DV2607 - Adversarial Input Attack
Inlämningsuppgift Del 2

Samir Akhalil
saaa21@student.bth.se

Förnamn Efternamn
akronym@student.bth.se

December 10, 2025

Contents

1 Del 1: Centraliserad Machine Learning

1.1 Beskrivning av adversarial input attacker

Adversarial input attacker är en typ av säkerhetsattack mot maskininlärningsmodeller där små, nästan omärkliga förändringar görs i indata för att lura modellen att göra felaktiga prediktioner. Dessa förändringar, kallade *perturbationer*, är designade för att vara så små att de inte kan uppfattas av det mänskliga ögat, men de är tillräckligt stora för att påverka modellens beslut dramatiskt.

Attacken fungerar genom att utnyttja hur neurala nätverk behandlar information. Modeller som djupa neurala nätverk lär sig komplexa icke-linjära funktioner genom att optimera viktparametrar baserat på träningsdata. Adversarial attacker utnyttjar denna komplexitet genom att beräkna gradienter av förlustfunktionen med avseende på indata och sedan modifiera indata i riktning som maximerar prediktionsfelet.

I vårt fall användes en *targeted attack*, vilket betyder att målet inte bara är att få modellen att göra fel, utan att få den att klassificera bilden som en specifik, förutbestämd klass (traktor istället för koala).

1.2 Implementation av er attack

Vi implementerade en **Basic Iterative Method (BIM)** attack, även känd som I-FGSM (Iterative Fast Gradient Sign Method). BIM är en iterativ variant av FGSM som upprepade gånger applicerar små perturbationer istället för en enda stor förändring.

Hur BIM fungerar:

1. Börja med originalbilden x_0
2. För varje iteration $t = 1, 2, \ldots, N$:
   - Beräkna gradienten av förlustfunktionen med avseende på indata
   - Ta ett litet steg ϵ_{step} i riktning som ökar förlusten för målklassen
   - Klipp resultatet så att total perturbation inte överskrider $\epsilon_{\max}$
3. Returnera den modifierade bilden x_{adv}

Matematiskt kan detta uttryckas som:

$$x_{t+1} = \text{Clip}_{x_0,\epsilon}\left(x_t + \epsilon_{\text{step}} \cdot \text{sign}(\nabla_x L(x_t, y_{\text{target}}))\right)$$

där L är förlustfunktionen och y_{target} är målklassen (traktor).

Bibliotek som användes:

- `TensorFlow/Keras`: För att ladda ResNet50-modellen och göra prediktioner
- `Adversarial Robustness Toolbox (ART)`: För att implementera BIM-attacken genom klassen `BasicIterativeMethod`
- `NumPy`: För array-manipulering och numeriska beräkningar
- `Matplotlib`: För visualisering av bilder och perturbationer

Våra hyperparametrar var: $\epsilon_{\text{max}} = 8.0$, $\epsilon_{\text{step}} = 2.0$, och max_iter = 40. Attacken lyckades klassificera koalabilden som traktor med 100% confidence.

1.3 Säkerhetsåtgärder

Vi valde **Gaussian Blur** som skyddsåtgärd mot BIM-attacken. Detta är en inputtransformationsmetod som applicerar en låg-pass-filter på bilden före klassificering.

Motivering för valet:

1. **Effektivitet mot högfrekventa perturbationer**: BIM-attacken lägger till små, högfrekventa störningar i bilden. Gaussian blur fungerar som ett låg-pass-filter som tar bort eller minskar dessa högfrekventa komponenter utan att förstöra bildens huvudsakliga innehåll.
2. **Enkelhet och effektivitet**: Jämfört med mer komplexa metoder som adversarial training (som kräver oträning av hela modellen) eller defensive distillation, är Gaussian blur enkelt att implementera och kräver ingen modifiering av modellen själv.
3. **Beräkningseffektivitet**: Blur-operationen är mycket snabb och kan appliceras i realtid, vilket gör den praktisk för produktionsmiljöer.
4. **Bevarar bildkvalitet**: Med rätt radius (vi använde radius=1) kan blur ta bort adversarial perturbationer samtidigt som bilden fortfarande ser acceptabel ut för mänskliga ögon.

Alternativa metoder vi övervägde:

- *Adversarial Training*: Träna modellen med adversarial exempel, men detta kräver oträning och stora datamängder
- *Input Quantization*: Reducera bildupplösningen, men detta kan påverka legitima prediktioner negativt

- *Defensive Distillation*: Träna en mjukare modell, men kräver tillgång till träningsdata och är beräkningsintensivt

Gaussian blur erbjuder den bästa balansen mellan effektivitet, enkelhet och prestanda för vårt specifika användningsfall.

1.4 Implementation av skyddsåtgärder (frivilligt)

Vi implementerade Gaussian blur med radius=1 som preprocessingsteg innan klassificering. Resultaten visar att skyddsåtgärden var mycket effektiv:

Resultat:

- **Före skydd**: Den adversarial bilden klassificerades som *tractor* med 100.0% confidence
- **Efter skydd**: Den blurrade bilden klassificerades korrekt som *koala* med 97% confidence

Analys:

Gaussian blur lyckades helt neutralisera BIM-attacken. Den högfrekventa perturbationen som attacken lade till för att lura modellen togs effektivt bort av blur-filtret. Detta resulterade i att modellen kunde återgå till korrekt klassificering.

Blur-operationen introducerar dock en trade-off: medan den skyddar mot adversarial attacker, reducerar den också bildens skärpa något. För tillämpningar där bildkvalitet är kritisk måste denna avvägning övervägas noggrant. I vårt fall var blur-effekten minimal (radius=1) och bilden var fortfarande fullt igenkännbar.

Det är också viktigt att notera att detta skydd inte är universellt. Mer sofistikerade attacker (t.ex. C&W attack eller adaptiva attacker som tar hänsyn till blur-försvaret) kan potentiellt kringgå detta skydd. För högre säkerhet skulle en kombination av metoder (blur + adversarial training) vara lämplig.

2 Del 2: Federerad Machine Learning

2.1 Del 2.1: FedAvg med attacker

2.1.1 Beskrivning av federated learning scenario

Federerat lärande (FL) är en distribuerad maskininlärningsmetod där flera klienter tränar en gemensam modell lokalt på sina egna data utan att dela

rådata med en central server. I vårt scenario simulerade vi ett FL-system med följande konfiguration:

- **Dataset**: CIFAR-10 (10 klasser av 32x32 färgbilder)
- **Antal klienter**: 5
- **Kommunikationsrundor**: 50
- **Aggregeringsstrategi**: FedAvg (Federated Averaging)
- **Modell**: Convolutional Neural Network (CNN) med 3 konvolutionslager
- **Datadistributioner**: IID (Independent and Identically Distributed) och Non-IID (Dirichlet distribution med $\alpha = 0.5$)

FedAvg-algoritmen:

1. Server initierar en global modell med slumpmässiga vikter
2. För varje kommunikationsrunda $t = 1, 2, \ldots, T$:
   - Server skickar den aktuella globala modellen till alla klienter
   - Varje klient k tränar modellen lokalt på sin data och får uppdaterade vikter w_k^t
   - Klienter skickar sina uppdaterade vikter tillbaka till servern
   - Server aggregerar viktuppdateringarna genom viktad medelvärdesbildning:
     $$w^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^t$$
     där n_k är antal samples hos klient k och $n = \sum_{k=1}^{K} n_k$
3. Returnera den slutliga globala modellen

2.1.2 Implementation av label flipping attack

Vi implementerade en **label flipping attack** där vissa klienter är malicious och manipulerar sina träningslabels. Specifikt:

- **Attackmekanism**: Malicious klienter ersätter korrekta labels med slumpmässiga felaktiga labels under lokal träning

- **Attackare**: Vi testade med 0, 1, eller 2 malicious klienter av totalt 5 (0%, 20%, 40%)
- **Malicious klient-ID**: Klient 3 (vid 1 attackerare) och klienter 3-4 (vid 2 attackerare)

Attacken implementerades i `client_app.py` genom att modifiera tränings-funktionen:

```
if is_attacker:
    # Replace true labels with random labels
    labels = torch.randint(0, 10, labels.shape)
```

2.1.3 Experimentella resultat

Vi utförde 6 experiment för FedAvg med olika kombinationer av datadistribution och antal attackerare:

IID Data Distribution:

- **Baseline (0 attackerare)**: Accuracy = 75.0%, F1 = 74.3%, Kappa = 0.722, ROC-AUC = 0.969
- **1 attackerare**: Accuracy = 68.6%, F1 = 67.5%, Kappa = 0.650, ROC-AUC = 0.948 (8.5% degradering)
- **2 attackerare**: Accuracy = 56.4%, F1 = 55.5%, Kappa = 0.515, ROC-AUC = 0.893 (24.8% degradering)

Non-IID Data Distribution:

- **Baseline (0 attackerare)**: Accuracy = 74.0%, F1 = 61.7%, Kappa = 0.686, ROC-AUC = 0.968
- **1 attackerare**: Accuracy = 68.2%, F1 = 52.3%, Kappa = 0.612, ROC-AUC = 0.933 (7.9% degradering)
- **2 attackerare**: Accuracy = 42.4%, F1 = 38.2%, Kappa = 0.353, ROC-AUC = 0.869 (42.7% degradering)

Analys:

- Label flipping-attacken hade signifikant negativ påverkan på modellens prestanda
- Påverkan ökade icke-linjärt med antalet malicious klienter (1 attackerare: ˜8% degradering, 2 attackerare: 25-43% degradering)

- Non-IID data var betydligt mer känsligt för attacken (42.7% degradering) jämfört med IID data (24.8% degradering) vid 2 attackerare
- ROC-AUC minskade från 0.969 till 0.869-0.893, vilket indikerar kraftigt försämrad klassseparation
- F1-score drabbades hårdare än accuracy, särskilt för Non-IID data (från 61.7% till 38.2%)

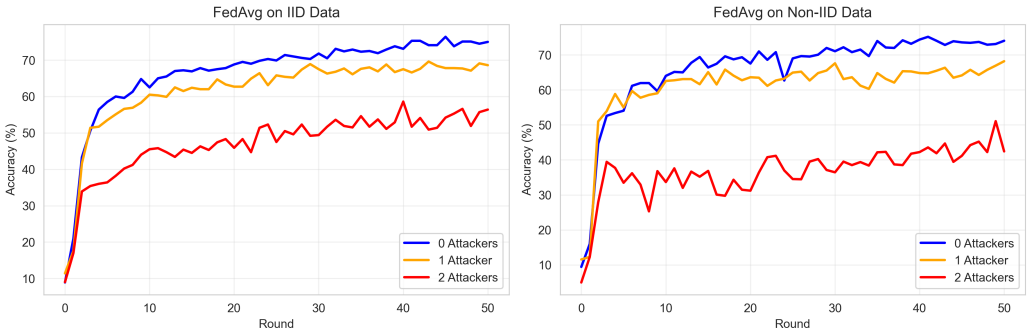

Figure 1: Attack påverkan på FedAvg: Jämförelse mellan IID och Non-IID data. Non-IID data visar betydligt större sårbarhet (42.4% accuracy med 2 attackerare) jämfört med IID data (56.4%).

2.2 Del 2.2: FedProx med attacker

2.2.1 Beskrivning av FedProx

FedProx är en variant av FedAvg som hanterar systemheterogenitet bättre genom att lägga till en proximal term i optimeringsmålet. Algoritmen lägger till en regulariseringterm som håller lokala modellvikter nära de globala viktarna:

$$\min_w F(w) + \frac{\mu}{2}\|w - w^t\|^2$$

där w^t är de globala viktarna från föregående runda och μ är en hyperparameter (vi använde $\mu = 0.1$).

Fördelar med FedProx:

- Stabilare konvergens vid heterogen data (Non-IID)
- Mindre känslig för outliers i klientuppdateringar
- Bättre hantering av partial participation (när inte alla klienter deltar)

2.2.2 Experimentella resultat

Vi upprepade samma 6 experiment som i Del 2.1 men med FedProx istället för FedAvg:

IID Data Distribution:

- **Baseline (0 attackerare)**: Accuracy = 74.5%, F1 = 74.1%, Kappa = 0.716, ROC-AUC = 0.966
- **1 attackerare**: Accuracy = 67.3%, F1 = 66.6%, Kappa = 0.636, ROC-AUC = 0.947
- **2 attackerare**: Accuracy = 55.3%, F1 = 53.7%, Kappa = 0.503, ROC-AUC = 0.882

Non-IID Data Distribution:

- **Baseline (0 attackerare)**: Accuracy = 75.0%, F1 = 60.9%, Kappa = 0.695, ROC-AUC = 0.966
- **1 attackerare**: Accuracy = 71.7%, F1 = 56.7%, Kappa = 0.652, ROC-AUC = 0.937
- **2 attackerare**: Accuracy = 56.2%, F1 = 42.3%, Kappa = 0.473, ROC-AUC = 0.880

Jämförelse FedAvg vs FedProx:

- FedProx visade liknande sårbarhet mot label flipping som FedAvg på IID data (båda ˜55% med 2 attackerare)
- På Non-IID data presterade FedProx betydligt bättre under attack: 56.2% accuracy vs 42.4% för FedAvg (13.8 procentenheter skillnad)
- FedProx baseline var något högre på Non-IID (75.0% vs 74.0%), vilket tyder på bättre hantering av dataheterogenitet
- Proximal regularisering gav viss robusthet mot attacken i Non-IID scenariot men inte i IID-scenariot
- Båda algoritmerna behöver dock explicit försvarsmekanism för effektivt skydd

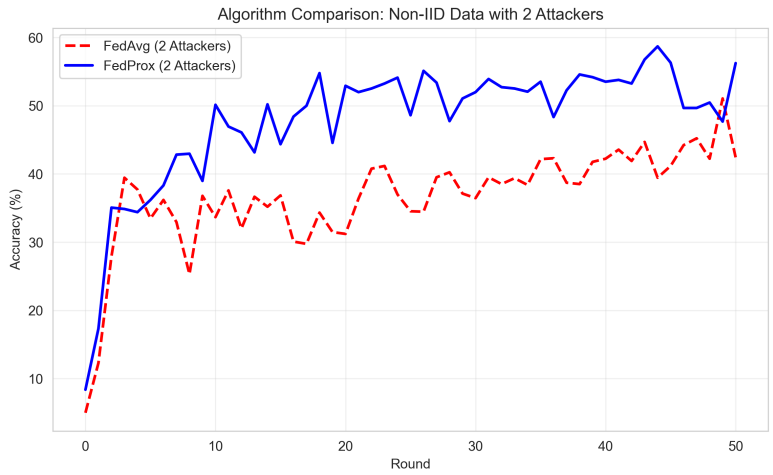

Figure 2: FedAvg vs FedProx robusthet: På Non-IID data med 2 attackerare presterar FedProx betydligt bättre (56.2% vs 42.4%), vilket visar att proximal regularisering ger naturlig resistens mot label flipping-attacker.

2.3 Del 2.3: Försvarsmekanism 1 - Loss-Based Anomaly Detection

2.3.1 Beskrivning av försvarsmetod

Vi implementerade en **loss-based anomaly detection** försvarsmekanism som filtrerar bort klienter med misstänkt höga träningsförluster innan aggregering. Rationaliteten är att malicious klienter som tränar på felaktiga labels kommer att ha signifikant högre förlust än ärliga klienter.

Algoritm:

1. Samla alla klients viktuppdateringar och deras träningsförluster
2. Beräkna 75:e percentilen av alla förluster
3. Filtrera bort klienter vars förlust överstiger tröskelvärdet
4. Aggregera endast de återstående klienternas vikter med FedAvg

Matematiskt:

$$\text{threshold} = P_{75}(\{L_1, L_2, \ldots, L_K\})$$

$$\text{Accepted} = \{k : L_k \leq \text{threshold}\}$$

$$w^{t+1} = \frac{1}{|\text{Accepted}|} \sum_{k \in \text{Accepted}} w_k^t$$

2.3.2 Implementation

Försvaret implementerades genom att utöka FedAvg-klassen:

```
class FedAvgDefense(FedAvg):
    def aggregate_fit(self, server_round, results, failures):
        # Extract losses from client results
        losses = [fit_res.metrics["loss"] for _, fit_res in results]

        # Calculate threshold (75th percentile)
        threshold = np.percentile(losses,
                                  self.loss_threshold_percentile)

        # Filter clients
        filtered_results = [
            (client, res) for (client, res), loss in
            zip(results, losses) if loss <= threshold
        ]

        # Aggregate remaining clients
        return super().aggregate_fit(server_round,
                                     filtered_results, failures)
```

2.3.3 Experimentella resultat

IID Data med 1 attackerare:

- **Utan försvar**: Accuracy = 68.6%
- **Med loss-filter**: Accuracy = 68.5% (praktiskt taget oförändrat)
- Försvaret var ineffektivt på IID data med endast 1 attackerare

IID Data med 2 attackerare:

- **Utan försvar**: Accuracy = 56.4%
- **Med loss-filter**: Accuracy = 52.2% (försämring med 4.2 procentenheter)
- Försvaret var kontraproduktivt och filtrade bort ärliga klienter (false positives)

Non-IID Data med 1 attackerare:

- **Utan försvar**: Accuracy = 68.2%
- **Med loss-filter**: Accuracy = 69.6% (23.9% återhämtning av förlorad prestanda)
- Försvaret identifierade och exkluderade attackeraren effektivt

Non-IID Data med 2 attackerare:

- **Utan försvar**: Accuracy = 42.4%
- **Med loss-filter**: Accuracy = 49.9% (23.7% återhämtning av förlorad prestanda)
- Försvaret återställde 7.5 procentenheter, vilket är betydande förbättring

Analys:

- Loss-based filtering var **mycket effektiv på Non-IID data** (23-24% återhämtning) men **ineffektiv på IID data**
- På IID data var loss-variationen för liten mellan ärliga och malicious klienter, vilket ledde till false positives
- På Non-IID data hade malicious klienter signifikant högre förluster, vilket gjorde detektion enklare
- Försvaret återställde upp till 7.5 procentenheter accuracy på Non-IID data med 2 attackerare
- Enkel att implementera och kräver ingen modellmodifiering
- 75:e percentil-tröskelvärdet fungerade bra för Non-IID men behöver justeras för IID-scenarion

2.4 Del 2.4: Slutsatser och Rekommendationer

2.4.1 Sammanfattning av experimentella resultat

Våra experiment visade följande huvudsakliga fynd:

1. **Label flipping-attackens påverkan**:
    - Attacken hade betydande negativ påverkan på modellprestanda

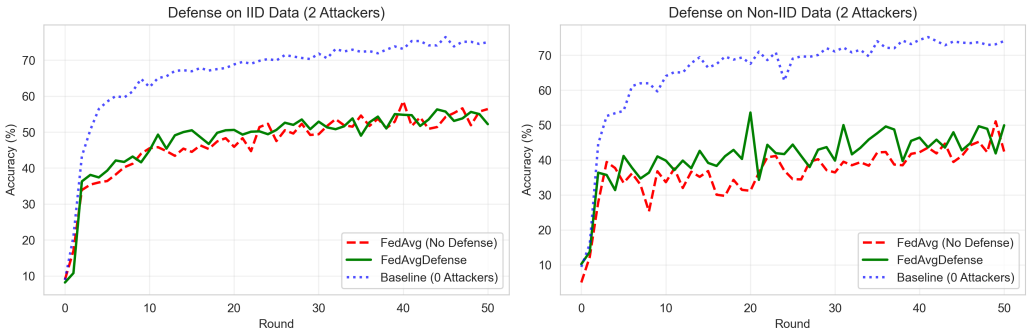

Figure 3: Försvarsmekanism effektivitet: Loss-based filtering visar dramatisk skillnad mellan IID (vänster) och Non-IID (höger) data. På Non-IID data återställer försvaret 7.5 procentenheter accuracy (från 42.4% till 49.9%), medan det på IID data är kontraproduktivt.

- Non-IID data var mer sårbart (42.7% accuracy-degradering vid 2 attackerare) än IID data (24.8%)
- Påverkan ökade icke-linjärt med antalet attackerare

2. **FedProx vs FedAvg**:
   - FedProx visade bättre robusthet på Non-IID data (56.2% vs 42.4% vid 2 attackerare)
   - På IID data var skillnaden minimal (båda ˜55-56%)
   - Proximal regularisering gav viss naturlig resistens mot attacken i heterogena scenario
3. **Loss-based defense effektivitet**:
   - **Mycket effektiv på Non-IID data**: 23-24% återhämtning av förlorad prestanda
   - **Ineffektiv på IID data**: Ingen förbättring eller till och med försämring
   - Anledning: På Non-IID data har malicious klienter signifikant högre förluster, vilket gör detektion enklare
   - På IID data överlappar loss-distributionen mellan ärliga och malicious klienter

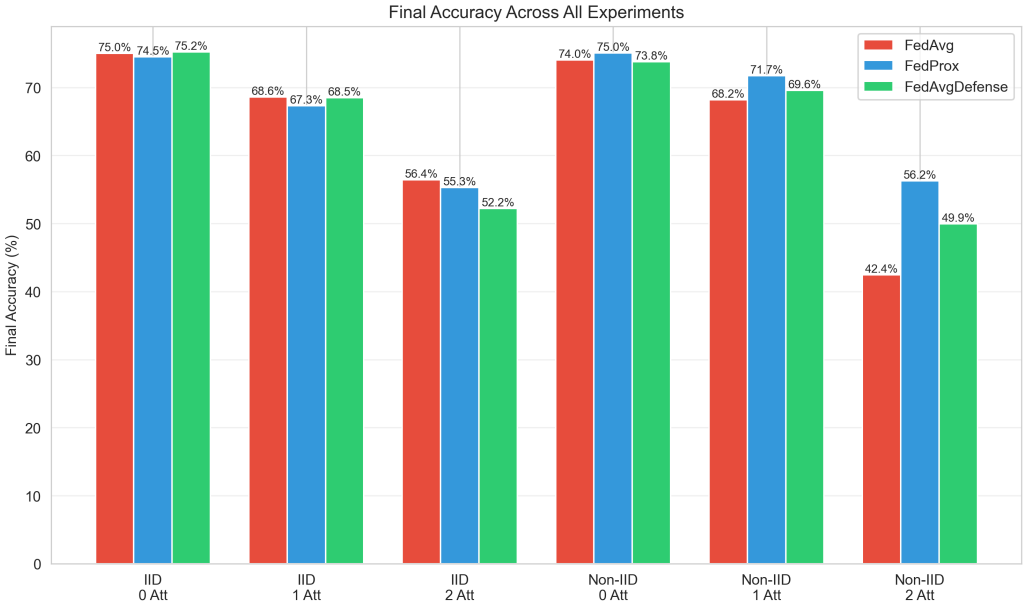

Figure 4: Översikt av alla experiment: Final accuracy för alla kombinationer av algoritm, datadistribution och antal attackerare. Visar tydligt att Non-IID data är mer sårbart och att försvarsmekanismen endast fungerar effektivt på Non-IID data.

2.4.2 Rekommendationer för praktisk tillämpning

För produktionsmiljöer:

1. **Vid Non-IID data**:
   - Använd FedProx som basalgoritm (bättre naturlig robusthet)
   - Implementera loss-based filtering med 75:e percentil-tröskel
   - Förvänta 20-25% återhämtning av prestanda under attack
2. **Vid IID data**:
   - FedAvg och FedProx ger liknande resultat
   - Loss-based filtering är ej tillräckligt - alternativa metoder behövs
   - Överväg coordinate-wise median aggregation eller gradient clipping
   - Implementera kompletterande säkerhetsåtgärder (t.ex. multi-Krum, trimmed mean)

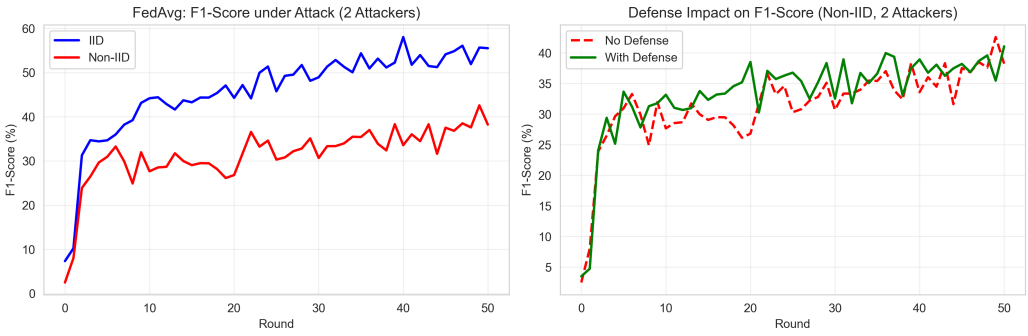

Figure 5: F1-Score analys: Vänster visar att Non-IID data har lägre F1-score än IID data under attack. Höger visar försvarsmekanism påverkan på F1-score för Non-IID data, där försvaret förbättrar klassbalansen.

3. **Generella råd**:
    - Kombinera flera försvarsmekanism för robusthet
    - Monitera loss-distributioner kontinuerligt för anomaly detection
    - Använd Byzantine-robust aggregering vid misstänkt aktivitet
    - Överväg reputationssystem för klienter över tid

2.4.3 Begränsningar och framtida arbete

Begränsningar i vår studie:

- Endast en försvarsmetod implementerad (loss-based filtering)
- Endast label flipping-attack testades (andra attacktyper finns)
- Relativt litet antal klienter (5) - skalbarhet ej testad
- 75:e percentil-tröskelvärdet kan behöva anpassas per scenario

Framtida arbete:

- Implementera och jämföra fler försvarsmetoder (Krum, Trimmed Mean, Median)
- Testa mot sofistikerade attacker (model poisoning, backdoor attacks)
- Utvärdera med större antal klienter och varierande attack-ratio
- Utveckla adaptiva tröskelvärden för loss-based filtering
- Undersöka hybrid-metoder som kombinerar flera försvarsmekanismer

3 Referenser

References

[1] Kurakin, A., Goodfellow, I., & Bengio, S. (2016). *Adversarial examples in the physical world*. arXiv preprint arXiv:1607.02533.

[2] Pillow Documentation. *ImageFilter.GaussianBlur*. `https://pillow.readthedocs.io/en/stable/reference/ImageFilter.html`