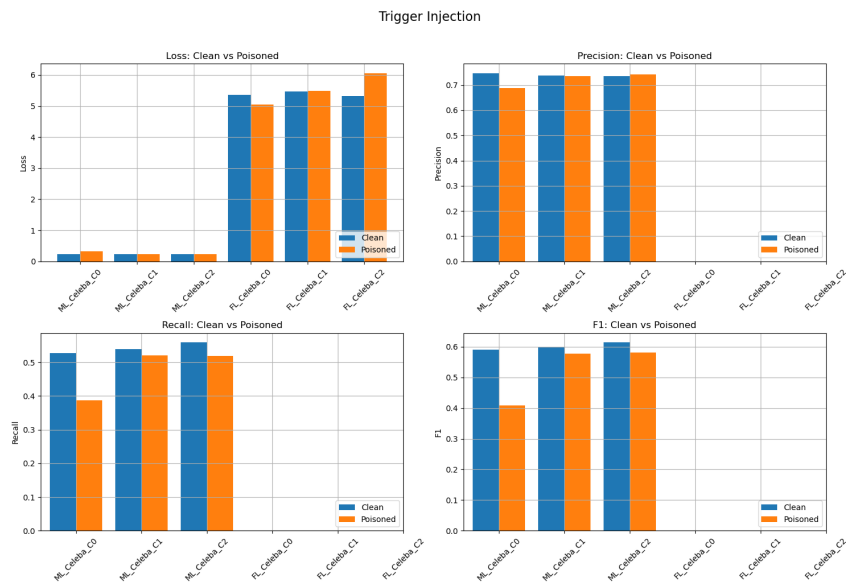


Figure 6.11: CelebA – performance metrics, trigger patch ratio 0.8.

Trigger Injection

| Client       | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|--------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| ML_Celeba_C0 | 0.232        | 0.317           | 0.747             | 0.688                | 0.528          | 0.387             | 0.591      | 0.409         |
| ML_Celeba_C1 | 0.231        | 0.238           | 0.738             | 0.735                | 0.539          | 0.521             | 0.599      | 0.577         |
| ML_Celeba_C2 | 0.229        | 0.237           | 0.735             | 0.743                | 0.559          | 0.519             | 0.614      | 0.581         |
| FL_Celeba_C0 | 5.364        | 5.045           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C1 | 5.468        | 5.493           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C2 | 5.322        | 6.057           | -                 | -                    | -              | -                 | -          | -             |

Figure 6.12: CelebA – clean vs poisoned results, trigger patch ratio 0.8.

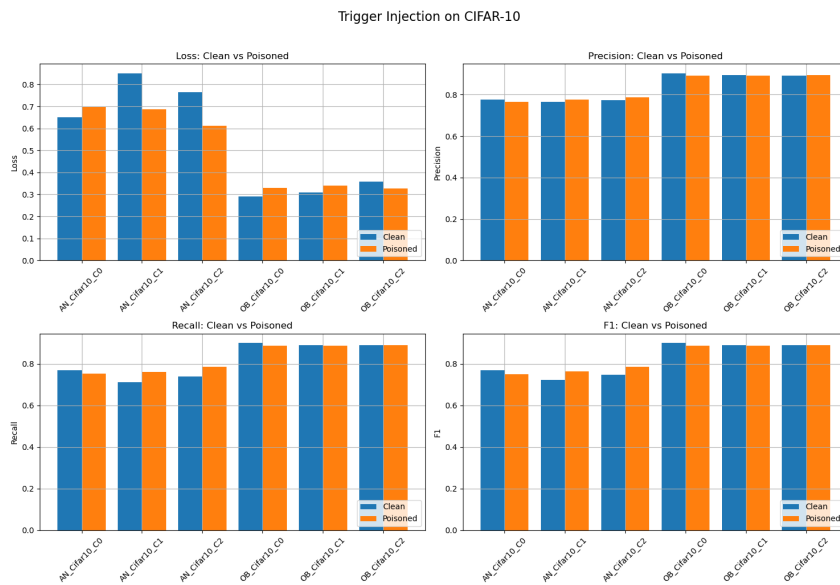


Figure 6.13: CIFAR-10 – performance metrics, trigger patch ratio 0.3.

Trigger Injection on CIFAR-10

| Client        | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|---------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| AN_Cifar10_C0 | 0.651        | 0.701           | 0.777             | 0.764                | 0.77           | 0.752             | 0.77       | 0.75          |
| AN_Cifar10_C1 | 0.851        | 0.687           | 0.766             | 0.775                | 0.713          | 0.761             | 0.722      | 0.765         |
| AN_Cifar10_C2 | 0.765        | 0.612           | 0.772             | 0.786                | 0.74           | 0.786             | 0.746      | 0.786         |
| DB_Cifar10_C0 | 0.29         | 0.331           | 0.903             | 0.892                | 0.9            | 0.886             | 0.9        | 0.886         |
| DB_Cifar10_C1 | 0.309        | 0.339           | 0.894             | 0.891                | 0.891          | 0.887             | 0.891      | 0.887         |
| DB_Cifar10_C2 | 0.357        | 0.328           | 0.892             | 0.894                | 0.89           | 0.889             | 0.889      | 0.889         |

Figure 6.14: CIFAR-10 – clean vs poisoned results, trigger patch ratio 0.3.

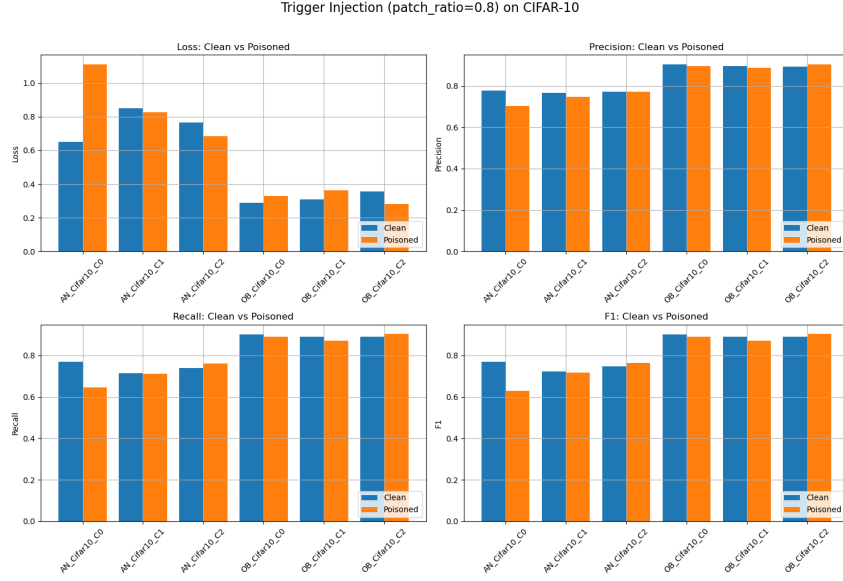


Figure 6.15: CIFAR-10 – performance metrics, trigger patch ratio 0.8.

Trigger Injection(patch\_ratio=0.8)on CIFAR-10

| Client        | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|---------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| AN_Cifar10_C0 | 0.651        | 1.111           | 0.777             | 0.703                | 0.77           | 0.646             | 0.77       | 0.629         |
| AN_Cifar10_C1 | 0.851        | 0.825           | 0.766             | 0.747                | 0.713          | 0.711             | 0.722      | 0.716         |
| AN_Cifar10_C2 | 0.765        | 0.684           | 0.772             | 0.771                | 0.74           | 0.762             | 0.746      | 0.763         |
| OB_Cifar10_C0 | 0.29         | 0.328           | 0.903             | 0.896                | 0.9            | 0.89              | 0.9        | 0.889         |
| OB_Cifar10_C1 | 0.309        | 0.363           | 0.894             | 0.886                | 0.891          | 0.871             | 0.891      | 0.872         |
| OB_Cifar10_C2 | 0.357        | 0.282           | 0.892             | 0.904                | 0.89           | 0.903             | 0.889      | 0.903         |

Figure 6.16: CIFAR-10 – clean vs poisoned results, trigger patch ratio 0.8.

**CelebA Experiment.** In the clean setting, the same result is used as in the previous two attack experiments.

*Small patch (9%).* Surprisingly, the 0.3<sup>2</sup> trigger has almost no adverse effect. The poisoned client’s loss changes from 0.232 to 0.231 (-0.4 %), precision drops 2.3 %, recall rises 4.9 %, and F1 increases 3.2 %. Non-poisoned Task-1 clients shift by less than 2 % on any metric; Task-2 losses change by at most  $\pm 10$  % with no trend in either direction. The small mosaic thus behaves like benign data augmentation.

*Large patch (64%).* When the patch dominates the image the poisoned client degrades: loss +36.6 %, precision -7.9 %, recall -26.7 %, F1 -30.8 %. Non-poisoned Task-1 clients move by <3 % and Task-2 losses shift by  $|5-14|$  %. The attack remains tightly localised, mirroring the pattern seen in the previous label-flip experiments.

**CIFAR-10 Experiment.** With clean data, Task-1 losses range 0.651–0.851 (F1 0.72–0.77) and Task-2 losses stay below 0.36 (F1 > 0.88).

*Small patch (9 %).* The poisoned client’s loss rises modestly from 0.651 to 0.701 (+7.7 %); F1 slips by only 2.0 %. Peers in Task 1 improve slightly (loss  $-3$ – $11$  %, F1  $+0.6$ – $3.9$  %). Task-2 clients see negligible shifts ( $<3$  % on any metric). Again, a small trigger appears more like a stochastic augmentation than a back-door.

*Large patch (64 %).* The larger trigger produces a clearer impact: `AN_Cifar10_C0` loss  $+70.7$  %, precision  $-9.5$  %, recall  $-16.1$  %, F1  $-18.3$  %. The two clean Task-1 clients improve slightly overall (average loss  $-7.0$  %, F1  $+2.0$  %). Task-2 clients move by  $\leq 2\%$  on classification metrics and by an average  $|17\%|$  in loss, confirming that FedPer-HCA confines the damage largely to the attacker.

**Summary of Trigger Injection Attack.** Across datasets, a small 9 % mosaic does not harm the accuracy of the model, and can even marginally benefit it, acting as a benign perturbation. A large 64 % trigger meaningfully degrades only the poisoned client ( $-18$ – $31$  % in F1), while non-poisoned peers change by  $\leq 3\%$  on key metrics. Across the three data poisoning scenarios examined, random label flipping, targeted label flipping, and visual trigger injection, the results consistently indicate that the FedPer-HCA framework exhibits strong resilience. In each case, performance degradation is primarily limited to the directly poisoned client, with only negligible spillover effects on others. These results highlight the framework’s resilience against data poisoning, demonstrating its effectiveness in countering both label-based and input-based attacks in DFMTL.

## 6.3 Model poisoning scenarios

The three model poisoning scenarios include sign flipping, scaled boost and AT2FL.

### 6.3.1 Sign Flipping

This section investigates a *model-poisoning* attack in which the signs of the local gradient update are inverted on a single Task 1 client during every communication round. All remaining clients remain clean. The CIFAR-10 outcome is shown in Figure 6.17 and Figure 6.18; the corresponding CelebA results are given in Figure 6.19 and Figure 6.20.

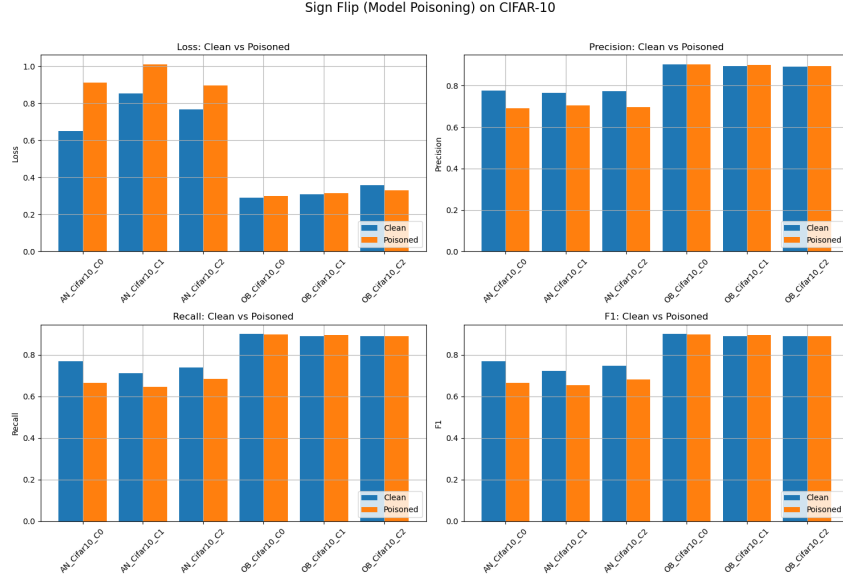


Figure 6.17: Performance metrics for CIFAR-10 clients (Sign Flip).

Sign Flip (Model Poisoning) on CIFAR-10

| Client        | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|---------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| AN_Cifar10_C0 | 0.651        | 0.912           | 0.777             | 0.69                 | 0.77           | 0.665             | 0.77       | 0.666         |
| AN_Cifar10_C1 | 0.851        | 1.01            | 0.766             | 0.704                | 0.713          | 0.646             | 0.722      | 0.655         |
| AN_Cifar10_C2 | 0.765        | 0.897           | 0.772             | 0.697                | 0.74           | 0.683             | 0.746      | 0.682         |
| OB_Cifar10_C0 | 0.29         | 0.3             | 0.903             | 0.901                | 0.9            | 0.899             | 0.9        | 0.898         |
| OB_Cifar10_C1 | 0.309        | 0.314           | 0.894             | 0.899                | 0.891          | 0.896             | 0.891      | 0.896         |
| OB_Cifar10_C2 | 0.357        | 0.329           | 0.892             | 0.895                | 0.89           | 0.889             | 0.889      | 0.89          |

Figure 6.18: Clean vs poisoned test results for CIFAR-10 (Sign Flip).

**CIFAR-10 Experiment.** *Clean baseline.* Losses span 0.651–0.851 for Task 1 (F1 0.722–0.770) and stay below 0.36 for Task 2 (F1 > 0.88).

**Analysis of the poisoned Task-1 client.** AN\_Cifar10\_C0 loss rises from 0.651 to 0.912, a 40.2 % increase. Precision falls 11.2 %, recall 13.6 %, and F1 13.5 %.

**Analysis of non-poisoned Task-1 clients.** Both honest peers deteriorate noticeably:

- AN\_Cifar10\_C1: loss +18.7%, precision –8.1%, recall –9.4%, F1 –9.3%.
- AN\_Cifar10\_C2: loss +17.3%, precision –9.7%, recall –7.7%, F1 –8.6%.

On average, the absolute loss change across the two clean Task 1 clients is 18.0%; average absolute changes in precision, recall, and F1 are 8.9 %, 8.5 %, and 9.0 %, respectively—an order of magnitude larger than under data-poisoning attacks.

**Analysis of Task-2 clients.** Loss shifts are modest: +3.4%, +1.6%, and  $-7.8\%$  for OB\_C0-C2 (mean absolute 4.3 %). The attack induces only minor effects on cross-task clients.

**Visual confirmation in Figure 6.17.** Orange bars drop well below blue bars in the *all three* Task-1 clients, while Task-2 bars almost overlap, confirming intra-task spread but slight cross-task impact.

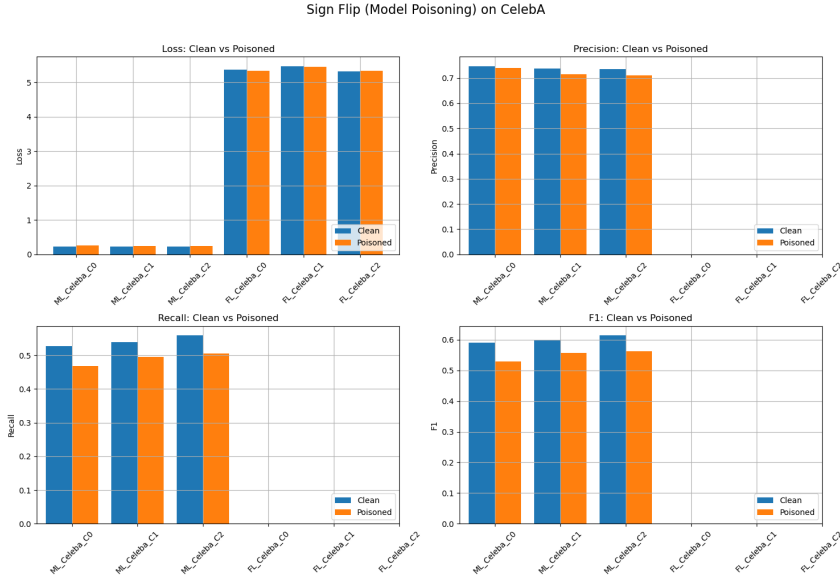


Figure 6.19: Performance metrics for CelebA clients (Sign Flip).

Sign Flip (Model Poisoning) on CelebA

| Client       | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|--------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| ML_CelebA_C0 | 0.232        | 0.252           | 0.747             | 0.741                | 0.528          | 0.468             | 0.591      | 0.529         |
| ML_CelebA_C1 | 0.231        | 0.251           | 0.738             | 0.714                | 0.539          | 0.495             | 0.599      | 0.557         |
| ML_CelebA_C2 | 0.229        | 0.249           | 0.735             | 0.71                 | 0.559          | 0.505             | 0.614      | 0.563         |
| FL_CelebA_C0 | 5.364        | 5.338           | -                 | -                    | -              | -                 | -          | -             |
| FL_CelebA_C1 | 5.468        | 5.446           | -                 | -                    | -              | -                 | -          | -             |
| FL_CelebA_C2 | 5.322        | 5.33            | -                 | -                    | -              | -                 | -          | -             |

Figure 6.20: Clean vs poisoned test results for CelebA (Sign Flip).

**CelebA Experiment.** The clean losses of Task-1 are between 0.229 and 0.232 (F1 0.591–0.614); the losses of Task-2 are 5.32–5.47.

**Task-1 impact.** Sign flipping raises the poisoned client’s loss by 8.6 % and lowers F1 by 10.5 %. Crucially, *both* honest Task-1 clients also degrade: average loss +8.7%, precision  $-3.3\%$ , recall  $-9.0\%$ , F1  $-7.7\%$ . Thus, unlike data poisoning attacks, the model-level perturbation propagates within the task group.

**Task-2 impact.** Landmark-detection clients change by less than 0.5 % in loss, again showing strong isolation across heterogeneous tasks.

**Visual confirmation.** Figure 6.19 shows uniform downward shifts of the orange bars for all three Task-1 clients, whereas the Task-2 bars remain nearly unchanged.

**Summary of Sign-Flipping Attack.** Unlike the three data poisoning attacks considered earlier, model poisoning via sign flipping degrades all clients that are in the same task as the poisoned client, while clients associated with other tasks remain unaffected.

### 6.3.2 Scaled Boost

This attack multiplies the poisoned client’s local gradient by a *fixed factor* before every upload. A boost factor of  $5\times$  is used, which is a relatively large value that still allows the 15 communication rounds to complete without training failure. Higher factors cause training failure: at  $8\times$  the federation collapses in round 9, and at  $10\times$  it fails in round 6 with a NaN in backbone! error. The CelebA results appear first in Figure 6.21 and Figure 6.22; the corresponding CIFAR-10 outcome follows in Figure 6.23 and Figure 6.24.

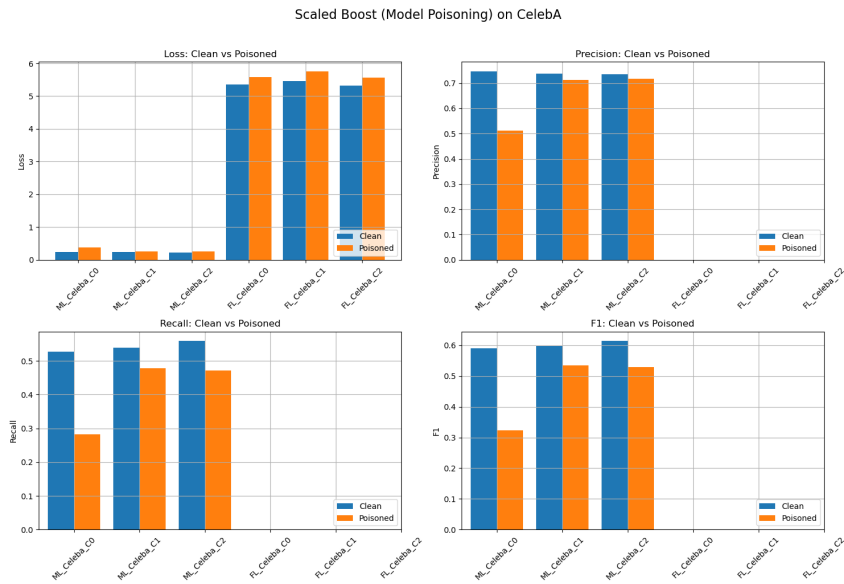
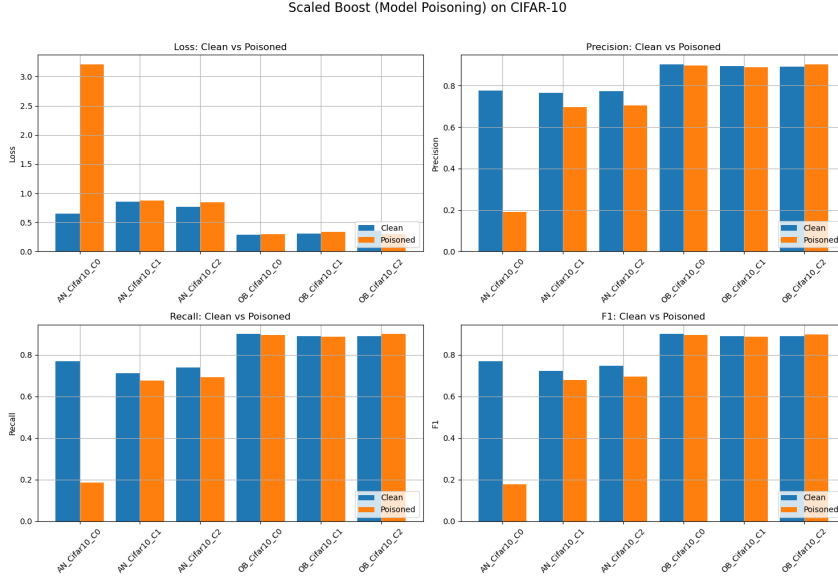


Figure 6.21: CelebA – performance metrics (Scaled Boost,  $5\times$ ).

Scaled Boost (Model Poisoning) on CelebA

| Client       | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|--------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| ML_Celeba_C0 | 0.232        | 0.374           | 0.747             | 0.513                | 0.528          | 0.283             | 0.591      | 0.323         |
| ML_Celeba_C1 | 0.231        | 0.253           | 0.738             | 0.713                | 0.539          | 0.478             | 0.599      | 0.535         |
| ML_Celeba_C2 | 0.229        | 0.253           | 0.735             | 0.718                | 0.559          | 0.472             | 0.614      | 0.53          |
| FL_Celeba_C0 | 5.364        | 5.588           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C1 | 5.468        | 5.759           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C2 | 5.322        | 5.557           | -                 | -                    | -              | -                 | -          | -             |

Figure 6.22: CelebA – clean vs poisoned results (Scaled Boost,  $5\times$ ).

Figure 6.23: CIFAR-10 – performance metrics (Scaled Boost,  $5\times$ ).

Scaled Boost (Model Poisoning) on CIFAR-10

| Client        | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|---------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| AN_Cifar10_C0 | 0.651        | 3.212           | 0.777             | 0.191                | 0.77           | 0.187             | 0.77       | 0.177         |
| AN_Cifar10_C1 | 0.851        | 0.877           | 0.766             | 0.697                | 0.713          | 0.677             | 0.722      | 0.679         |
| AN_Cifar10_C2 | 0.765        | 0.843           | 0.772             | 0.704                | 0.74           | 0.693             | 0.746      | 0.694         |
| DB_Cifar10_C0 | 0.29         | 0.299           | 0.903             | 0.898                | 0.9            | 0.896             | 0.9        | 0.896         |
| DB_Cifar10_C1 | 0.309        | 0.336           | 0.894             | 0.889                | 0.891          | 0.886             | 0.891      | 0.886         |
| DB_Cifar10_C2 | 0.357        | 0.301           | 0.892             | 0.901                | 0.89           | 0.9               | 0.889      | 0.899         |

Figure 6.24: CIFAR-10 – clean vs poisoned results (Scaled Boost,  $5\times$ ).

**CIFAR-10 Experiment.** *Clean baseline.* Task-1 losses range 0.651–0.851 (F1 0.722–0.770); Task-2 losses remain below 0.36 (F1  $> 0.88$ ).

**Analysis of the poisoned Task-1 client.** AN\_Cifar10\_C0 deteriorates sharply: loss +393% (0.651  $\rightarrow$  3.212), precision  $-75.4\%$ , recall  $-75.7\%$ , and F1  $-77.0\%$ .

**Analysis of non-poisoned Task-1 clients.** AN\_Cifar10\_C1: loss +3.1%, precision  $-9.0\%$ , recall  $-5.0\%$ , F1  $-6.0\%$ . AN\_Cifar10\_C2: loss +10.2%, precision  $-8.8\%$ , recall  $-6.4\%$ , F1  $-7.0\%$ . *Average absolute change* among these honest peers: loss 6.6%, precision 8.9%, recall 5.7%, F1 6.5%. Hence, the boost not only cripples the attacker but *also spreads* noticeable harm within the task.

**Analysis of Task-2 clients.** Loss shifts are +3.1%, +8.7%, and  $-15.7\%$  (mean absolute 9.2 %). Precision, recall, and F1 vary by  $\leq 1.1\%$  on average, confirming strong cross-task isolation.

**CelebA Experiment.** *Clean baseline.* Task-1 losses 0.229–0.232 (F1 0.591–0.614); Task-2 losses 5.32–5.47.

**Task-1 impact.** The poisoned client’s loss jumps +61.2% ( $0.232 \rightarrow 0.374$ ); precision  $-31.3\%$ , recall  $-46.4\%$ , F1  $-45.4\%$ . Honest peers also decay: mean loss  $+10.0\%$ ; precision  $-2.9\%$ ; recall  $-13.4\%$ ; F1  $-12.2\%$ .

**Task-2 impact.** Landmark-detection losses rise modestly by 4–6 %, with no classification metrics affected.

**Visual Confirmation.** Figures 6.21 and 6.23 show that orange bars fall well below blue bars for *all* Task-1 clients, whereas Task-2 bars remain nearly unchanged—mirroring the numeric analysis.

**Summary of Scaled-Boost Attack.** Boosting the poisoned client’s gradient by a factor of five inflicts: F1 drops 77 % on the attacked client and 6–12 % on unattacked peers on the same task, while cross-task clients experience only marginal changes. Larger factors trigger numerical instability, indicating that FedPer-HCA resists extreme boosts by halting training rather than propagating corrupt updates. Overall, gradient-amplification attacks are more damaging than sign flipping, but their impact remains confined to the attacker’s task group, reinforcing earlier conclusions about the framework’s cross-task robustness.

### 6.3.3 AT2FL

AT2FL (**ATT**ack on **F**ederated **L**earning) is the strongest *model-poisoning* method among the three model attack methods, which is a gradient/weight perturbation attacking specifically targeting the global aggregation process in federated learning. It reshapes **the entire local update** of the adversarial client so as to *maximise* the global loss while staying within a small  $\ell_\infty$ -budget, thereby evading simple clipping defences. In practice, the attack is numerically fragile: using the default coefficients proposed by the authors [13] causes the FedPer-HCA framework to diverge during training, resulting in a **NaN in backbone!** error. After extensive clipping and sanitation, we found that  $\epsilon = 0.015$  and only poisoning the first 8 mini-batches let the training complete all 15 rounds without instability. Even at this conservative setting AT2FL degrades performance more than Sign Flip and approaches the impact of Scaled Boost.

The CIFAR-10 results appear in Figure 6.25 and Table 6.26; the corresponding CelebA evaluation is given in Figure 6.27 and Table 6.28.

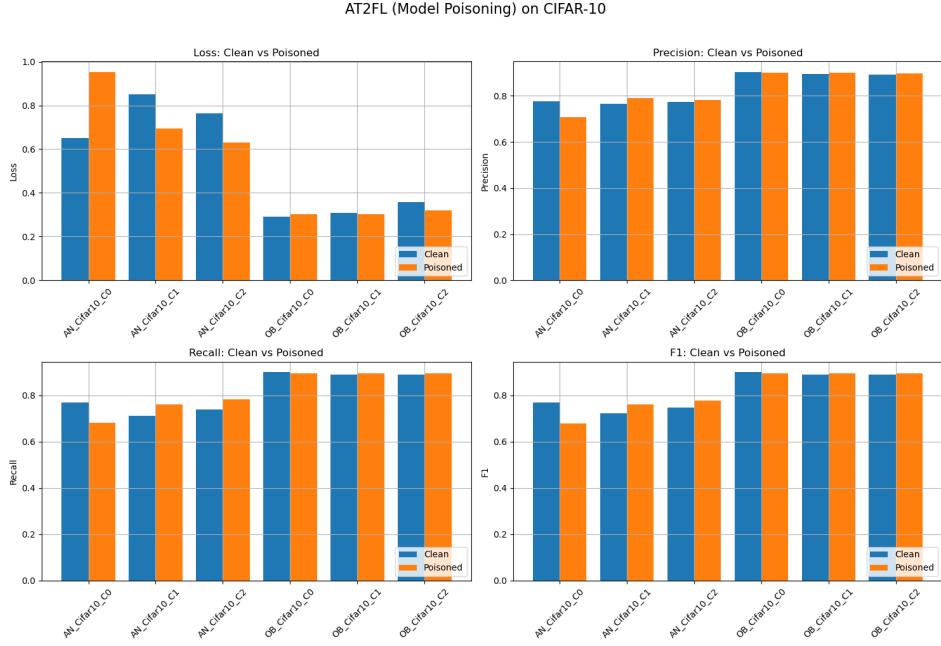


Figure 6.25: Performance metrics for CIFAR-10 clients (AT2FL).

AT2FL (Model Poisoning) on CIFAR-10

| Client        | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|---------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| AN_Cifar10_C0 | 0.651        | 0.953           | 0.777             | 0.706                | 0.77           | 0.681             | 0.77       | 0.68          |
| AN_Cifar10_C1 | 0.851        | 0.693           | 0.766             | 0.791                | 0.713          | 0.762             | 0.722      | 0.762         |
| AN_Cifar10_C2 | 0.765        | 0.63            | 0.772             | 0.782                | 0.74           | 0.782             | 0.746      | 0.778         |
| OB_Cifar10_C0 | 0.29         | 0.302           | 0.903             | 0.9                  | 0.9            | 0.896             | 0.9        | 0.896         |
| OB_Cifar10_C1 | 0.309        | 0.302           | 0.894             | 0.9                  | 0.891          | 0.896             | 0.891      | 0.896         |
| OB_Cifar10_C2 | 0.357        | 0.32            | 0.892             | 0.896                | 0.89           | 0.895             | 0.889      | 0.894         |

Figure 6.26: Clean vs. poisoned test results for each CIFAR-10 client (AT2FL).

**CIFAR-10 Experiment.** Prior to poisoning, Task 1 (Animals) losses range from 0.651 – 0.851 with  $F1 = 0.722 - 0.770$ , while Task 2 (Objects) keeps loss  $< 0.36$  and  $F1 > 0.88$ .

**Poisoned Task 1 client.** Client AN\_Cifar10\_C0 shows the steepest decline: loss soars by 46.4 % ( $0.651 \rightarrow 0.953$ ), precision drops 9.1 %, recall 11.6 %, and F1 11.7 %.

**Non-poisoned Task 1 clients.** AN\_C1 and AN\_C2 *improve* slightly— loss decreases by an average 18.1 %, while precision (+2.3 %), recall (+6.3 %) and F1 (+4.9 %) rise. The damage therefore does *not* propagate aggressively within the task group.

**Task 2 clients.** Absolute loss shifts are modest (mean 5.6 %), and all other metrics vary by  $\leq 0.6$  %, confirming that cross-task FedHCA aggregation absorbs most of the poisoned update.

**Visual confirmation.** Figure 6.25 highlights these observations: the orange bar for AN\_C0 deviates most significantly from its blue counterpart, reflecting direct poisoning. Its task-1 peers, AN\_C1 and AN\_C2, also show noticeable changes, while all task-2 clients remain tightly clustered.

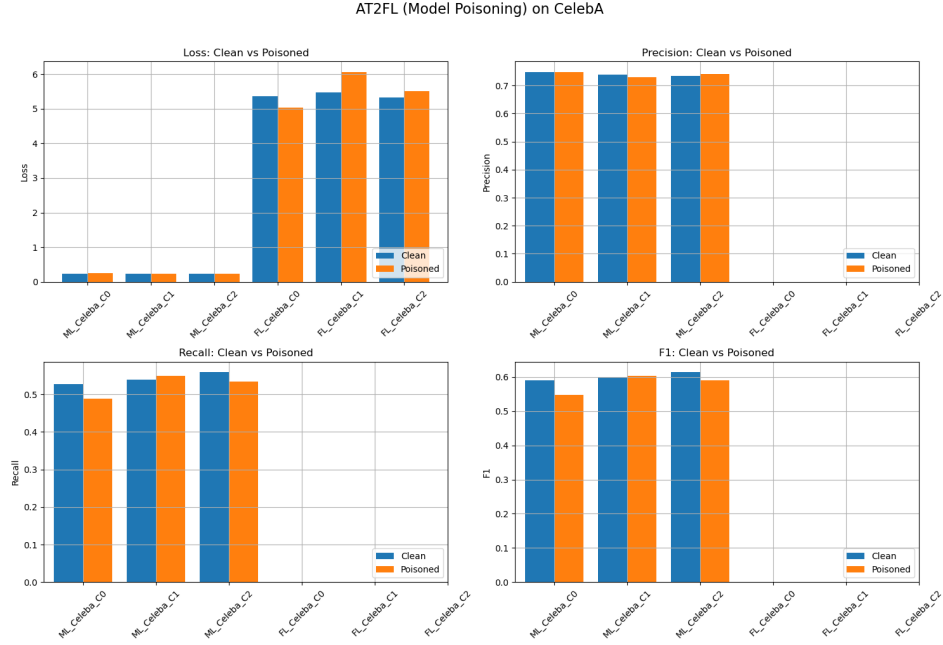


Figure 6.27: Performance metrics for CelebA clients (AT2FL).

AT2FL (Model Poisoning) on CelebA

| Client       | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|--------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| ML_Celeba_C0 | 0.232        | 0.243           | 0.747             | 0.748                | 0.528          | 0.488             | 0.591      | 0.547         |
| ML_Celeba_C1 | 0.231        | 0.231           | 0.738             | 0.729                | 0.539          | 0.549             | 0.599      | 0.604         |
| ML_Celeba_C2 | 0.229        | 0.233           | 0.735             | 0.741                | 0.559          | 0.535             | 0.614      | 0.591         |
| FL_Celeba_C0 | 5.364        | 5.026           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C1 | 5.468        | 6.06            | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C2 | 5.322        | 5.511           | -                 | -                    | -              | -                 | -          | -             |

Figure 6.28: Clean vs. poisoned test results for each CelebA client (AT2FL).

**CelebA Experiment.** Baseline performance mirrors the earlier sections: Task 1 (multi-label attributes) loss  $\approx 0.23$  with F1  $\approx 0.60$ ; Task 2 (facial landmarks) loss 5.32 – 5.47.

**Poisoned Task 1 client.** ML\_Celeba\_C0 sees a moderate loss rise of 4.7 % and an F1 decrease of 7.4 %.

**Non-poisoned Task 1 peers.** Average absolute changes are small— loss 0.9 %, precision 1.0 %, recall 3.1 %, F1 2.3 %.

**Task 2 clients.** Loss shifts remain within  $\pm 11\%$ , once again indicating that cross-task aggregation prevents wider contamination.

**Summary of AT2FL.** With carefully clipped parameters, AT2FL harms the attacked client itself (up to  $+46\%$  loss) and, unlike Sign Flip, appears to improve the performance of its task-group peers (AN\_C1 and AN\_C2), while leaving the other task almost untouched. However, these results reflect a heavily optimized training configuration: numerical sanitization, parameter cutting, and reduced attack strength were applied to ensure convergence over 15 communication rounds. In practice, AT2FL is the most disruptive of the three model poisoning methods: applying the original coefficients of [13] quickly destabilizes the FedPer-HCA framework (**NaN in backbone!**), halting training within a few rounds. The apparent performance gains in peer clients are artifacts of defensive mitigation and highlight how fragile the framework becomes under aggressive gradient-level perturbation, even when partially contained.

Overall, combining all model poisoning studies (Sign Flip, Scaled Boost, AT2FL) and the three data poisoning studies (Random and Targeted Label Flips and trigger injection), a consistent pattern emerges: FedPer-HCA confines the bulk of the damage to the compromised client itself; while model poisoning can disrupt in-task peers to some extent, all six attack strategies fail to impact clients in other tasks. These findings, observed across heterogeneous datasets and attack styles, emphasize the robustness of the proposed federation in isolating cross-task contamination.

## 6.4 Malicious Aggregation

This attack tampers with the *cross-task aggregation step* itself. Instead of altering a single client, the adversary scales the conflict-averse direction returned by HCA by a factor **2** before broadcasting it:

$$\tilde{\Delta}_{\text{agg}} = 2 \Delta_{\text{HCA}}.$$

Larger factors ( $\geq 5$ ) or other perturbations (gradient sign flip, Gaussian noise) caused **NaN in backbone** errors, indicating that ColNet’s optimisation quickly becomes unstable once the aggregator is heavily corrupted. Figure 6.29 and Figure 6.30 plot the CIFAR-10 outcome; Figure 6.31 and Figure 6.32 report CelebA.

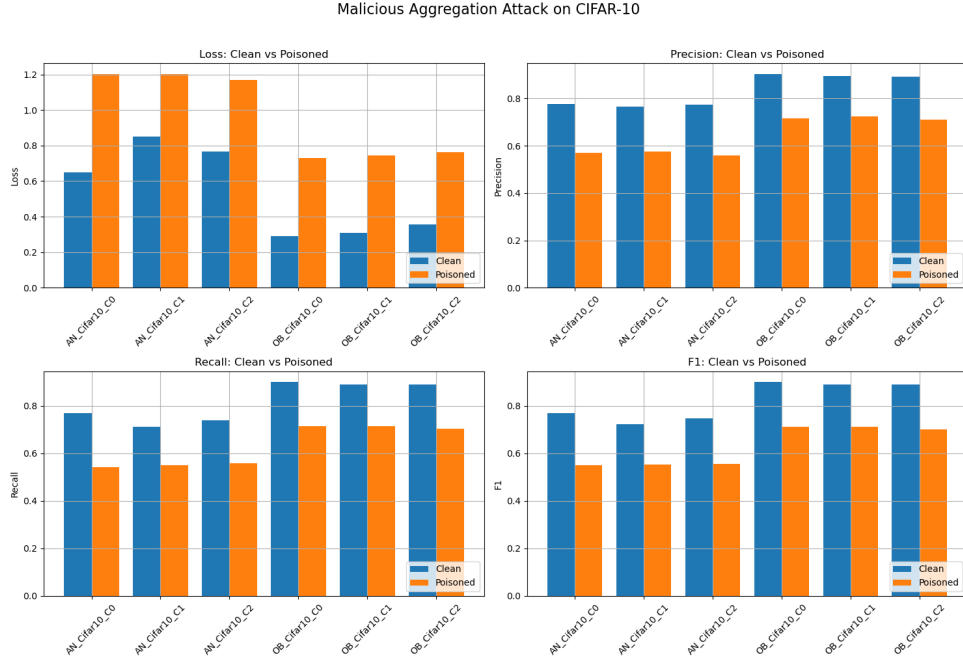


Figure 6.29: Performance metrics for CIFAR-10 clients (Malicious Aggregation).

Malicious Aggregation Attack on CIFAR-10

| Client        | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|---------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| AN_Cifar10_C0 | 0.651        | 1.203           | 0.777             | 0.57                 | 0.77           | 0.543             | 0.77       | 0.55          |
| AN_Cifar10_C1 | 0.851        | 1.202           | 0.766             | 0.574                | 0.713          | 0.55              | 0.722      | 0.552         |
| AN_Cifar10_C2 | 0.765        | 1.168           | 0.772             | 0.56                 | 0.74           | 0.559             | 0.746      | 0.556         |
| OB_Cifar10_C0 | 0.29         | 0.73            | 0.903             | 0.716                | 0.9            | 0.714             | 0.9        | 0.713         |
| OB_Cifar10_C1 | 0.309        | 0.744           | 0.894             | 0.723                | 0.891          | 0.715             | 0.891      | 0.713         |
| OB_Cifar10_C2 | 0.357        | 0.764           | 0.892             | 0.71                 | 0.89           | 0.704             | 0.889      | 0.702         |

Figure 6.30: Clean vs. poisoned results for each CIFAR-10 client (Malicious Aggregation).

**CIFAR-10 Experiment.** All clients performed well in the clean setting (Task 1 losses 0.651–0.851, F1 0.722–0.77; Task 2 losses < 0.36, F1 > 0.89). After the aggregation is tripled:

- **Task 1 (Animals).** Loss rises by **60%** on average (up to +85% at AN\_C0); precision, recall and F1 each fall by  $\approx 26\%$ .
- **Task 2 (Objects).** Loss surges even more—**+135%** on average—yet the classification metrics drop by about **20%**. Thus both task groups suffer sizeable, *fairly uniform* degradation.

**Visual confirmation.** Figure 6.29 shows every orange bar well below its blue counterpart—no client is spared, confirming the system-wide spread.

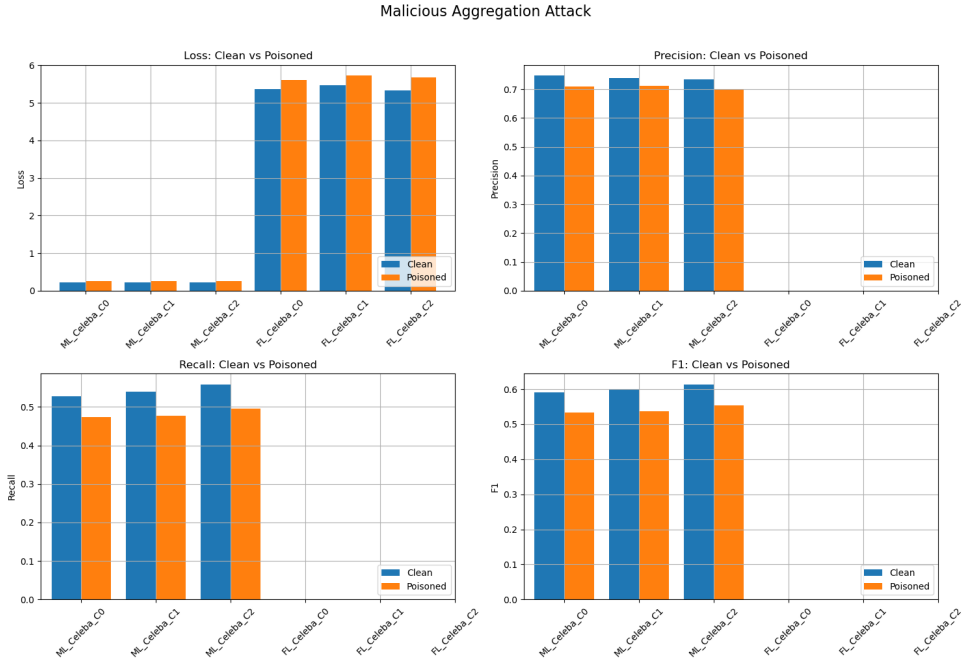


Figure 6.31: Performance metrics for CelebA clients (Malicious Aggregation).

Malicious Aggregation Attack

| Client       | Loss (Clean) | Loss (Poisoned) | Precision (Clean) | Precision (Poisoned) | Recall (Clean) | Recall (Poisoned) | F1 (Clean) | F1 (Poisoned) |
|--------------|--------------|-----------------|-------------------|----------------------|----------------|-------------------|------------|---------------|
| ML_Celeba_C0 | 0.232        | 0.255           | 0.747             | 0.71                 | 0.528          | 0.474             | 0.591      | 0.534         |
| ML_Celeba_C1 | 0.231        | 0.254           | 0.738             | 0.711                | 0.539          | 0.477             | 0.599      | 0.537         |
| ML_Celeba_C2 | 0.229        | 0.254           | 0.735             | 0.698                | 0.559          | 0.496             | 0.614      | 0.553         |
| FL_Celeba_C0 | 5.364        | 5.611           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C1 | 5.468        | 5.721           | -                 | -                    | -              | -                 | -          | -             |
| FL_Celeba_C2 | 5.322        | 5.677           | -                 | -                    | -              | -                 | -          | -             |

Figure 6.32: Clean vs. poisoned results for each CelebA client (Malicious Aggregation).

**CelebA Experiment.** The same tripling produces a milder yet still consistent drop.

- **Task 1 (Multi-label).** Loss increases by  $\approx 10\%$ ; precision drops 5%, recall 11%, and F1 10% on average.
- **Task 2 (Landmark Regression).** Losses rise by only 5–7%—small but systematic.

Again, *all* clients are affected, although less severely than in CIFAR-10.

**Summary of Malicious Aggregation Attack.** Unlike previous attacks that remained largely localized, scaling the HCA update corrupts the *shared encoder* broadcast to every participant, propagating errors across tasks and datasets. Even with the smallest factor

that preserves training stability ( $\times 2$ ), CIFAR-10 experiences a 20–60% drop in all metrics, while CelebA shows a uniform 5–10% decline. These results highlight HCA’s critical role as a “single point of failure”: tampering with the aggregation routine degrades the entire federation, and stronger perturbations crash the framework outright.



# Chapter 7

## Summary, Conclusions and Future Work

Overall, this project fulfilled the goal of conducting robustness testing for DFMTL frameworks, carried out a theoretical analysis and practical implementation of poisoning attacks tailored to such frameworks, and obtained the corresponding empirical results. This chapter presents a summary of the work, discusses the main conclusions drawn from the experiments, and outlines directions for future research.

### 7.1 Summary

This thesis was set up to provide the first systematic poisoning-robustness study of a DFMTL framework. Building on the FedPer–HCA architecture:

1. designed and implemented seven representative poisoning strategies that jointly cover *data*, *model*, and *aggregation* attack surfaces;
2. integrated these attacks into the open-source DFMTL code base with *minimal* changes to the original training pipeline;
3. evaluated the impact of poisoning attack strategies on class label and task heterogeneity across 15 communication rounds, reporting four standard metrics for every client and task;

#### 7.1.1 Key Findings

**(1) Data-level poisoning is largely contained.** Random label flipping, targeted label flipping, and mosaic trigger injection degraded the *poisoned* client severely (up to 77 % F1 drop), yet the average absolute change on *unpoisoned* clients in the same task never exceeded 6 %. Cross-task clients were affected by at most 2 %, indicating that FedPer–HCA effectively localises input-space corruption.

**(2) Model-level poisoning propagates within a task but not across tasks.** Sign flip and Scaled Boost reduced the F1 of unpoisoned clients on the same task with poisoned clients by roughly 9 % and 12 %, respectively, while leaving the other task almost intact ( $\leq 1$  % on average). AT2FL, with conservative clipping ( $\epsilon = 0.015$ ), inflicted up to 46 % loss increase on the poisoned client.

**(3) Aggregation manipulation is the critical vulnerability.** Malicious Aggregation Filter, which is scaling the HCA update by a factor of 2, produced a system-wide performance drop of 20–60 % on CIFAR-10 and 5–10 % on CelebA. Higher factors ( $\geq 5$ ) caused the training to diverge numerically and terminate within ten rounds. Thus the cross-task reconciliation step constitutes a “single point of failure” in the current design.

**(4) Unexpected side observations.** A small 9 % mosaic patch behaved like benign augmentation, slightly *improving* metric performance. Reducing the HCA update by half occasionally produced NaNs earlier than moderate amplification, suggesting non-linear stability regions that merit separate study.

## 7.2 Conclusions

1. **Practical robustness.** The fedper-hca framework can prevent most data poisoning attacks from propagating, confirming the potential of DFMTL for heterogeneous and privacy-sensitive deployments.
2. **Residual risks.** Model poisoning attacks remain partially effective, damaging not only the targeted client but also its peers in the same task. Any compromise of the aggregation routine can destabilize the entire DFMTL system.
3. **Contribution of this work.** The presented attack suite and empirical protocol provide a reproducible baseline against which future DFMTL defenses can be compared.

## 7.3 Limitations and Future Work

Although this work provides the first systematic poisoning robustness study of DFMTL frameworks, several limitations remain and open the door to future research.

- The study considers a *white-box* threat model in which the adversary controls code execution on one client or the aggregator. In practice, adversaries may have more restricted access or rely on black-box query interfaces.
- Only two datasets (CelebA and CIFAR-10) and a single backbone architecture (ResNet-18 with “minus 1” configuration) were evaluated. The observed robustness may not generalize to larger models or different modalities such as audio, text, or multimodal tasks.

- The numerical stability constraints required conservative parameter tuning for AT2FL and malicious aggregation attacks. Stronger yet still stealthy variants could not be tested due to training divergence, but may exist.

Building on these observations, we propose several directions for future research:

1. **Black-box and limited-permission attacks.** Developing poisoning strategies that operate under restricted access assumptions, e.g., using model-inversion techniques, query-based triggers, or data-level perturbations that do not require full code control.
2. **Adaptive defences.** Enhancing DFMTL frameworks with task-aware anomaly detection, conflict-aware regularization, or Byzantine-robust optimisation tailored to the multi-task nature of the federation.
3. **Task-coupling analysis.** Theoretical study of when and why cross-task aggregation helps contain poisoning, possibly via gradient similarity measures, alignment scores, or task dependency graphs.
4. **Broader benchmarks.** Extending evaluation to other domains such as language modelling, speech recognition, or medical imaging, and testing deeper or more modern backbones like transformers or diffusion-based architectures.
5. **Stronger aggregation attacks.** Designing structured perturbations (e.g., low-rank interference, targeted direction injection) that subtly manipulate the global update without destabilizing training.

Overall, the results indicate that DFMTL with FedPer–HCA offers promising resilience against *data* poisoning, moderate robustness to *model* poisoning, and clear vulnerabilities in the *aggregation* layer. The public release of the framework, attacks, and evaluation scripts is intended to accelerate future work on both offensive and defensive techniques in decentralized multi-task federated learning.



# Bibliography

- [1] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, “Federated multi-task learning”, in *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [2] P. Kairouz, H. B. McMahan, *et al.*, “Advances and open problems in federated learning”, *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
- [3] Y. Lu, S. Huang, Y. Yang, S. Sirejiding, Y. Ding, and H. Lu, *Fedhca<sup>2</sup>: Towards hetero-client federated multi-task learning*, 2024. arXiv: 2311.13250 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2311.13250>.
- [4] E. T. Martínez Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys and Tutorials*, vol. 25, no. 4, pp. 2983–3013, 2023.
- [5] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated learning with non-iid data”, *arXiv preprint arXiv:1806.00582*, 2018.
- [6] R. Li, F. Ma, W. Jiang, and J. Gao, “Online federated multitask learning”, in *IEEE International Conference on Big Data*, 2019, pp. 215–220.
- [7] J. Yu, Y. Dai, X. Liu, *et al.*, *Unleashing the power of multi-task learning: A comprehensive survey spanning traditional, deep, and pretrained foundation model eras*, 2024. arXiv: 2404.18961 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2404.18961>.
- [8] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, “Federated learning with personalization layers”, *arXiv preprint arXiv:1912.00818*, 2019.
- [9] C. Feng, N. F. Kohler, A. H. Celdran, G. Bovet, and B. Stiller, *Colnet: Collaborative optimization in decentralized federated multi-task learning systems*, 2025. arXiv: 2501.10347 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2501.10347>.
- [10] G. Xia, J. Chen, C. Yu, and J. Ma, “Poisoning attacks in federated learning: A survey”, *IEEE Access*, vol. 11, pp. 10 708–10 722, 2023.
- [11] L. Yang, Y. Miao, Z. Liu, *et al.*, “Enhanced model poisoning attack and multi-strategy defense in federated learning”, *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 3877–3892, 2025.
- [12] E. T. Martínez Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2983–3013, 2023.

- [13] G. Sun, Y. Cong, J. Dong, Q. Wang, L. Lyu, and J. Liu, “Data poisoning attacks on federated machine learning”, *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 11 365–11 375, 2022.
- [14] C. Feng, Y. Li, Y. Gao, *et al.*, *Dmpa: Model poisoning attacks on decentralized federated learning for model differences*, 2025. arXiv: 2502.04771 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2502.04771>.
- [15] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data”, in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, vol. 54, PMLR, 20–22 Apr 2017, pp. 1273–1282.
- [16] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, “A review of applications in federated learning”, *Computers & Industrial Engineering*, vol. 149, p. 106 854, 2020, ISSN: 0360-8352.
- [17] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, “Braintorrent: A peer-to-peer environment for decentralized federated learning”, *arXiv preprint arXiv:1905.06731*, 2019.
- [18] S. Ruder, “An overview of multi-task learning in deep neural networks”, *arXiv preprint arXiv:1706.05098*, 2017.
- [19] Y. Zhang and Q. Yang, “A survey on multi-task learning”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5586–5609, 2021.
- [20] R. Aoki, F. Tung, and G. L. Oliveira, “Heterogeneous multi-task learning with expert diversity”, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 19, no. 6, pp. 3093–3102, 2022.
- [21] M. Crawshaw, “Multi-task learning with deep neural networks: A survey”, *arXiv preprint arXiv:2009.09796*, 2020.
- [22] L. Yuan, Z. Wang, L. Sun, P. S. Yu, and C. G. Brinton, “Decentralized federated learning: A survey and perspective”, *IEEE Internet of Things Journal*, vol. 11, no. 21, pp. 34 617–34 638, 2024.
- [23] C. Chen, T. Liao, X. Deng, Z. Wu, S. Huang, and Z. Zheng, “Advances in robust federated learning: Heterogeneity considerations”, *arXiv preprint arXiv:2405.09839*, 2024.
- [24] O. Sener and V. Koltun, *Multi-task learning as multi-objective optimization*, 2019. arXiv: 1810.04650 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1810.04650>.
- [25] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, *Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks*, 2018. arXiv: 1711.02257 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1711.02257>.
- [26] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, “Gradient surgery for multi-task learning”, *Advances in neural information processing systems*, vol. 33, pp. 5824–5836, 2020.
- [27] O. Marfoq, G. Neglia, A. Bellet, L. Kameni, and R. Vidal, “Federated multi-task learning under a mixture of distributions”, *Advances in neural information processing systems*, vol. 34, pp. 15 434–15 447, 2021.

- [28] S. Park, G. Kim, J. Kim, B. Kim, and J. C. Ye, “Federated split task-agnostic vision transformer for covid-19 cxr diagnosis”, *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 617–24 630, 2021.
- [29] Y.-Q. Chen, T. Zhang, X.-L. Jiang, Q. Chen, C.-L. Gao, and W.-L. Huang, “Fed-bone: Towards large-scale federated multi-task learning”, *Journal of Computer Science and Technology*, vol. 39, no. 5, pp. 1040–1057, 2024.
- [30] R. Cai, X. Chen, S. Liu, *et al.*, “Many-task federated learning: A new problem setting and a simple baseline”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 5037–5045.
- [31] W. Zhuang, Y. Wen, L. Lyu, and S. Zhang, “Mas: Towards resource-efficient federated multiple-task learning”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 23 414–23 424.
- [32] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, “Can machine learning be secure?”, in *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, 2006, pp. 16–25. DOI: 10.1145/1128817.1128824.
- [33] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning”, *Pattern Recognition*, vol. 84, pp. 317–331, 2018. DOI: 10.1016/j.patcog.2018.07.023.
- [34] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, “Badnets: Evaluating backdooring attacks on deep neural networks”, *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019. DOI: 10.1109/ACCESS.2019.2909068.
- [35] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning”, in *International conference on artificial intelligence and statistics*, PMLR, 2020, pp. 2938–2948.
- [36] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, “Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning”, in *2022 IEEE symposium on security and privacy (SP)*, IEEE, 2022, pp. 1354–1371.
- [37] X. Zhou, M. Xu, Y. Wu, and N. Zheng, “Deep model poisoning attack on federated learning”, *Future Internet*, vol. 13, no. 3, p. 73, 2021.
- [38] Y. Sun, H. Ochiai, and J. Sakuma, “Semi-targeted model poisoning attack on federated learning via backward error analysis”, in *2022 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2022, pp. 1–8.
- [39] X. Cao and N. Z. Gong, “Mpaf: Model poisoning attacks to federated learning based on fake clients”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 3396–3404.
- [40] M. T. Hossain, S. Islam, S. Badsha, and H. Shen, “Desmp: Differential privacy-exploited stealthy model poisoning attacks in federated learning”, in *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, IEEE, 2021, pp. 167–174.
- [41] H. Li, X. Sun, and Z. Zheng, “Learning to attack federated learning: A model-based reinforcement learning attack framework”, *Advances in Neural Information Processing Systems*, vol. 35, pp. 35 007–35 020, 2022.

- [42] Y. Zhe, R. Nagaike, D. Nishiyama, K. Fukuchi, and J. Sakuma, *Adversarial attacks on hidden tasks in multi-task learning*, 2024. arXiv: 2405.15244 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2405.15244>.
- [43] L. Zhang, X. Liu, K. Mahmood, C. Ding, and H. Guan, *Multi-task models adversarial attacks*, 2023. arXiv: 2305.12066 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2305.12066>.
- [44] Y. Wan, Y. Qu, W. Ni, Y. Xiang, L. Gao, and E. Hossain, *Data and model poisoning backdoor attacks on wireless federated learning, and the defense mechanisms: A comprehensive survey*, 2023. arXiv: 2312.08667 [cs.CR]. [Online]. Available: <https://arxiv.org/abs/2312.08667>.
- [45] J. Xu, Z. Chen, T. Q. Quek, and K. F. E. Chong, “Fedcorr: Multi-stage federated learning for label noise correction”, in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10 184–10 193.
- [46] B. Zeng, X. Yang, Y. Chen, H. Yu, and Y. Zhang, “Clc: A consensus-based label correction approach in federated learning”, *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 5, pp. 1–23, 2022.
- [47] N. M. Jebreel, J. Domingo-Ferrer, D. Sánchez, and A. Blanco-Justicia, “Lfighter: Defending against the label-flipping attack in federated learning”, *Neural Networks*, vol. 170, pp. 111–126, 2024.
- [48] B. Biggio, B. Nelson, and P. Laskov, *Poisoning attacks against support vector machines*, 2013. arXiv: 1206.6389 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1206.6389>.

# Abbreviations

|         |                                                           |
|---------|-----------------------------------------------------------|
| FL      | Federated Learning                                        |
| ML      | Machine Learning                                          |
| DFL     | Decentralized Federated Learning                          |
| MTL     | Multi-Task Learning                                       |
| FMTL    | Federated Multi-Task Learning                             |
| CFMTL   | Centralized Federated Multi-Task Learning                 |
| DFMTL   | Decentralized Federated Multi-Task Learning               |
| HDFMTL  | Heterogeneous Decentralized Federated Multi-Task Learning |
| Non-IID | Non-Independently and Non-Identically Distributed         |
| AT2FL   | Attack on Federated Learning                              |
| FGSM    | Fast Gradient Sign Method                                 |
| BN      | Batch Normalization                                       |
| HCA     | Hyper Conflict-Averse                                     |
| FedPer  | Federated Personalization                                 |
| ResNet  | Residual Network                                          |
| MSE     | Mean Squared Error                                        |
| F1      | F1-Score                                                  |
| TPR     | True Positive Rate                                        |
| TNR     | True Negative Rate                                        |



# List of Figures

|     |                                                                                                                                  |    |
|-----|----------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1 | Illustration of FL framework proposed by Google [16] . . . . .                                                                   | 6  |
| 2.2 | CFL, DFL and Semi-DFL as depicted by [4] . . . . .                                                                               | 7  |
| 2.3 | FedPer-HCA framework based on Nicolas et al. [9]. Numbers indicate the order of operations in one global training round. . . . . | 15 |
| 3.1 | A taxonomy of MTL attack types, including poisoning attacks, adversarial attacks, and novel hybrid attacks . . . . .             | 22 |
| 4.1 | Overall Attack Design . . . . .                                                                                                  | 32 |
| 5.1 | Top-level directory structure of the project. . . . .                                                                            | 42 |
| 5.2 | Directory layout within the <b>src/</b> folder. . . . .                                                                          | 42 |
| 5.3 | Directory layout under <b>results/</b> . Each subfolder corresponds to a different attack type. . . . .                          | 44 |
| 6.1 | Performance metrics for CIFAR-10 clients (Random Label Flip). . . . .                                                            | 51 |
| 6.2 | Clean vs poisoned test results for each CIFAR-10 client (Random Label Flip). . . . .                                             | 51 |
| 6.3 | Performance metrics for CelebA clients (Random Label Flip). . . . .                                                              | 53 |
| 6.4 | Clean vs poisoned test results for each CelebA client (Random Label Flip). . . . .                                               | 53 |
| 6.5 | Performance metrics for CIFAR-10 clients (Targeted Label Flip). . . . .                                                          | 54 |
| 6.6 | Clean vs poisoned test results for each CIFAR-10 client (Targeted Label Flip). . . . .                                           | 55 |
| 6.7 | Performance metrics for CelebA clients (Targeted Label Flip). . . . .                                                            | 55 |
| 6.8 | Clean vs poisoned test results for each CelebA client (Targeted Label Flip). . . . .                                             | 55 |
| 6.9 | CelebA – performance metrics, trigger patch ratio 0.3. . . . .                                                                   | 57 |

|      |                                                                                      |    |
|------|--------------------------------------------------------------------------------------|----|
| 6.10 | CelebA – clean vs poisoned results, trigger patch ratio 0.3. . . . .                 | 57 |
| 6.11 | CelebA – performance metrics, trigger patch ratio 0.8. . . . .                       | 57 |
| 6.12 | CelebA – clean vs poisoned results, trigger patch ratio 0.8. . . . .                 | 58 |
| 6.13 | CIFAR-10 – performance metrics, trigger patch ratio 0.3. . . . .                     | 58 |
| 6.14 | CIFAR-10 – clean vs poisoned results, trigger patch ratio 0.3. . . . .               | 58 |
| 6.15 | CIFAR-10 – performance metrics, trigger patch ratio 0.8. . . . .                     | 59 |
| 6.16 | CIFAR-10 – clean vs poisoned results, trigger patch ratio 0.8. . . . .               | 59 |
| 6.17 | Performance metrics for CIFAR-10 clients (Sign Flip). . . . .                        | 61 |
| 6.18 | Clean vs poisoned test results for CIFAR-10 (Sign Flip). . . . .                     | 61 |
| 6.19 | Performance metrics for CelebA clients (Sign Flip). . . . .                          | 62 |
| 6.20 | Clean vs poisoned test results for CelebA (Sign Flip). . . . .                       | 62 |
| 6.21 | CelebA – performance metrics (Scaled Boost, $5\times$ ). . . . .                     | 63 |
| 6.22 | CelebA – clean vs poisoned results (Scaled Boost, $5\times$ ). . . . .               | 63 |
| 6.23 | CIFAR-10 – performance metrics (Scaled Boost, $5\times$ ). . . . .                   | 64 |
| 6.24 | CIFAR-10 – clean vs poisoned results (Scaled Boost, $5\times$ ). . . . .             | 64 |
| 6.25 | Performance metrics for CIFAR-10 clients (AT2FL). . . . .                            | 66 |
| 6.26 | Clean vs. poisoned test results for each CIFAR-10 client (AT2FL). . . . .            | 66 |
| 6.27 | Performance metrics for CelebA clients (AT2FL). . . . .                              | 67 |
| 6.28 | Clean vs. poisoned test results for each CelebA client (AT2FL). . . . .              | 67 |
| 6.29 | Performance metrics for CIFAR-10 clients (Malicious Aggregation). . . . .            | 69 |
| 6.30 | Clean vs. poisoned results for each CIFAR-10 client (Malicious Aggregation). . . . . | 69 |
| 6.31 | Performance metrics for CelebA clients (Malicious Aggregation). . . . .              | 70 |
| 6.32 | Clean vs. poisoned results for each CelebA client (Malicious Aggregation). . . . .   | 70 |

# List of Tables

|     |                                                                  |    |
|-----|------------------------------------------------------------------|----|
| 2.1 | Spectrum of Federated Multi-Task Learning Architectures. . . . . | 11 |
| 4.1 | Summary of Poisoning Attacks in DFMTL . . . . .                  | 39 |
| 5.1 | Poisoning Attack Configurations and Modules . . . . .            | 43 |