UNLOCKING THE HIDDEN POTENTIAL OF CLIP IN GENERALIZABLE DEEPFAKE DETECTION

Link - https://arxiv.org/pdf/2503.19683

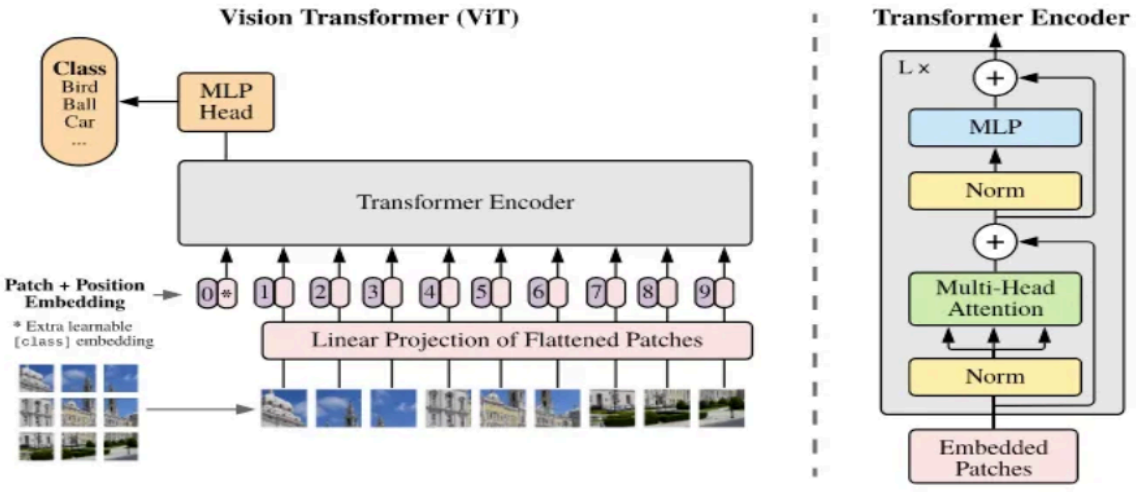
Abstract

This work addresses the challenge of detecting partially manipulated facial deepfakes, which are more difficult to identify than fully synthetic faces due to subtle modifications. It leverages the CLIP ViT-L/14 visual encoder for a generalizable deepfake detection approach, incorporating parameter-efficient fine-tuning (PEFT)—notably LN-tuning—to fine-tune minimal parameters while retaining CLIP's pre-trained knowledge. A specialized preprocessing pipeline for facial images, along with L2 normalization and metric learning on a hyperspherical manifold, further improves generalization.

Approach

The approach uses the **CLIP ViT-L/14** model as a frozen visual backbone to extract robust image features. Parameter-Efficient Fine-Tuning (PEFT) with **Layer Normalization (LN) tuning** is applied, fine-tuning only a small subset of normalization parameters to adapt the model efficiently without overfitting. A **linear classification head with normalization** is trained on top for binary deepfake detection. The training process uses SLERP-based feature augmentation, a cosine learning rate scheduler, and cross-entropy loss to improve robustness and generalization. This setup balances leveraging pre-trained knowledge with lightweight, targeted fine-tuning to detect deepfakes effectively.

Artitecture



Note - only targeting image encoder part of clip model i.e ViT

Background: How CLIP-ViT-L/14 Works in our case !

1. Input image
   - image size = 224×224×3
   - we give this image to the model for encoding.

2. Patchify image + Linear Projection
   - Patch size = **14×14**
   - So: 224/14 = 16 patches per side→16×16=256 patches total
   - Each patch shape = `14×14×3 = 588` → flattened into **1D vector of size 588**
   - So the **patch embedding matrix**: X-patches 256*1024 (because CLIP projects 588 → 1024 using a learned linear layer)

3. Add Learnable [CLS] Token
   - one learnable `[CLS]` token of shape `[1 × 1024]` is prepended.
   - New input: X-input : 257*1024

4. Add Positional Embeddings
   - Learnable position embeddings of shape `[257 × 1024]` are added:
   - This helps preserve **patch order** and **spatial location**.

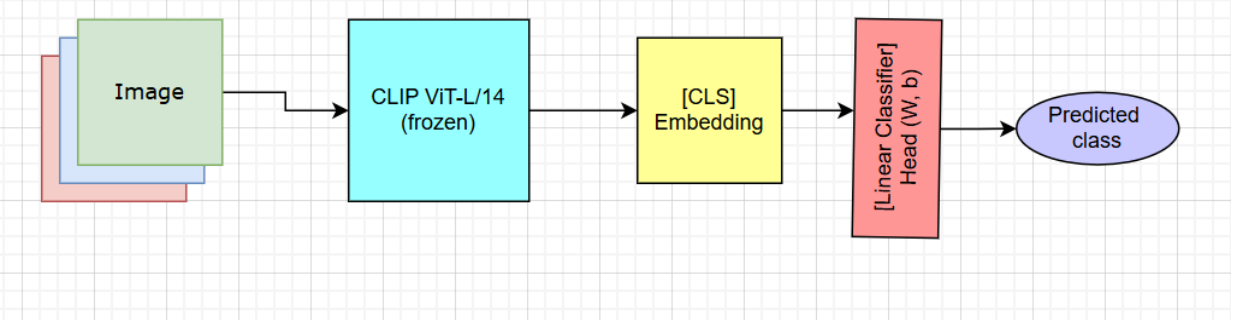
5. Transformer Encoder (24 layers in ViT-L/14)
   - Each of the 24 layers contains:
     - LayerNorm
     - Multi-Head Self-Attention (with Q/K/V projections)
     - Residual Connection
     - MLP Block
     - Another Residual
     - Each layer **processes the full sequence of 257 tokens** (`[CLS] + 256 patches`) at once.
   - After 24 layers: Z-final : 257*1024

6. Extract the `[CLS]` Token
   - `[CLS]` token is at position 0
   - A **1D vector** Of **dimension 1024** Representing the **entire image**
   - Final image representation: z_cls 1024
   - `[CLS]` embedding as a **feature vector**.

Now further we will be using this z_cls token in different methods like Linear Probing , LN-Tuning , LN-Tuning + Norm , LN-Tuning + Norm + Slerp

Linear Probing

Linear probing = using the **frozen encoder** + **trainable linear classifier** on top of the `z_cls` token.



- Freeze all parameters of the CLIP visual encoder (ViT-L/14)
- Train a simple linear layer to classify the image as:
  - Real (label = 0)
  - Fake (label = 1)

This sets the baseline performance for the rest of the paper.

Step	Description
1. Input	224×224×3 image
2. Visual Encoder	Extract CLS token (frozen encoder)
3. z_cls	Shape: 1024-dim vector
4. Classifier	`Linear(1024 → 2)`
5. Loss	Cross-entropy
6. Learnable params	Only W (1024×2) and b (2,)

LN-Tuning

Instead of training the entire ViT encoder or even just the classifier head, **we fine-tune only the LayerNorm (LN) parameters** inside the ViT blocks.

This is a **parameter-efficient fine-tuning (PEFT)** method, meaning:

- Only a **tiny subset of parameters** is trainable
- All other CLIP weights are **frozen**

These are the exact module name substrings used to find all `LayerNorm` instances inside CLIP's visual encoder.

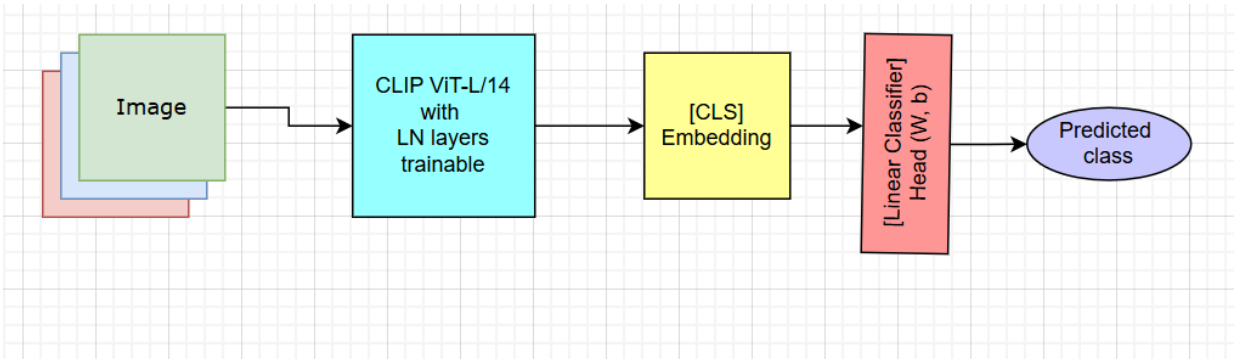
It tries to match these inside model layer names:

Name	Meaning
pre_layrnorm	LN applied before transformer begins
layer_norm1	LN before self-attention in each block
layer_norm2	LN before MLP in each block
post_layernorm	LN applied after last transformer layer
layernorm	Catch-all fallback (in case of different naming style)

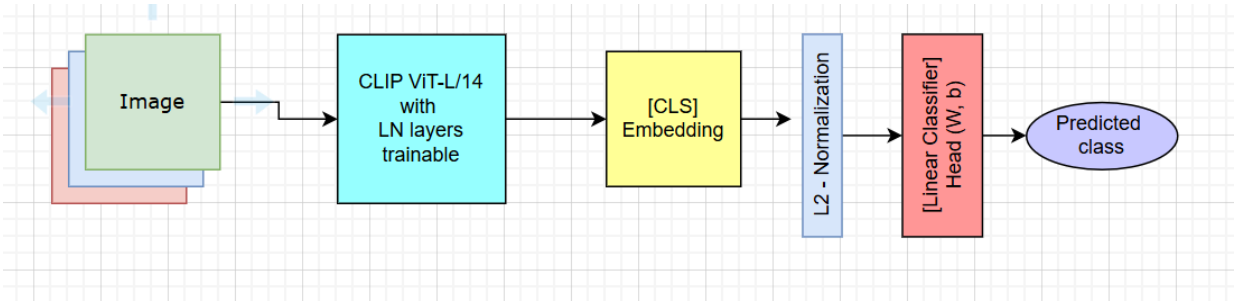
During model initialization , fine-tuning setup, the script will:

1. Loop over all modules in the CLIP model
2. Match module names that contain any of these substrings
3. Set `requires_grad = True` for:
    - `.weight` = γ
    - `.bias` = β

Everything else remains frozen.



LN-Tuning + Norm



- LayerNorm tuning **+** L2 normalization of the extracted `z_cls` embedding.
- This adds a **simple but powerful regularization trick** on top of LN-Tuning.
- Extracting `z_cls` from CLIP ViT-L/14
- Training **only the LayerNorm γ & β parameters**

But now, before feeding `z_cls` to the classifier:

- **normalize** it with **L2 norm**
- **L2 normalize embeddings**
- This constraints `z_cls` to lie on a **hypersphere**
- **The norm (magnitude) of each vector is the same:**

What is L2 Normalization?

$$\mathbf{z}_{\text{cls}} \in \mathbb{R}^{1024}$$

$$\hat{\mathbf{z}} = \frac{\mathbf{z}_{\text{cls}}}{\|\mathbf{z}_{\text{cls}}\|_2}$$

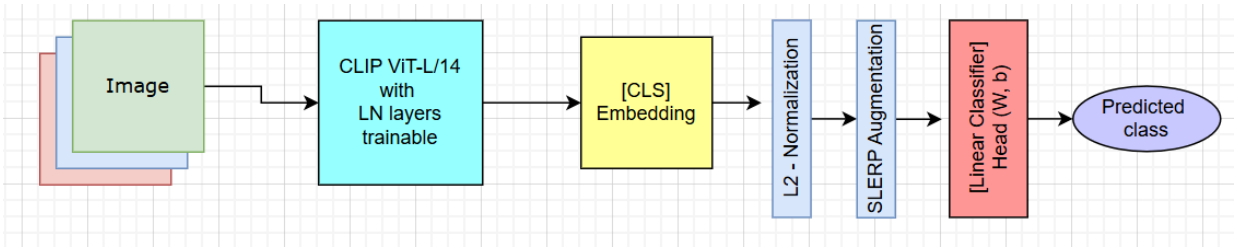
- $\|\hat{\mathbf{z}}\|_2 = 1$
- Now $\hat{\mathbf{z}}$ lies on the surface of a **unit hypersphere** in 1024-dimensional space.

Why Normalize to a Hypersphere?

Benefits:

- Forces the model to focus on angular differences, not magnitude
- Improves generalization to unseen manipulations
- Reduces overfitting (like in face recognition)

LN-Tuning + Norm + Slerp



Dataset information and Preprocessing Steps

Dataset	Subsets	Description
FaceForensics (FF++)	- DF (Deepfakes) - FS (FaceSwap) - F2F (Face2Face) - NT (NeuralTextures) - Real	Benchmark dataset for detecting manipulated facial videos using various forgery techniques.
Celeb-DF v1	- Celeb-real - Celeb-synthesis - YouTube-real	Contains high-quality deepfake videos with better visual realism than previous datasets.
Celeb-DF v2	- Celeb-real - Celeb-synthesis - YouTube-real	Extended version of v1 with more deepfake videos and greater diversity for robust generalization tests.

Data Preprocessing -(link)

➔ **Frame Extraction**:
Raw videos from datasets like FaceForensics++, Celeb-DF v1, and v2 are processed using `extract_frames.py`.

- This script extracts frames at a fixed interval (commonly 1 frame per second or 10 fps depending on the config).
- Extracted frames are stored in organized directories with one folder per video.
- This reduces the computational burden of processing entire videos and provides a consistent set of visual samples for detection.
-

➔ **Face Detection & Alignment**:
The extracted frames are passed through a face detection and alignment pipeline using **`resize_and_align_faces.py`**.

- **Face detection** is performed using MTCNN or RetinaFace to locate bounding boxes of faces in the frames.
- **Face alignment** is applied using facial landmarks (eyes, nose, mouth corners) to normalize orientation and scale.

- The detected and aligned faces are cropped and resized to a fixed resolution (typically 224×224 pixels), ensuring consistent input dimensions across the dataset.
- Aligned face images are saved in the output directory, maintaining the original video-label association.

Frames/images are extracted separately for each submodule of a dataset (e.g., DF, F2F, NT, real in FaceForensics++).

About Data and Preprocessing.

- **FF25** → Uses **25%** of the total data for training.
- **FF50** → Uses **50%** of the total data.
- **FF75** → Uses **75%** of the dataset.
- **FF100** → full dataset

For FaceForensic++ Dataset

Class	Videos	Images
FS	999	31924
real	999	31949
DF	999	31843
NT	999	31931
F2F	999	31949

Note - ~1000 videos each , label : real - 0 , fake - 1 (FS,DF,NT,F2F)
Below split consists of images for FF100,FF75,FF50,FF25 respectively.
Split used is 70:15:15 (train:val:test)

FF100 Dataset Splits

Subst	Train	Validation	Test
NT	22,354	4,748	4,829
F2F	22,353	4,765	4,831
DF	22,269	4,750	4,824
FS	22,337	4,761	4,826
Real	22,360	4,761	4,828

FF75 Dataset Splits

Subst	Train	Validation	Test
NT	16,755	3,583	3,601
F2F	16,756	3,582	3,615
DF	16,699	3,555	3,614
FS	16,743	3,578	3,616
Real	16,761	3,583	3,615

FF50 Dataset Splits

Subst	Train	Validation	Test
NT	11,160	2,367	2,432
F2F	11,160	2,368	2,432
DF	11,100	2,367	2,432
FS	11,163	2,353	2,431
Real	11,163	2,368	2,430

FF25 Dataset Splits

Subst	Train	Validation	Test
NT	5,566	1,184	1,214
F2F	5,566	1,184	1,215
DF	5,521	1,182	1,214
FS	5,568	1,184	1,216
Real	5,567	1,184	1,216

Dataset Splits:

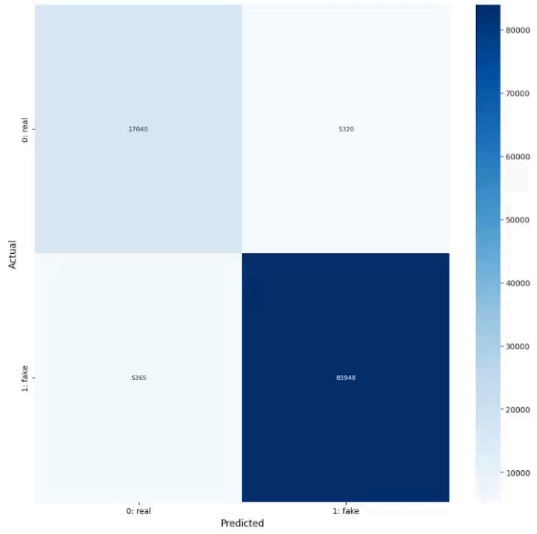
- **FF++ 100% : original FaceForensics++ dataset**
- **FF++ 75% : 75% of original FaceForensics++ dataset**
- **FF++ 50% : 50% original FaceForensics++ dataset**
- **FF++ 25% :25% original FaceForensics++ dataset**

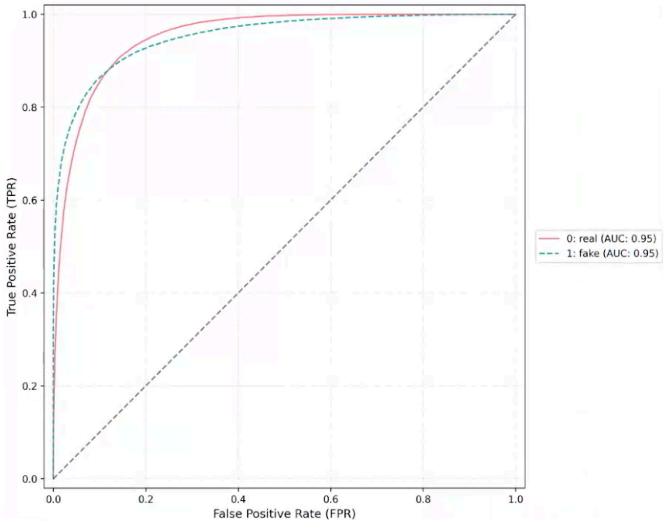
Experiment Setup 1 — The model was trained using the FaceForensics++ (FF++) dataset. (FF100)

Hyperparameters Used -

- Backbone: `openai/clip-vit-large-patch14` (frozen feature extractor)
- Batch size: 128
- Epochs: 10
- Learning rate: 8e-5
- Weight decay: 0.0
- Loss: Cross-Entropy
- PEFT: Enabled with LN-tuning only (LoRA disabled)
- Head: Linear layer with normalization (LinearNorm)
- Precision: Mixed bf16
- Feature augmentation: SLERP enabled
- Number of classes: 2 (binary classification)
- Random seed: 42

Results on Training Data



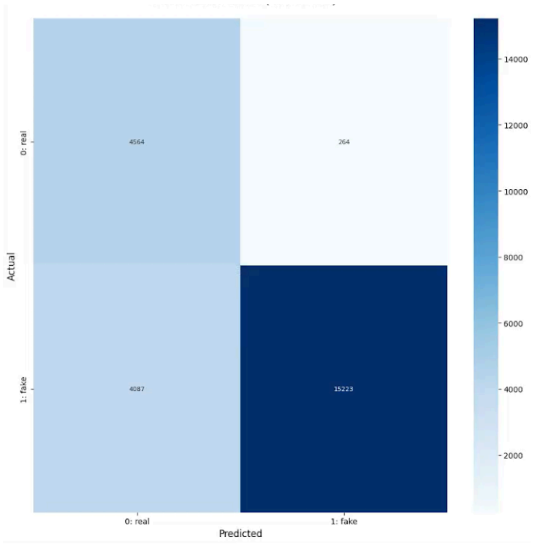


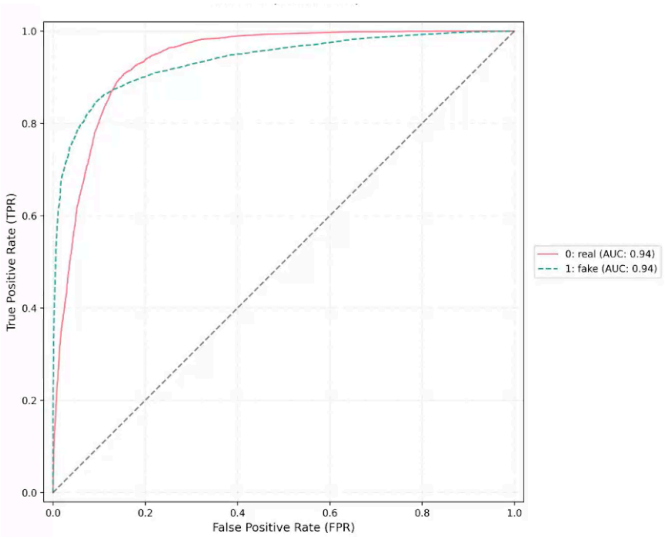
Metric	Value
Accuracy	89.38%
Precision	94.04%
Recall	93.99%
F1 Score	94.01%

AUC-ROC

0 real (auc = 0.95)
1 fake (auc = 0.95)

Result on Testing Data





Metric	Value
Accuracy	82.03%
Precision	98.29%
Recall	78.82%
F1 Score	87.54%

AUC-ROC

0 real (auc = 0.94)
1 fake (auc = 0.94)

Justification

Training Results

- Accuracy: 89.38% — The model performs well overall on training data, correctly classifying most samples.
- Precision: 94.04% — High precision means when the model predicts a deepfake, it's very likely correct, minimizing false positives.
- Recall: 93.99% — High recall shows the model detects most of the actual deepfakes, missing very few.
- F1 Score: 94.01% — Balanced precision and recall, indicating the model is both accurate and reliable.
- AUC-ROC (~0.95 for both classes) — Excellent discrimination ability, showing the model separates real and fake classes very well on training.

Testing Results

- Accuracy: 82.03% — Some drop compared to training, expected due to unseen data, but still good overall performance.
- Precision: 98.29% — Very high precision, the model almost never falsely labels a real image as fake
- Recall: 78.82% — Lower recall than training suggests the model misses some deepfakes on test data, meaning some fakes slip through.
- F1 Score: 87.54% — Still a strong balance, but slightly reduced compared to training, indicating some trade-off between precision and recall on new data.

AUC-ROC (0.94 for both classes) — Maintains excellent class separation on test data, confirming robustness

Note : Train Balanced accuracy has also been used

Train Balanced Accuracy is a metric used during training that accounts for class imbalance by calculating the average of recall (true positive rate) obtained on each class.

Experiments

1) LN TUNING + Norm + Slerp

Definition

This method builds on **LN Tuning + Norm** by introducing **latent space augmentation** using **Spherical Linear Interpolation (Slerp)**. It augments training data by interpolating feature embeddings between samples of the same class, promoting better generalization and robustness to unseen forgeries

For each feature vector `xi` in a batch:

- Randomly choose another vector `xj` from the **same class**
- Sample `t ~ U(0, 1)`
- Compute:

$$\text{slerp}(x_i, x_j, t) = \frac{\sin((1-t)\theta)}{\sin(\theta)} x_i + \frac{\sin(t\theta)}{\sin(\theta)} x_j$$

  where:

$$\theta = \arccos(x_i^\top x_j)$$

- Use the augmented vector as an additional training example.

Trainable Parameters

- Same as LN tuning + norm (~104K trainable out of ~303M)

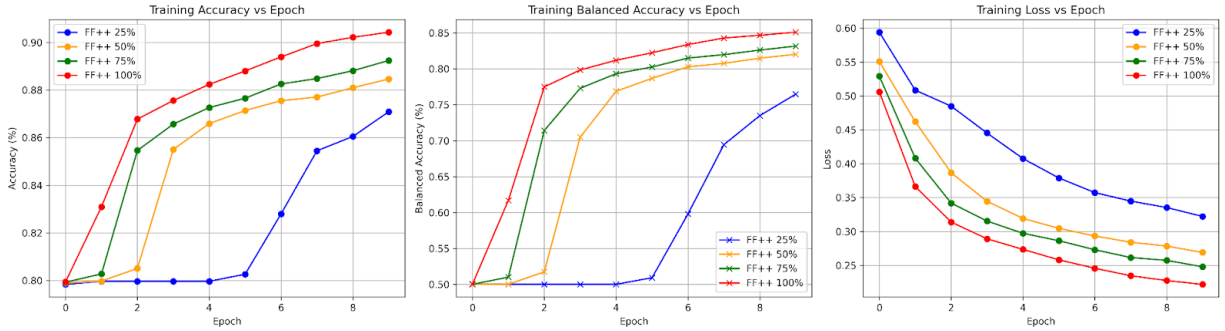
Hyperparameters Used

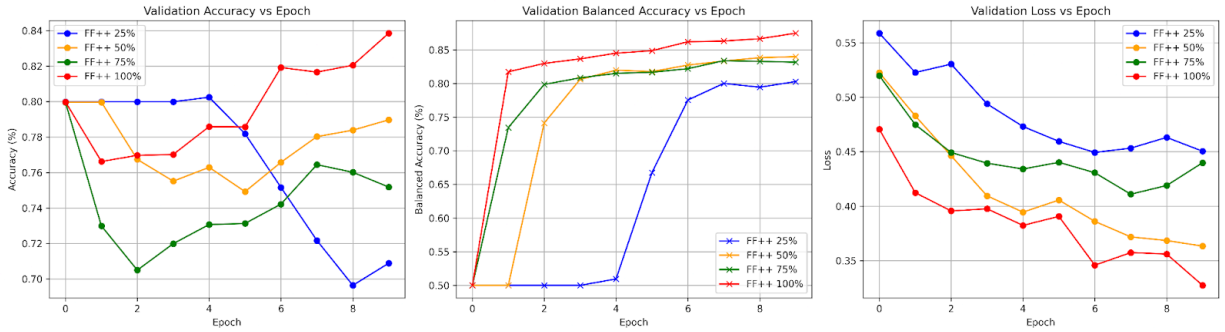
- Feature extractor frozen: ✅ (`freeze_feature_extractor = True`)
- PEFT / LN-tuning: (`peft.enabled = True`)
- Classification head: `LinearNorm`
- Feature augmentation (SLERP): (`slerp_feature_augmentation = True`)

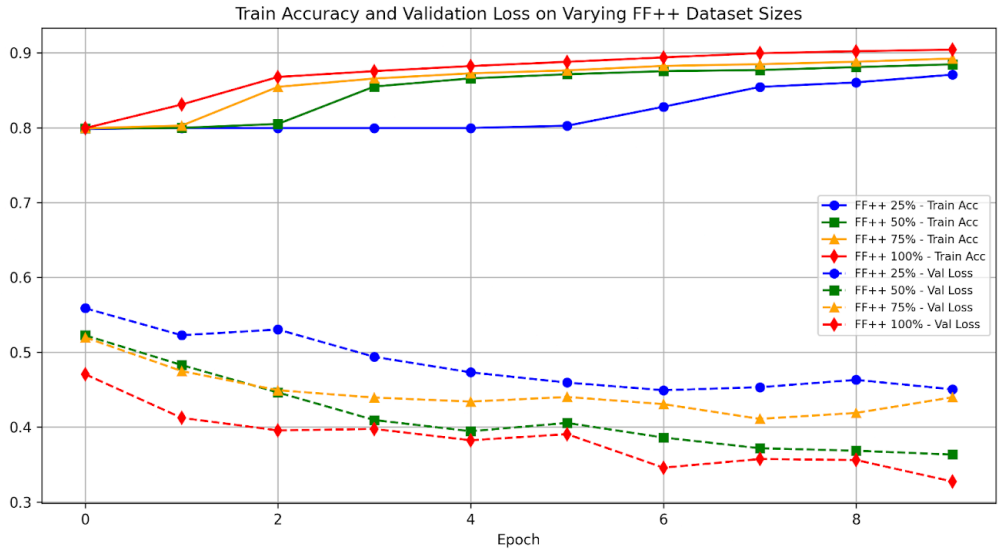
Training Configuration

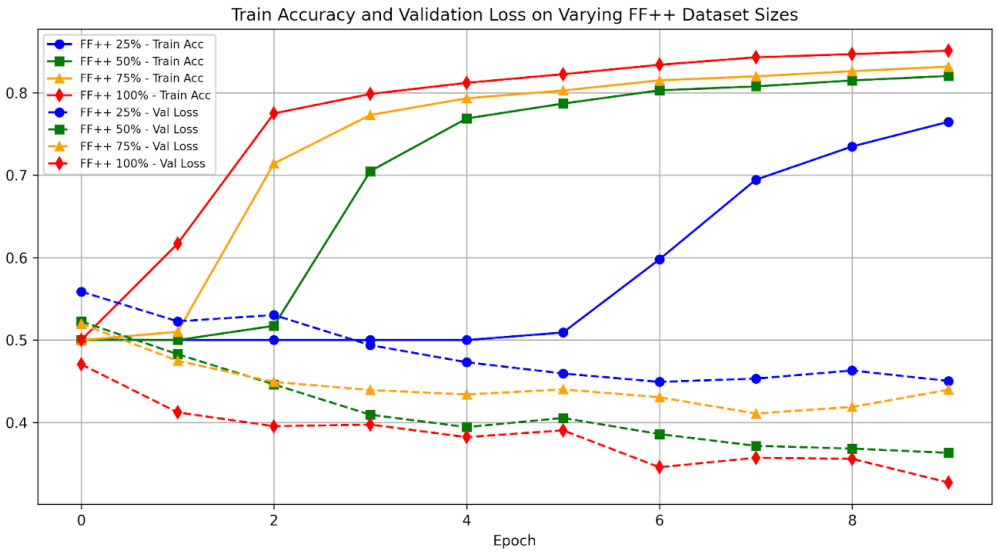
- **Batch size:** `128`
- **Learning rate:** `8e-5`
- **Minimum learning rate:** `5e-5`
- **Weight decay:** `0`
- **Learning rate scheduler:** `cosine`
- **Epochs:** `10`

Results on Training & Validation Data



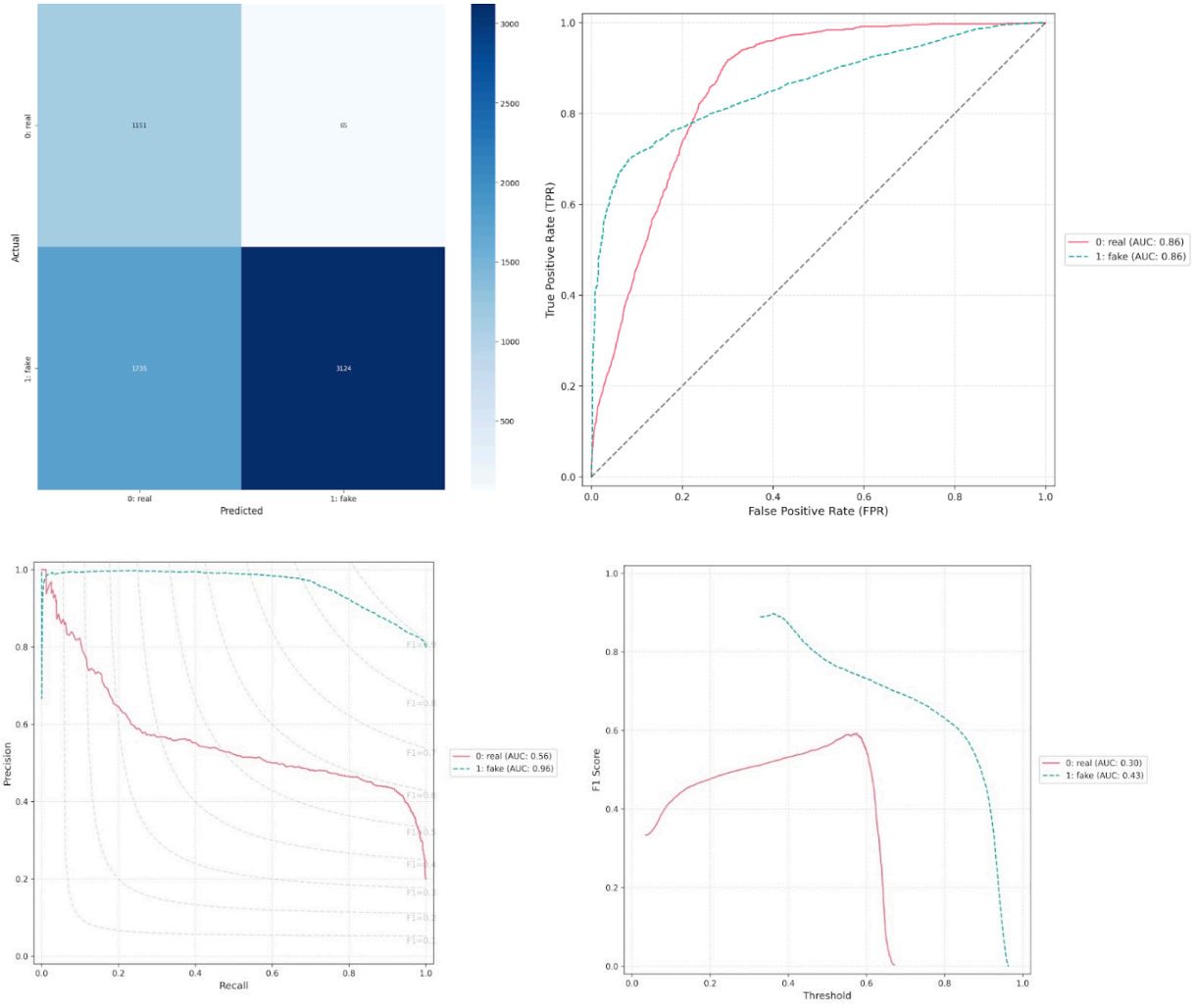




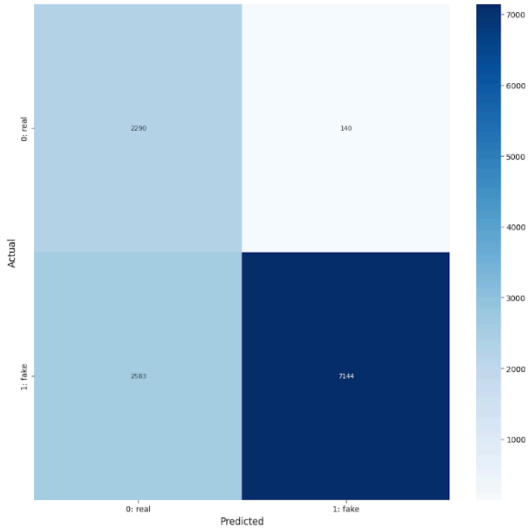


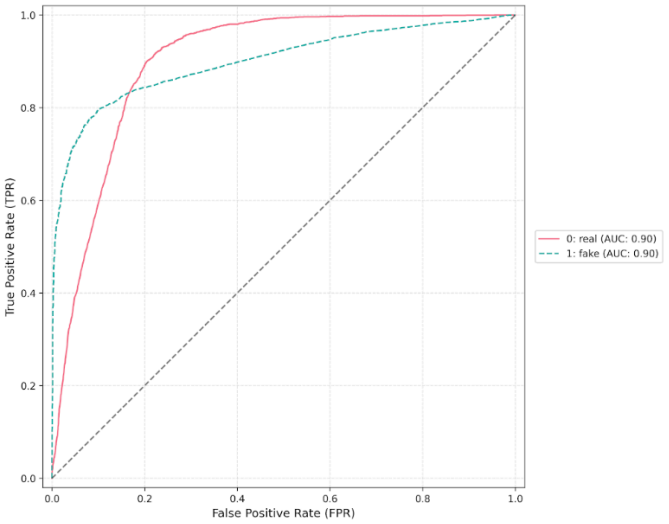
Results on Testing Data

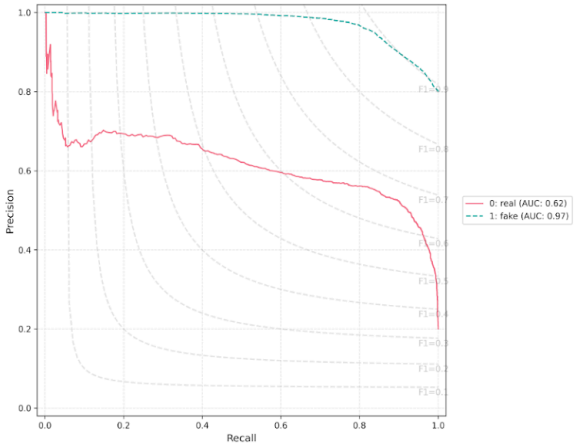
FF25

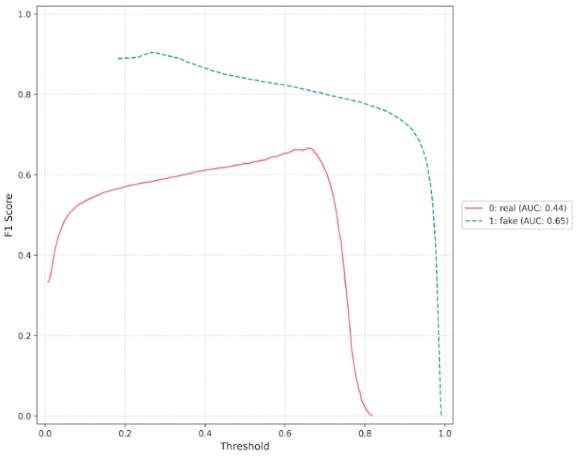


FF50

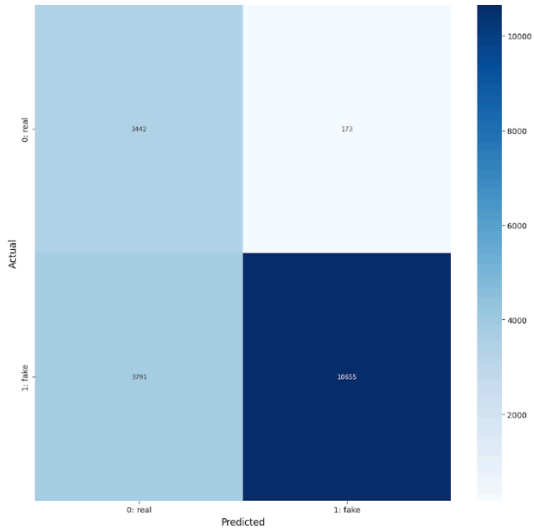

Actual
0: real
1: fake
2290
149
2583
7144
0: real
1: fake
Predicted
7000
6000
5000
4000
3000
2000
1000

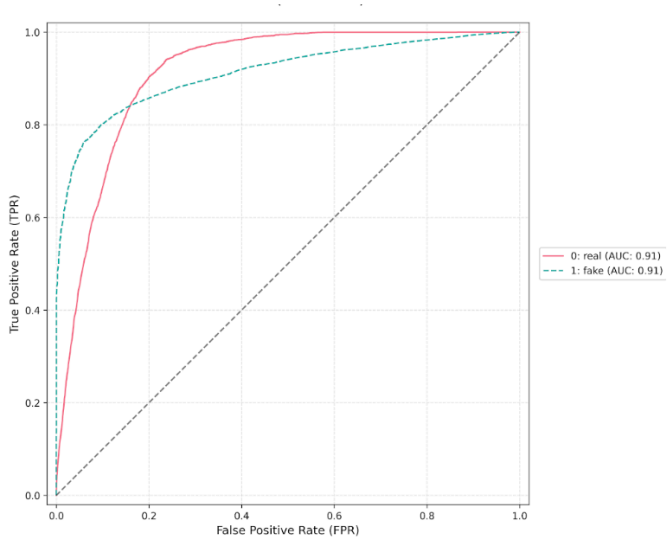

True Positive Rate (TPR)
False Positive Rate (FPR)
0: real (AUC: 0.90)
1: fake (AUC: 0.90)

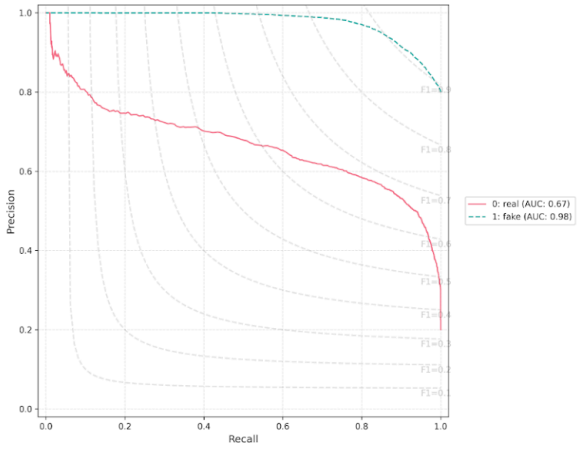

Precision
Recall
0: real (AUC: 0.62)
1: fake (AUC: 0.97)

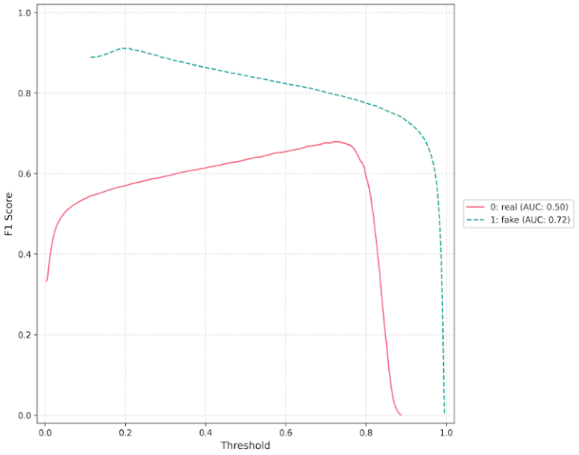

F1 Score
Threshold
0: real (AUC: 0.44)
1: fake (AUC: 0.65)

FF75

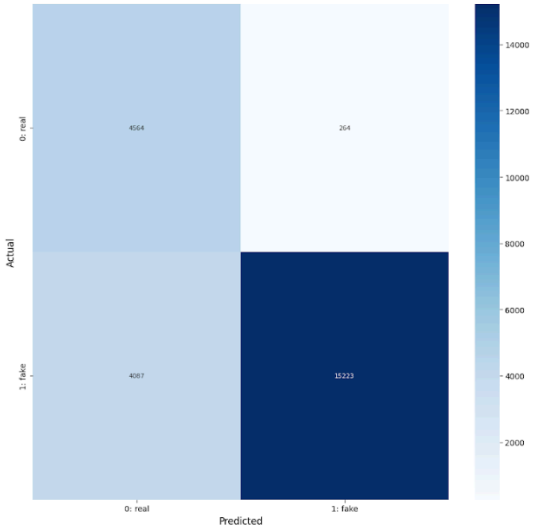

Actual
0: real
1: fake
3442
173
3791
10655
0: real
1: fake
Predicted
10000
8000
6000
4000
2000

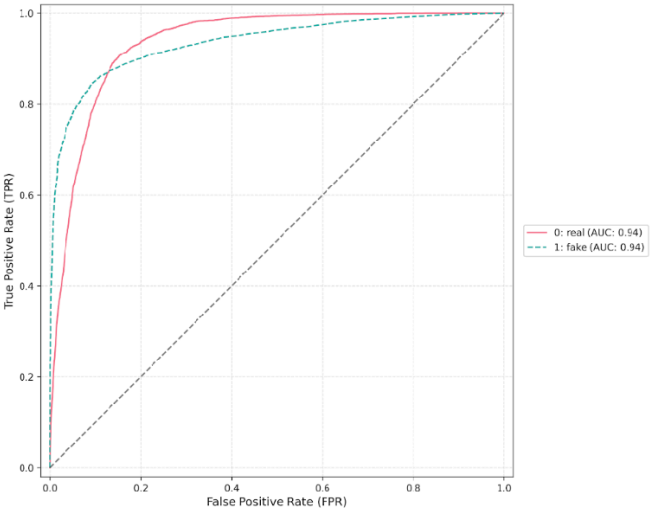

True Positive Rate (TPR)
False Positive Rate (FPR)
0: real (AUC: 0.91)
1: fake (AUC: 0.91)

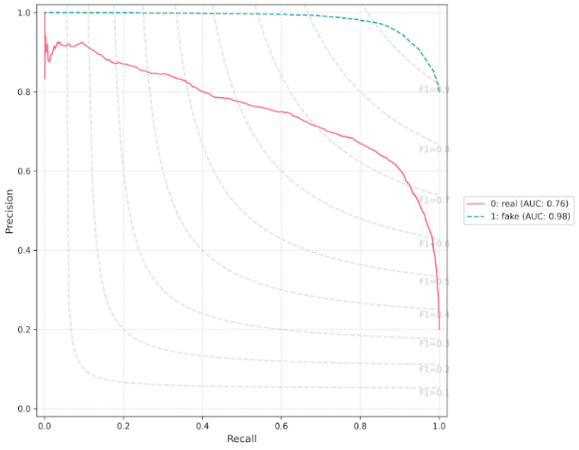


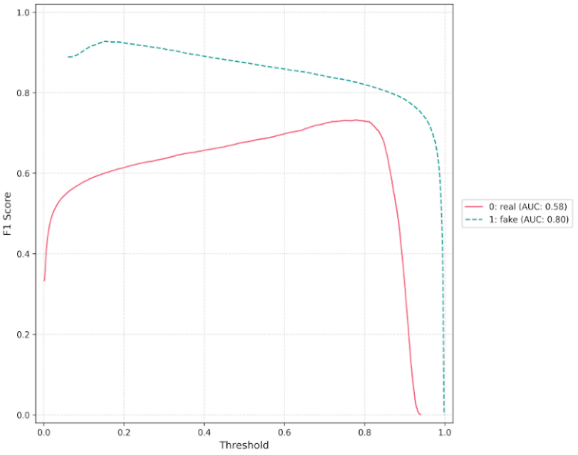


FF++









Overall Stats [class fake(1)]

Dataset	Precision	Recall	F1 score	Accuracy

FF25	**0.98**	**0.64**	**0.77**	**0.70**
FF50	**0.98**	**0.73**	**0.83**	**0.77**
FF75	**0.98**	**0.74**	**0.84**	**0.78**
FF++	**0.98**	**0.79**	**0.88**	**0.82**

Justification

- With a very limited dataset FF 25%, the model struggles to converge quickly or learn meaningful patterns efficiently
- A larger and more diverse training dataset provides the model with more examples to learn, leading to higher performance on the training set itself.
- With only FF 25% of the data, the model likely struggles with **underfitting** (as indicated by the slower training accuracy gain)

In observation, the results strongly justify that utilizing a larger proportion of the FF++ dataset significantly improves the model's ability to learn and generalize, resulting in higher balanced accuracy on the training set and critically, lower loss on the unseen validation set.

- More Data = Better Generalization:
- Optimal Learning with Sufficient Data**:** The curves for higher data percentages (75%, 100%) show a more rapid improvement in both training accuracy and a steeper, more consistent decrease in validation loss
- The FF 25% case eventually reaches good training accuracy but still has high validation loss .i.e relying solely on training accuracy can be misleading,

2) Linear Probing (Baseline)

- Definition

  Linear probing is a technique to evaluate the **quality of pretrained representations** (like those from CLIP) by **freezing the backbone (e.g., ViT in CLIP)** and training **only a shallow linear classifier** (e.g., a single-layer MLP, `nn.Linear`) on top of it.

- **Goal**

  Test whether the pretrained features are **linearly separable** for the downstream classification task (e.g., real vs. fake).

Methodology

- Use a pretrained model like `CLIP-ViT-L/14` as the **feature extractor**.
- **Freeze all layers** of the backbone.
- **Extract the classification token** (`[CLS]`) from ViT.
- Pass `[CLS]` through a **linear layer** to predict class probabilities.
- Train only this linear head using cross-entropy loss.

Pseudocode

1. Initialize CLIP ViT-L/14 backbone with pretrained weights

2. Freeze all parameters of the CLIP backbone

3. Initialize a linear classification head:

   - Input dimension: 1024 (CLIP's CLS token)

   - Output dimension: 2 (binary classification)

4. For each training step:

   a. Load a batch of images and corresponding labels

   b. Pass images through the CLIP backbone to extract CLS token features (size: 1024)

   c. Pass features through the linear head to get logits (size: 2)

   d. Compute cross-entropy loss between logits and labels

   e. Backpropagate loss and update only the linear head's parameters

```
# Step 1: Initialize CLIP ViT-L/14 backbone
clip_model = load_pretrained_clip_model(variant="ViT-L/14")
image_encoder = clip_model.visual_encoder  # Extract visual part only
# Step 2: Freeze all parameters of the encoder
for param in image_encoder.parameters():
    param.requires_grad = False
# Step 3: Initialize linear classification head
linear_head = nn.Linear(in_features=1024, out_features=2)  # Binary classification
# Step 4: Training loop
for images, labels in dataloader:
    # a. Forward pass through frozen CLIP encoder
    with torch.no_grad():
        z_cls = image_encoder(images)  # Output shape: [batch_size, 1024]
    # b. Forward pass through trainable linear head
    logits = linear_head(z_cls)  # Output shape: [batch_size, 2]
    # c. Compute loss
    loss = cross_entropy_loss(logits, labels)
    # d. Backpropagation and update linear head only
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Only the final linear head parameters are updated.

- **Total parameters**: ~303M
- **Trainable**: 2,050
- **Trainable %**: ~0.0007%

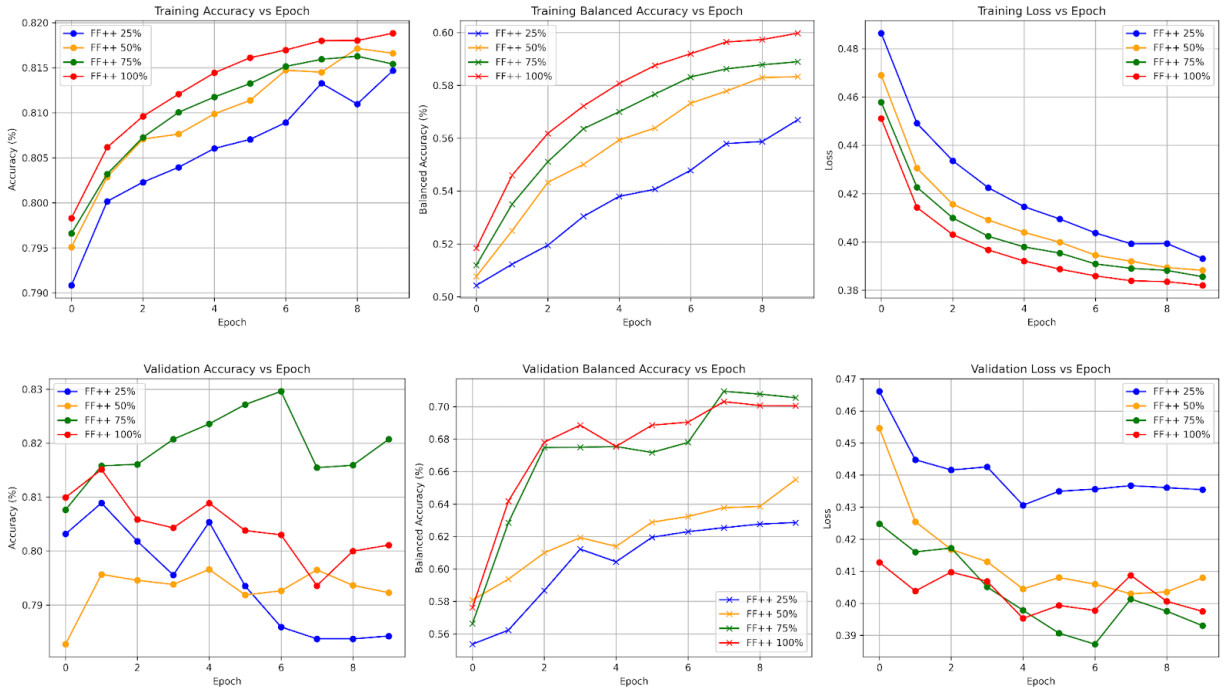
Hyperparameters Used

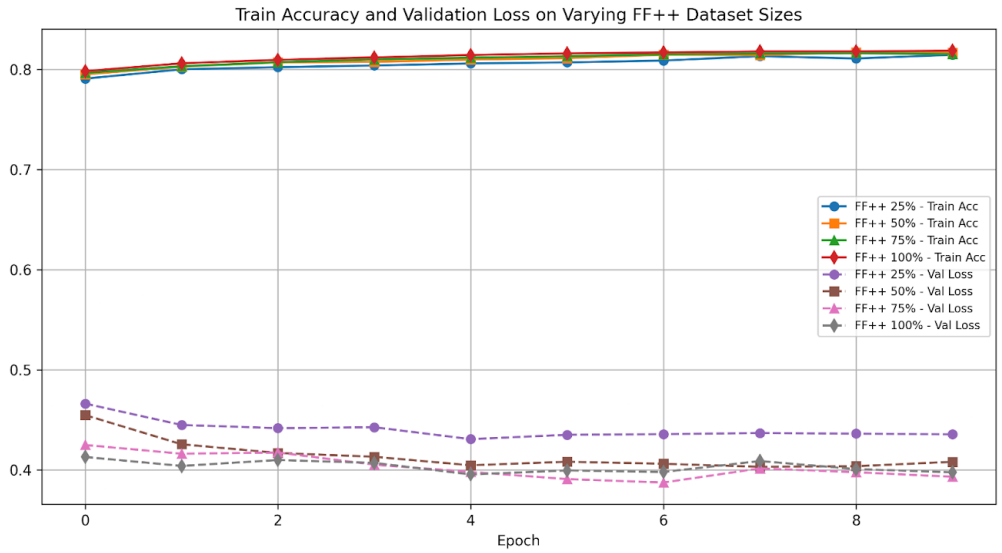
- Feature extractor frozen: ✅ (`freeze_feature_extractor = True`)
- PEFT / LN-tuning: ❌ (`peft.enabled = False`)
- Classification head: `Linear` (no normalization, direct `Linear(1024 → 2)`)
- Feature augmentation (SLERP): ❌ (`slerp_feature_augmentation = False`)

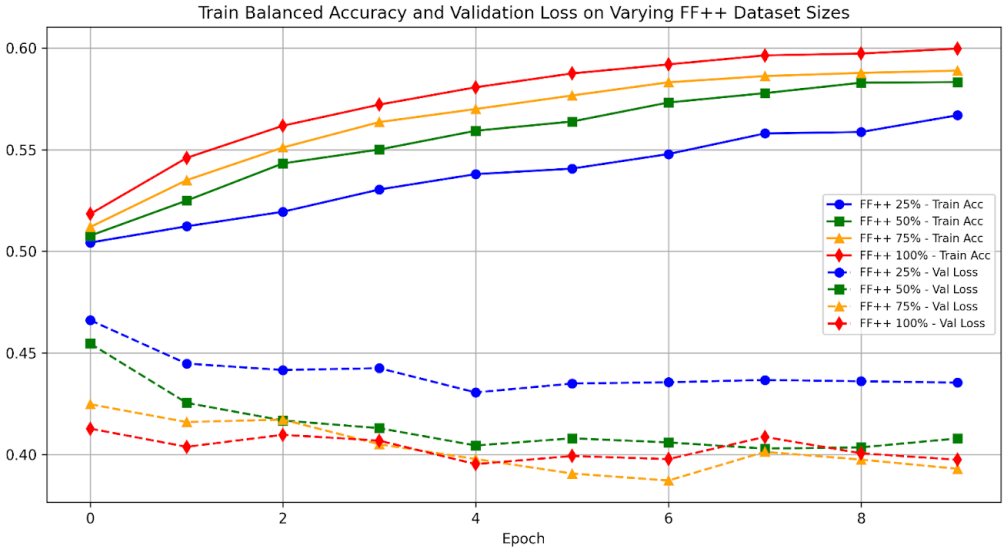
Training Configuration

- **Batch size:** `128`
- **Learning rate:** `8e-5`
- **Minimum learning rate:** `5e-5`
- **Weight decay:** `0`
- **Learning rate scheduler:** `cosine`
  Epochs: `10`

Results on Training & Validation Data

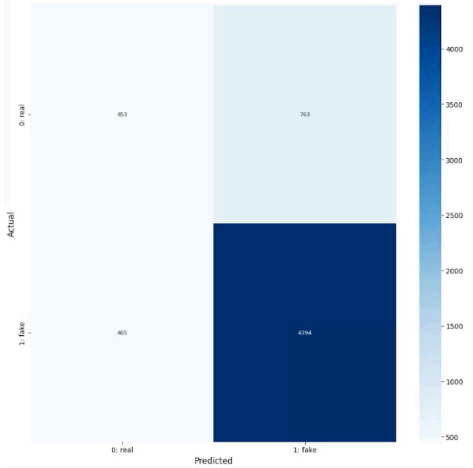


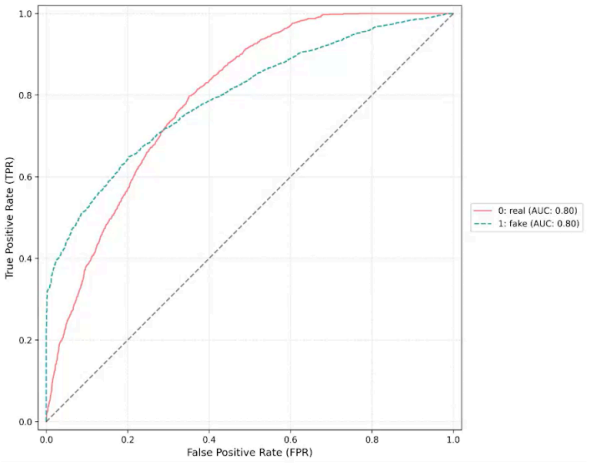


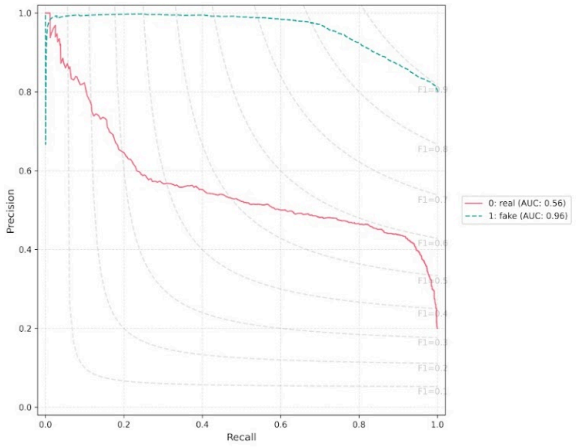


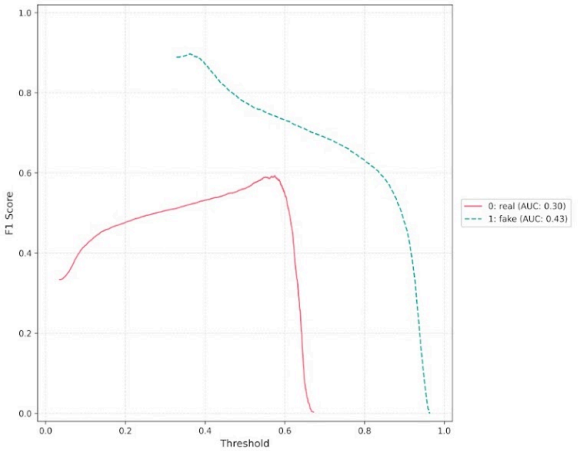
Result on Testing Data

FF25

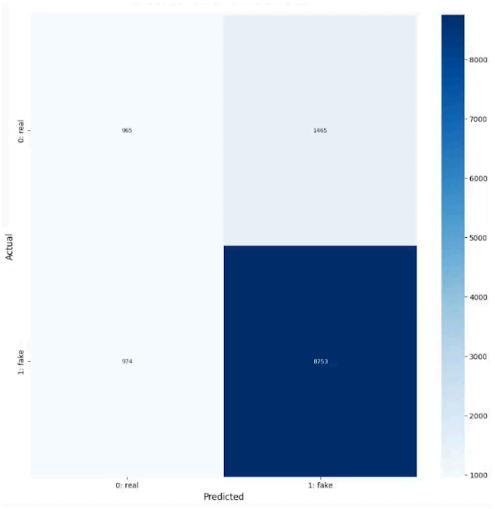


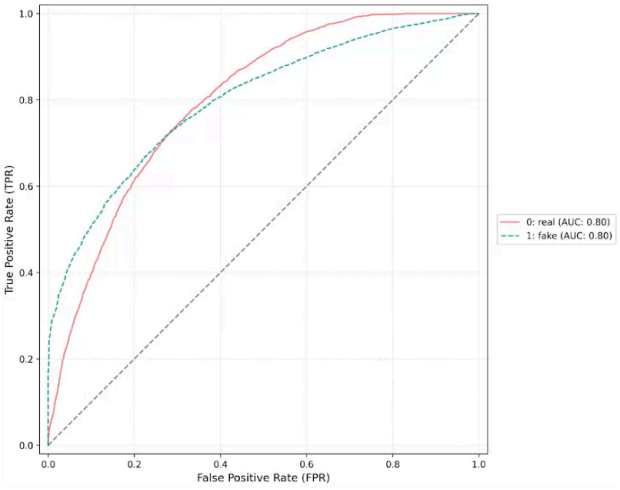


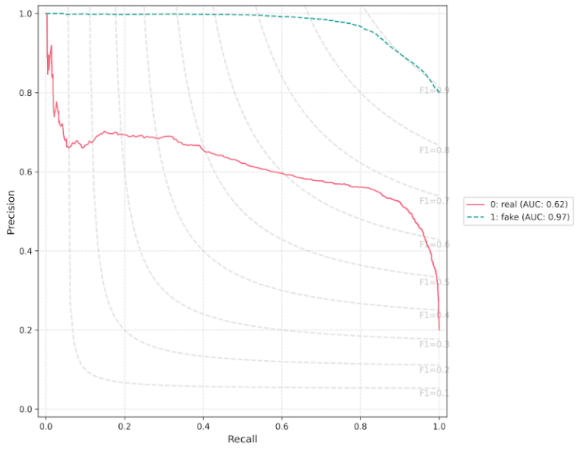

Precision
Recall
0: real (AUC: 0.56)
1: fake (AUC: 0.96)

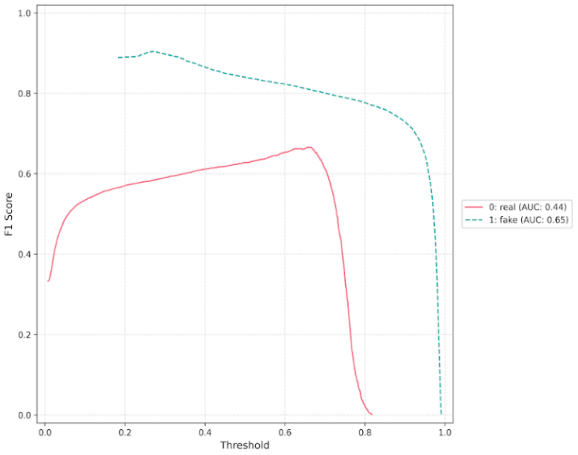

F1 Score
Threshold
0: real (AUC: 0.30)
1: fake (AUC: 0.43)

FF50

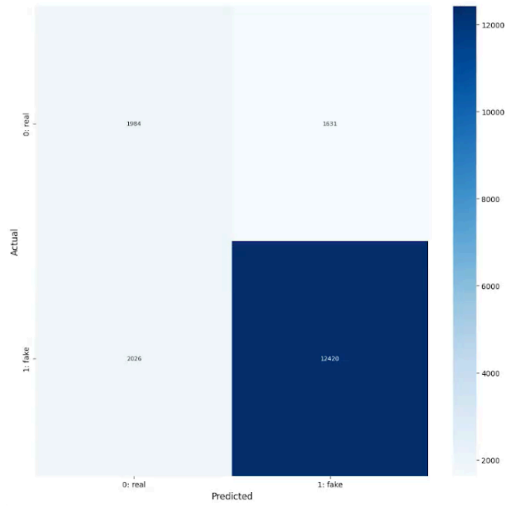

Actual
Predicted
0: real
1: fake
965
1465
974
8753

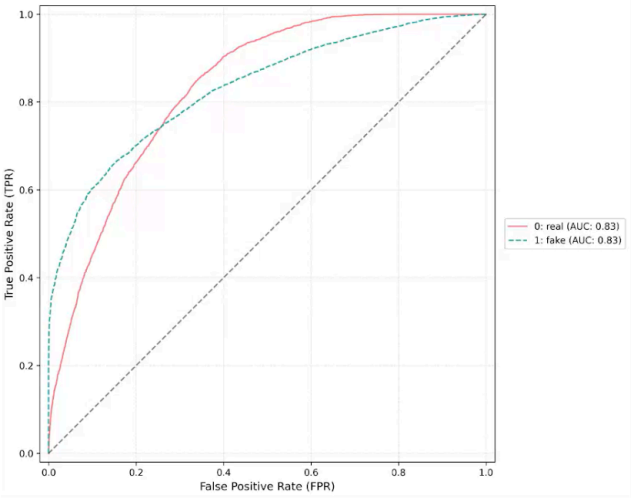

True Positive Rate (TPR)
False Positive Rate (FPR)
0: real (AUC: 0.80)
1: fake (AUC: 0.80)

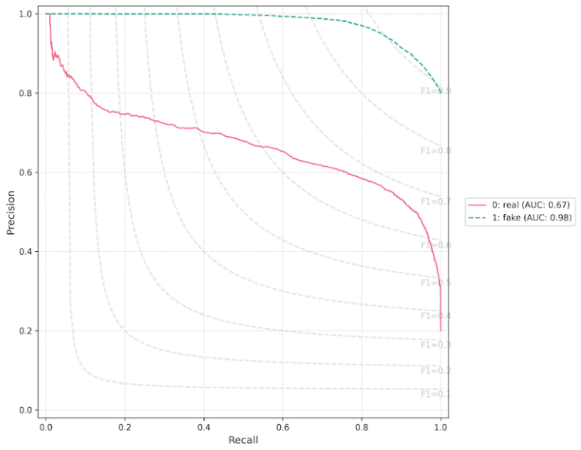

Precision
Recall
0: real (AUC: 0.62)
1: fake (AUC: 0.97)

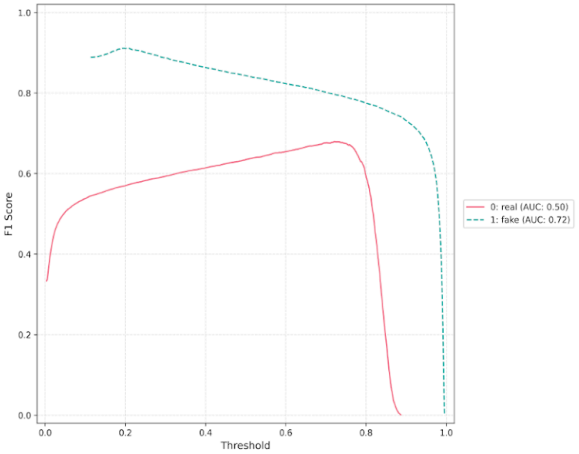

F1 Score
Threshold
0: real (AUC: 0.44)
1: fake (AUC: 0.65)

FF75

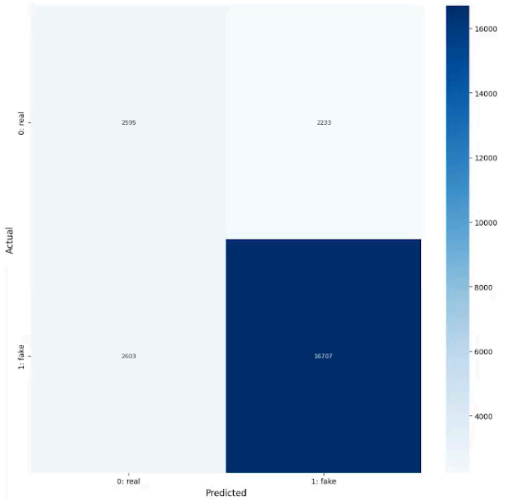

1984
1631
2026
12420
0: real
1: fake
Actual
Predicted
12000
10000
8000
6000
4000
2000

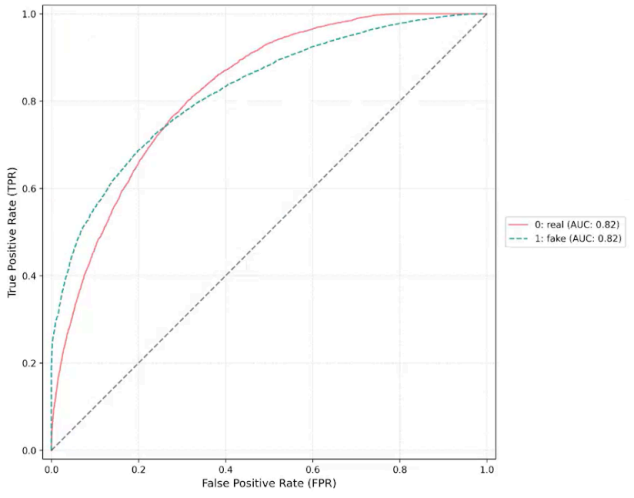

True Positive Rate (TPR)
False Positive Rate (FPR)
0: real (AUC: 0.83)
1: fake (AUC: 0.83)

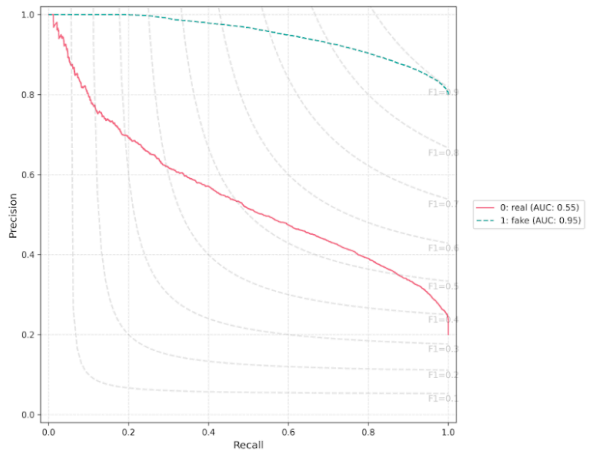

Precision
Recall
0: real (AUC: 0.67)
1: fake (AUC: 0.98)

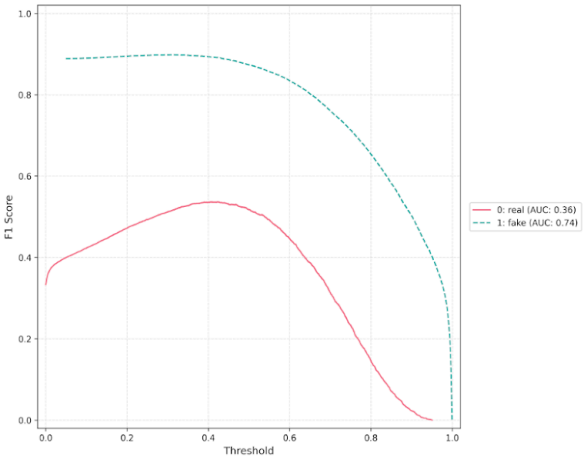

F1 Score
Threshold
0: real (AUC: 0.50)
1: fake (AUC: 0.72)

FF++


2595
2233
2603
16707
0: real
1: fake
Actual
Predicted
16000
14000
12000
10000
8000
6000
4000


True Positive Rate (TPR)
False Positive Rate (FPR)
0: real (AUC: 0.82)
1: fake (AUC: 0.82)





Overall stats [class fake(1)]

Dataset	Precision	Recall	F1 score	Accuracy
FF25	**0.85**	**0.90**	**0.87**	**0.79**
FF50	**0.85**	**0.89**	**0.88**	**0.80**
FF75	**0.88**	**0.86**	**0.87**	**0.80**
FF++	**0.88**	**0.87**	**0.87**	**0.80**

Justification

- **More Data = Better Training Performance:** The larger the dataset, the higher the balanced accuracy achieved on the training set. This is expected, as more data provides more examples for the model to learn from, leading to a better fit for the training distribution.

- **Model Capacity:** The model appears to have sufficient capacity to learn from larger datasets, as it continues to improve with more data.

- **No Significant Underfitting on Training:** None of the training accuracy curves are flat or excessively low, suggesting the model is generally capable of learning from the data, even with smaller subsets.

- The 25% dataset clearly struggles with generalization, exhibiting higher and more validation loss. This indicates that 25% of the FF++ dataset is likely insufficient for the model to learn robust, generalizable features.

3) LN TUNING ONLY

- Definition

  LayerNorm (LN) Tuning involves unfreezing only the **LayerNorm parameters** in the frozen CLIP backbone and training them, in addition to the linear classifier. It enables minimal fine-tuning while preserving most of the CLIP model's knowledge.

- Goal

  Improve performance over plain linear probing while keeping the number of trainable parameters very low (~0.034%).

Methodology

- Train:
  The **linear head** (W, b)
  Only the **LayerNorm parameters** in the CLIP backbone
- Freeze:
  All other parts of the CLIP backbone
- Loss: Cross-Entropy Loss using predicted logits.

Hyperparameters Used

- Feature extractor frozen: ✅ (`freeze_feature_extractor = True`)
- PEFT / LN-tuning: (`peft.enabled =True`)
- Classification head: `Linear` (no normalization, direct `Linear(1024 → 2)`)
- Feature augmentation (SLERP): ❌ (`slerp_feature_augmentation = False`)

Training Configuration

- **Batch size:** `128`
- **Learning rate:** `8e-5`
- **Minimum learning rate:** `5e-5`
- **Weight decay:** `0`
- **Learning rate scheduler:** `cosine`
- **Epochs:** `10`

Trainable Parameters

- Total CLIP Parameters: ~303M
- Trainable: ~104K
- % of model: ~0.034%

Pseudocode

 # Step 1: Load CLIP backbone

- clip_model ← load_CLIP_ViT_L_14(pretrained=True)

 # Step 2: Freeze all parameters

- freeze_all_params(clip_model)

 # Step 3: Unfreeze LayerNorm layers

FOR each layer_name, param IN clip_model:

  IF "layernorm" IN layer_name.lower():

    param.requires_grad ← True

 # Step 4: Add Linear Classifier Head

- head ← LinearLayer(input_dim=1024, output_dim=2)

 # Step 5: Training Loop

FOR each batch (images, labels) IN dataloader:

  features ← clip_model.encode_image(images) # [CLS] token output

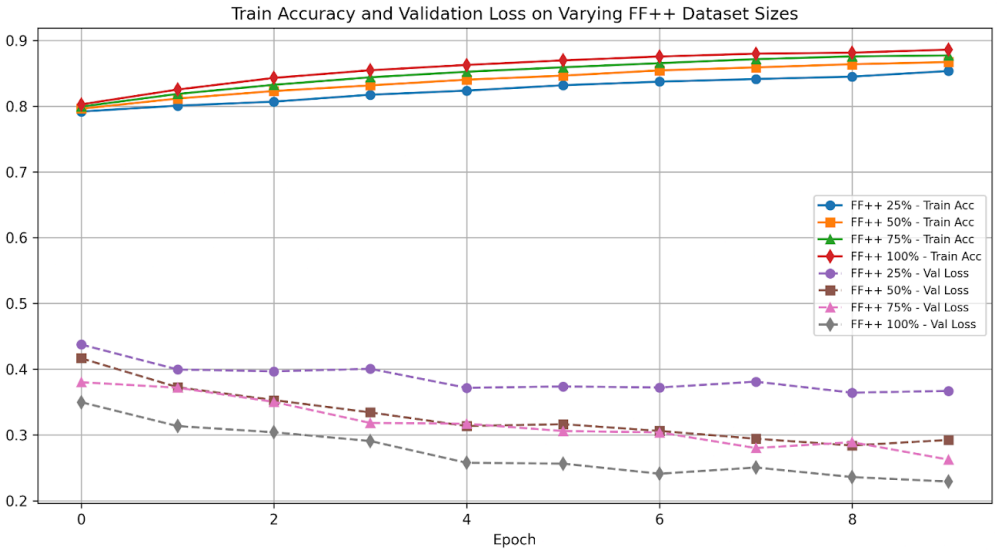
  logits ← head(features)

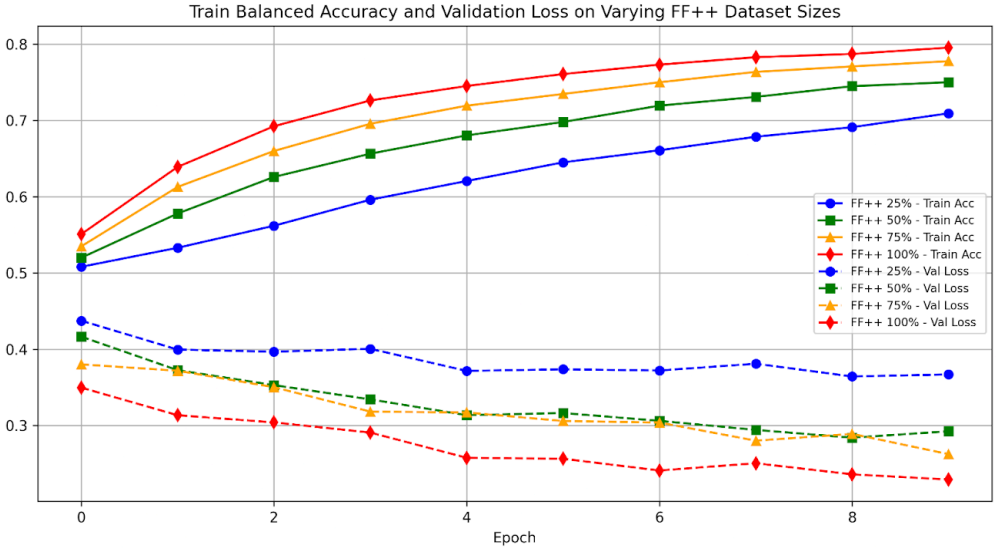
  loss ← CrossEntropyLoss(logits, labels)

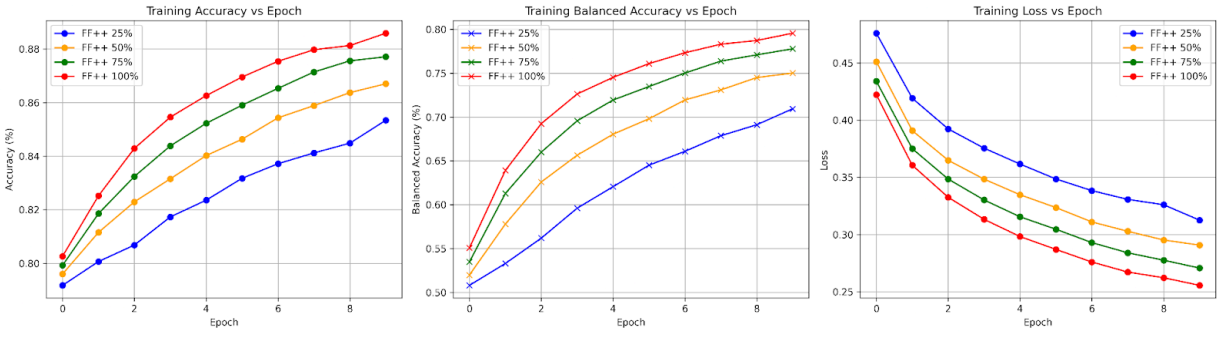
  loss.backward()

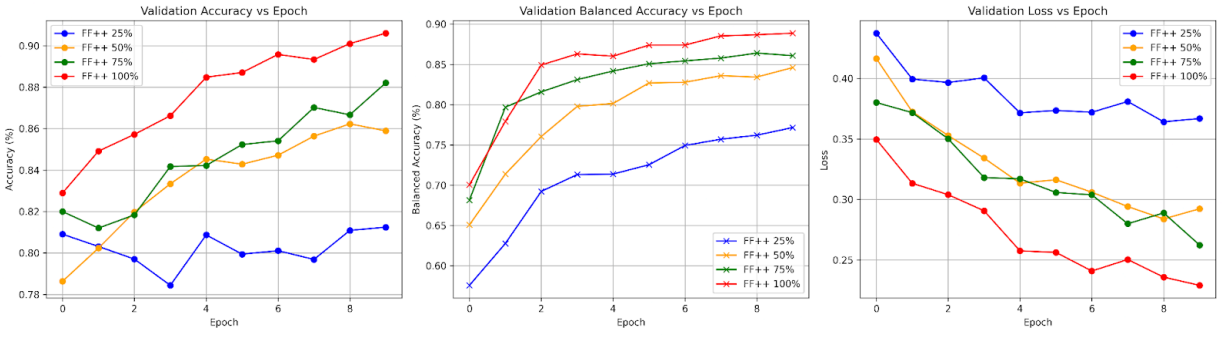
  optimizer.step()

Result on Training and Validation Data



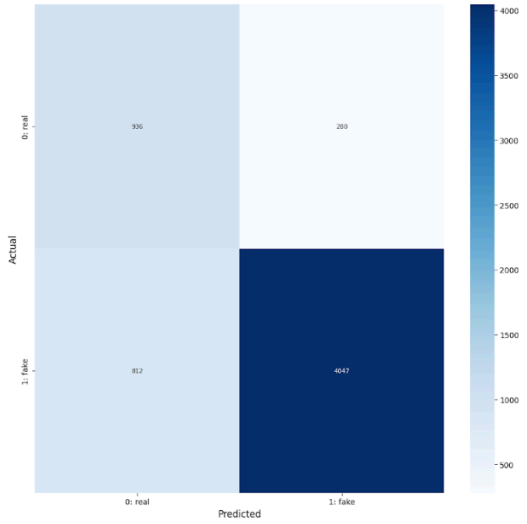


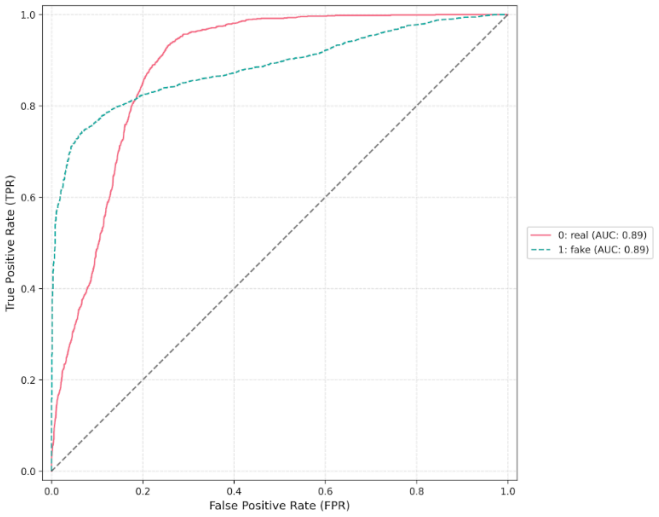


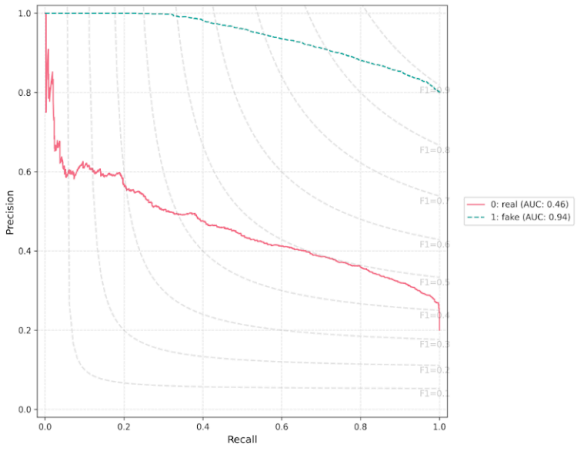


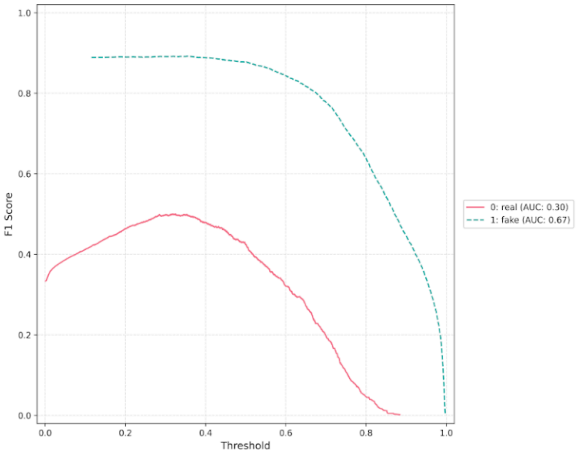
Result on Testing Data

FF25

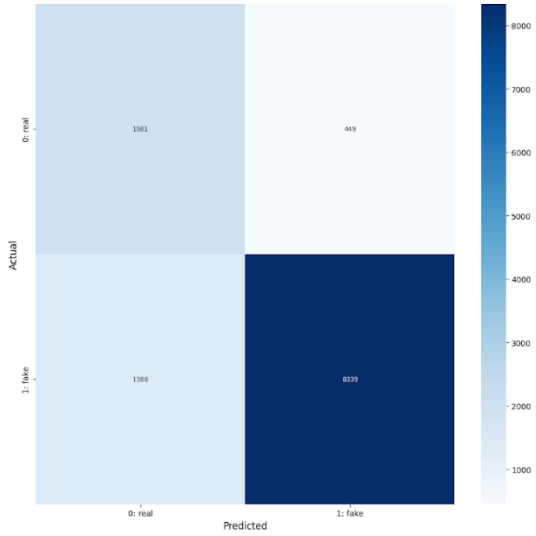


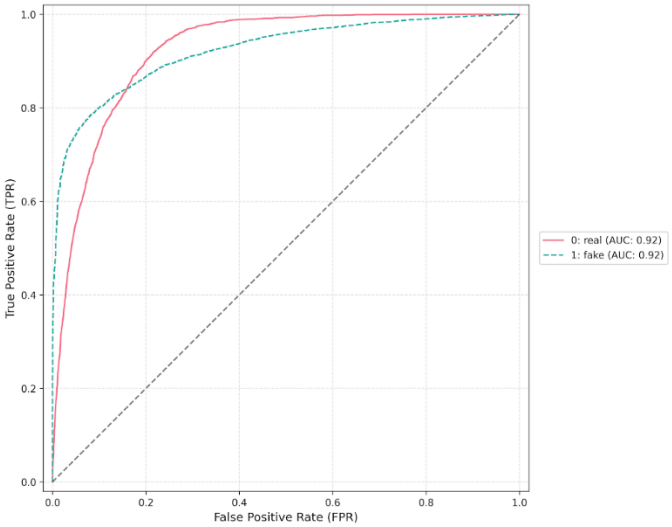


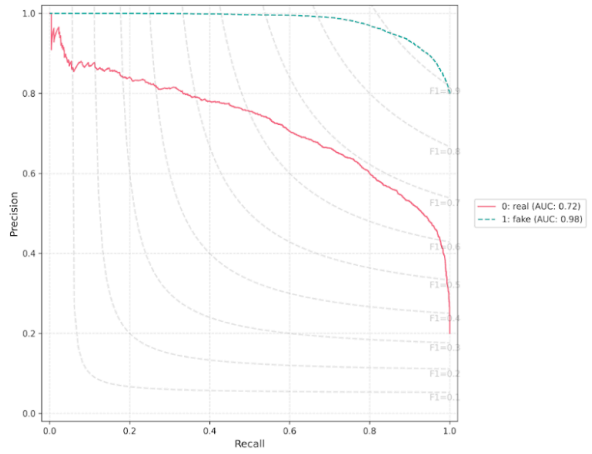


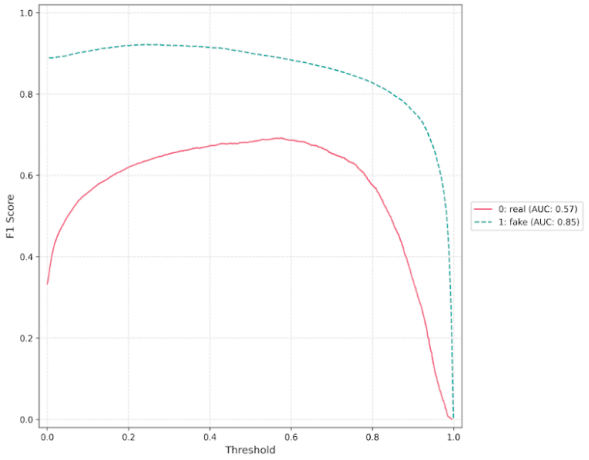


FF50

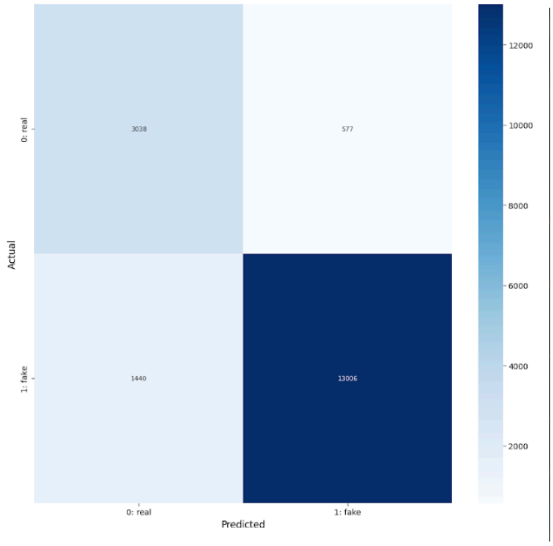


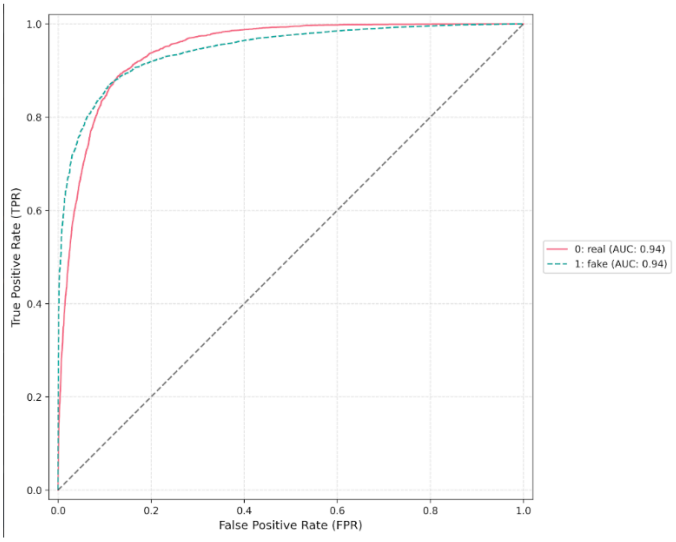


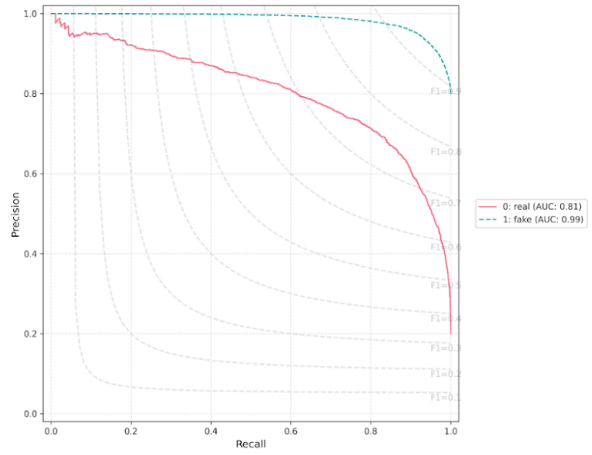

Precision
Recall
0: real (AUC: 0.72)
1: fake (AUC: 0.98)

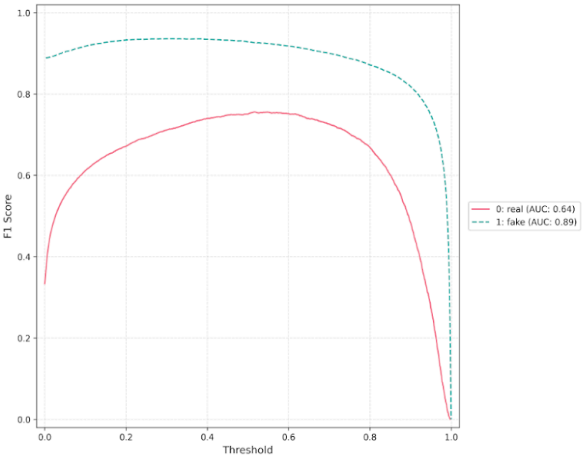

F1 Score
Threshold
0: real (AUC: 0.57)
1: fake (AUC: 0.85)

FF75

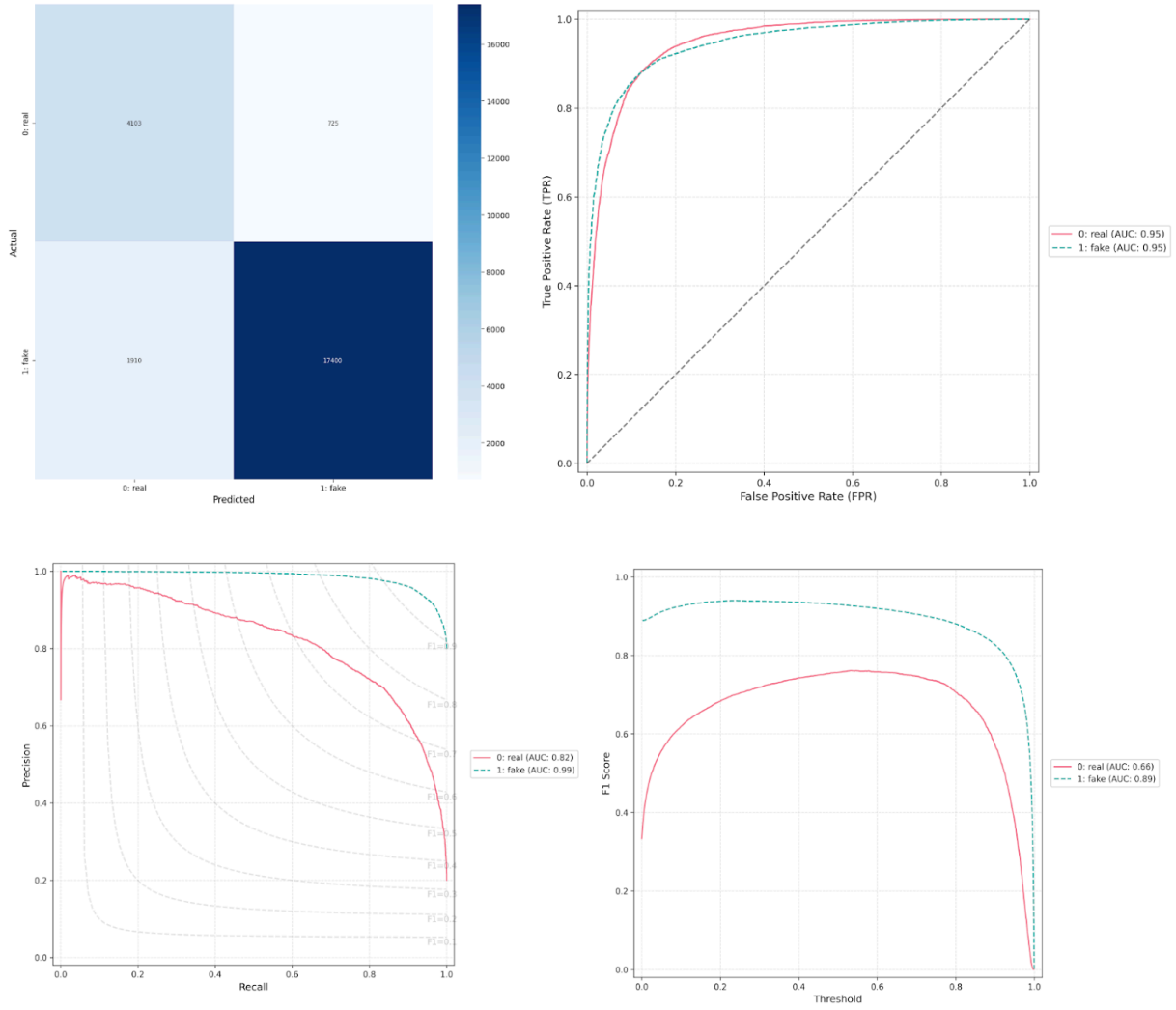

Actual
Predicted
0: real
1: fake
3038
577
1440
13006


True Positive Rate (TPR)
False Positive Rate (FPR)
0: real (AUC: 0.94)
1: fake (AUC: 0.94)


Precision
Recall
0: real (AUC: 0.81)
1: fake (AUC: 0.99)


F1 Score
Threshold
0: real (AUC: 0.64)
1: fake (AUC: 0.89)

FF++



Overall Stats [class fake(1)]

Dataset	Precision	Recall	F1 score	Accuracy
FF25	**0.93**	**0.83**	**0.88**	**0.82**
FF50	**0.94**	**0.85**	**0.90**	**0.84**
FF75	**0.95**	**0.90**	**0.92**	**0.88**
FF++	**0.96**	**0.90**	**0.93**	**0.89**

Justification

Good Training Convergence: For the larger datasets (100%, 75%, 50%), the validation loss curves show a good decrease and then stabilize, indicating that the training process is converging well without strong signs of overfitting within these 9 epochs. The models are learning to generalize effectively

4) LN Tuning + Norm

Definition

This method extends **LN Tuning** by **applying L2 normalization** to the CLIP [CLS] token features **before** passing them to the linear classifier. This constrains the features to lie on a unit hypersphere, improving class separability and robustness.

Training Objective

Train:

- **LayerNorm layers** in the CLIP backbone
- **Linear classifier head**

Apply **L2 normalization** on the [CLS] token
Loss: Cross-Entropy Loss between softmax(logits) and true labels

Hyperparameters Used

- Feature extractor frozen: ✅ (`freeze_feature_extractor = True`)
- PEFT / LN-tuning: (`peft.enabled =True`)
- Classification head: `LinearNorm`
- Feature augmentation (SLERP): ❌ (`slerp_feature_augmentation = False`)

Training Configuration

- **Batch size:** `128`
- **Learning rate:** `8e-5`
- **Minimum learning rate:** `5e-5`
- **Weight decay:** `0`
- **Learning rate scheduler:** `cosine`
- **Epochs:** `10`

Trainable Parameters

- Total CLIP Parameters: ~303M
- Trainable: ~104K
- % of model: ~0.034%

Pseudocode

 # Step 1: Load CLIP backbone

- clip_model ← load_CLIP_ViT_L_14(pretrained=True)

 # Step 2: Freeze all parameters

- freeze_all_params(clip_model)

 # Step 3: Unfreeze LayerNorm layers only

FOR each layer_name, param IN clip_model:

  IF "layernorm" IN layer_name.lower():

    param.requires_grad ← True

 # Step 4: Add Linear Classifier Head

- head ← LinearLayer(input_dim=1024, output_dim=2)

 # Step 5: Training Loop

FOR each batch (images, labels) IN dataloader:

  features ← clip_model.encode_image(images) # Extract CLS features

  norm_features ← L2Normalize(features) # Project onto unit sphere

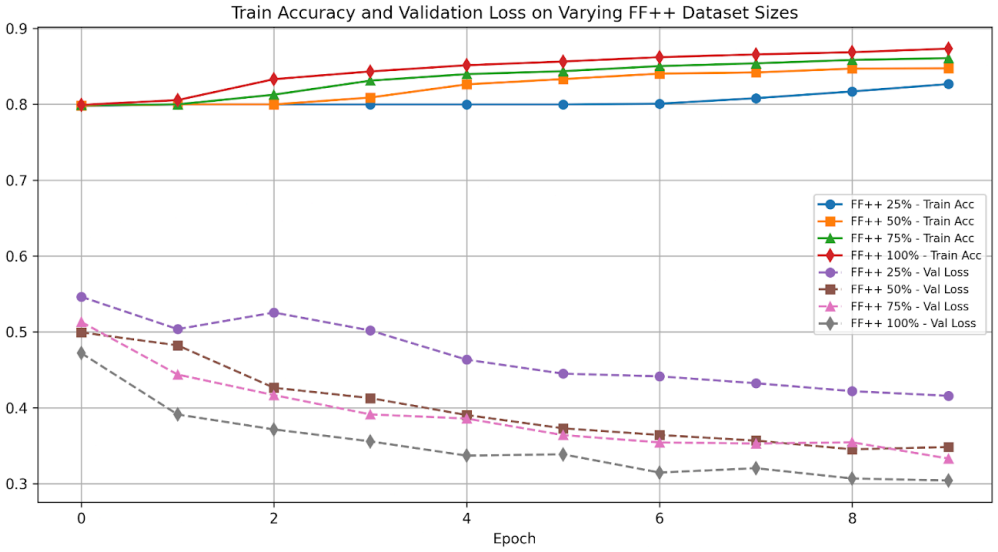
  logits ← head(norm_features)

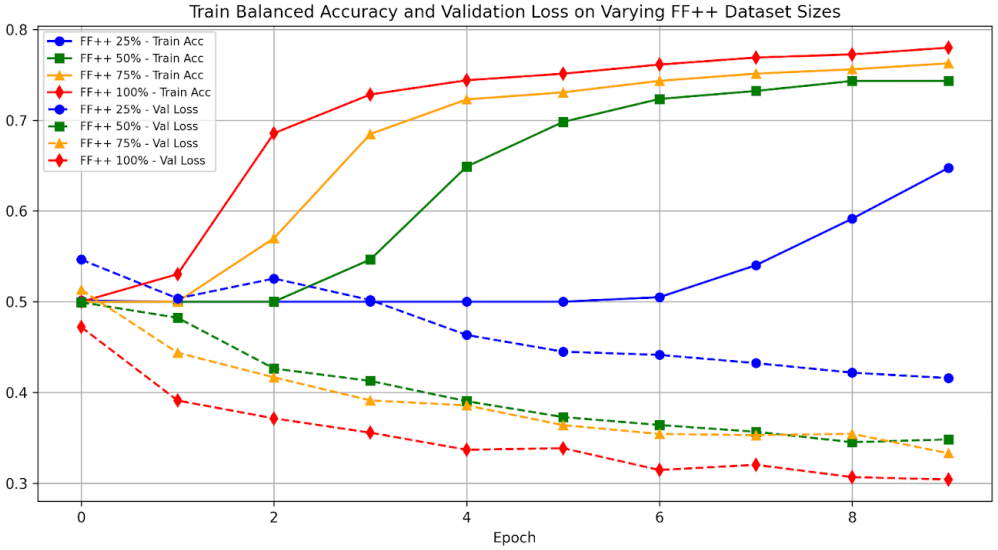
  loss ← CrossEntropyLoss(logits, labels)

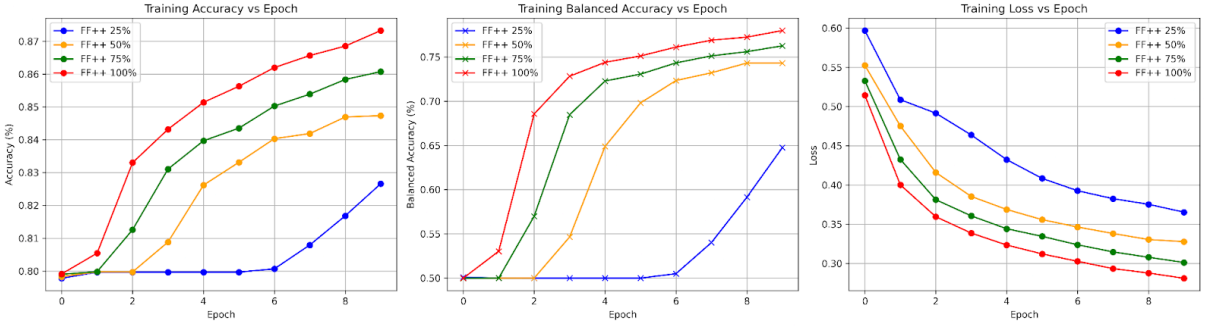
  loss.backward()

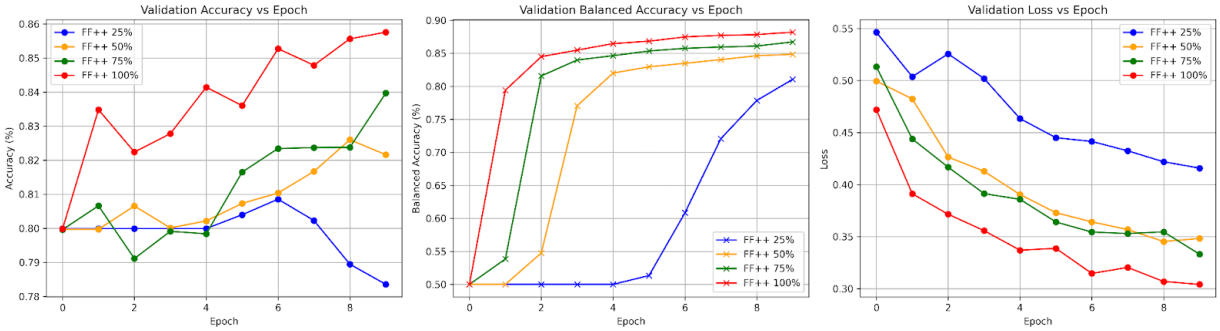
  optimizer.step()

Result on Training Data



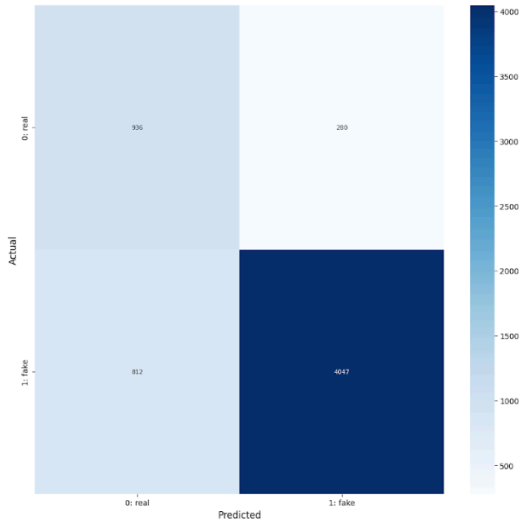


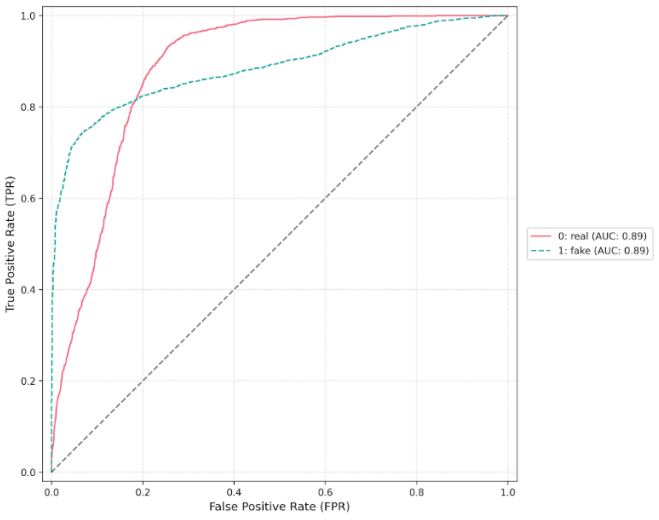


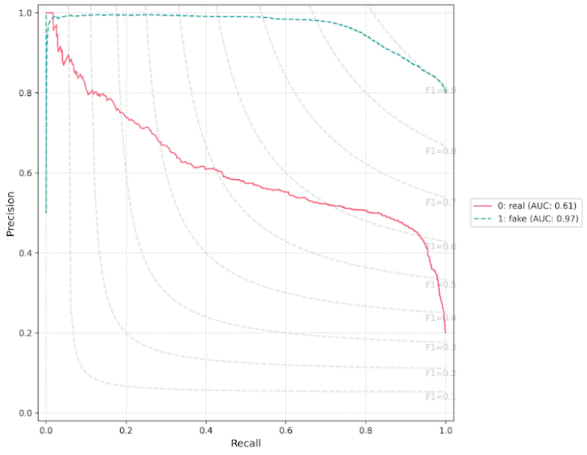


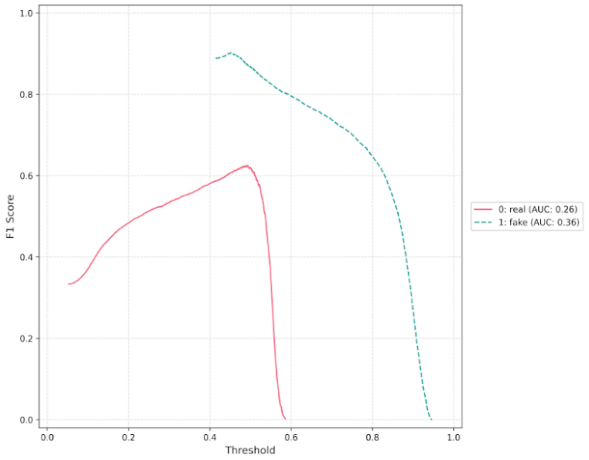
Result on Testing Data

FF25

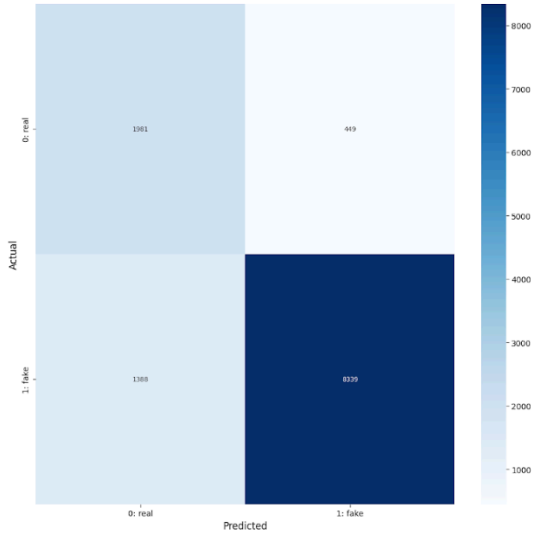


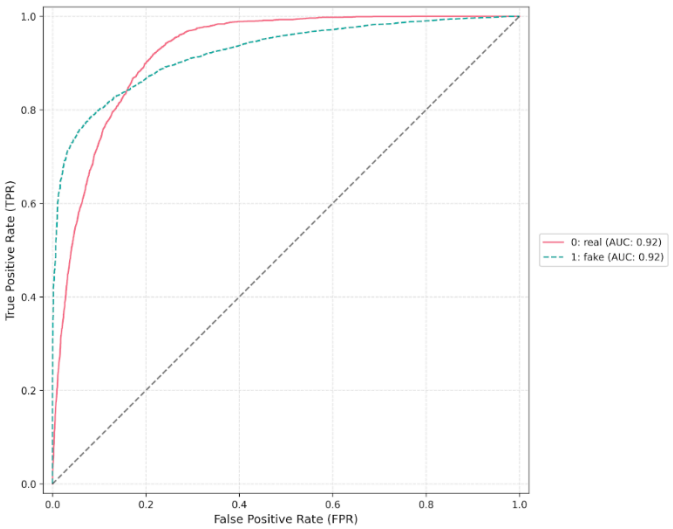


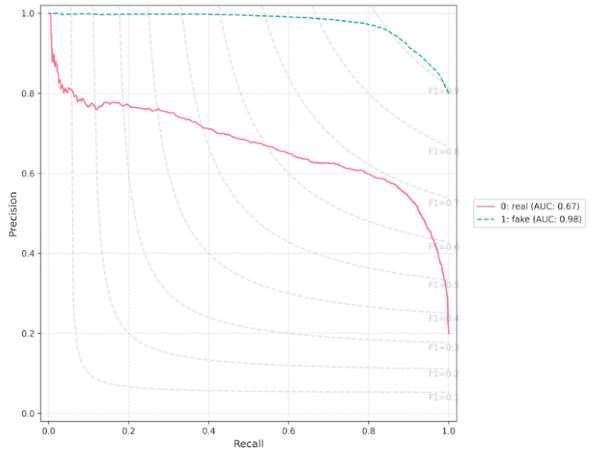


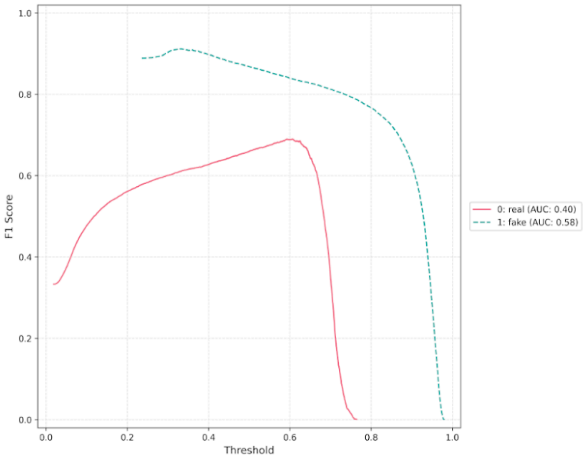


FF50

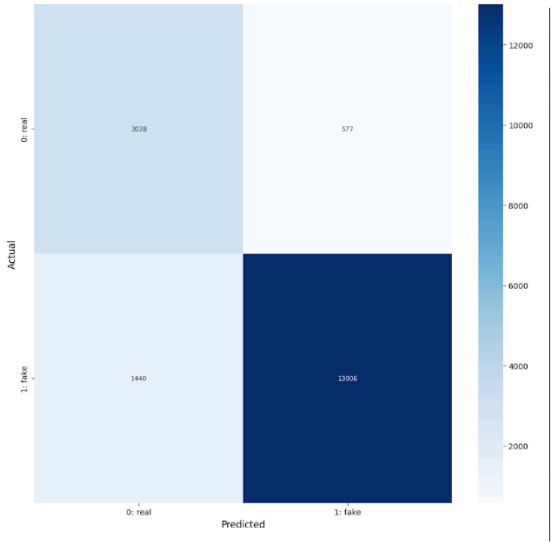


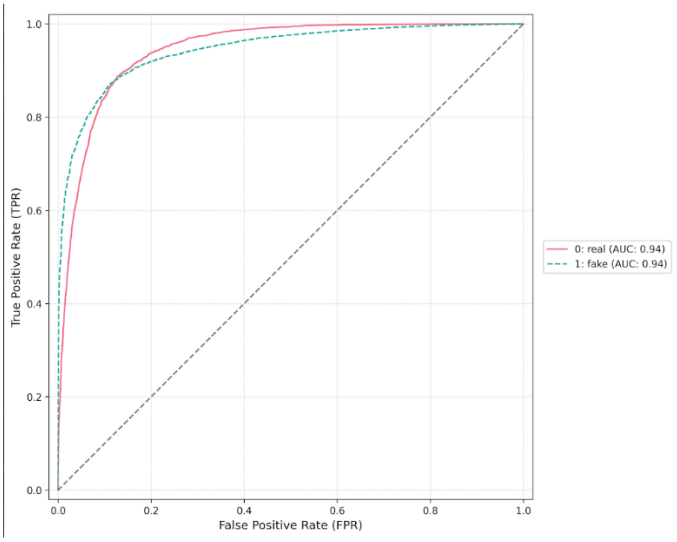


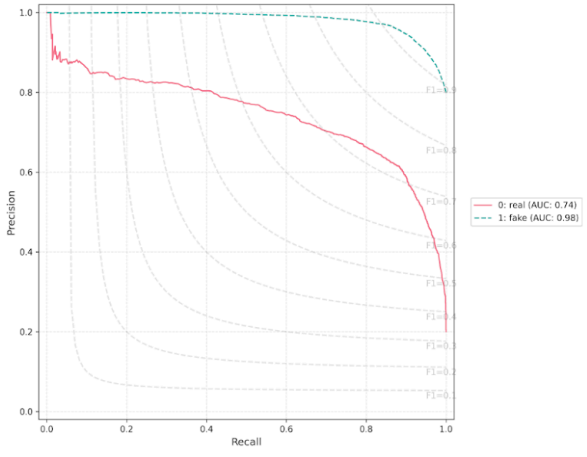

Precision
Recall
0: real (AUC: 0.67)
1: fake (AUC: 0.98)

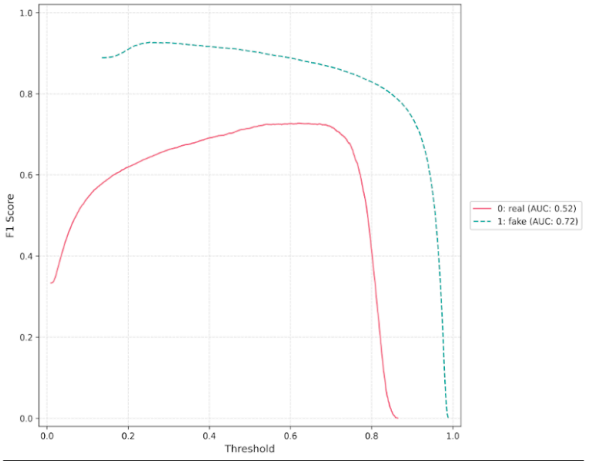

F1 Score
Threshold
0: real (AUC: 0.40)
1: fake (AUC: 0.58)

FF75

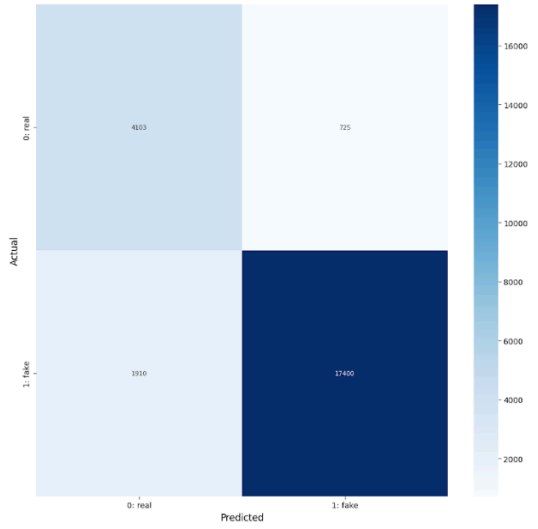

Actual
Predicted
0: real
1: fake
3038
577
1440
13006

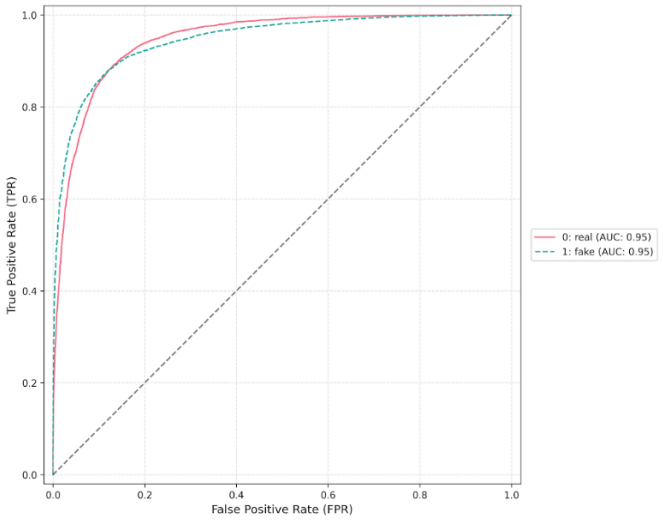

True Positive Rate (TPR)
False Positive Rate (FPR)
0: real (AUC: 0.94)
1: fake (AUC: 0.94)

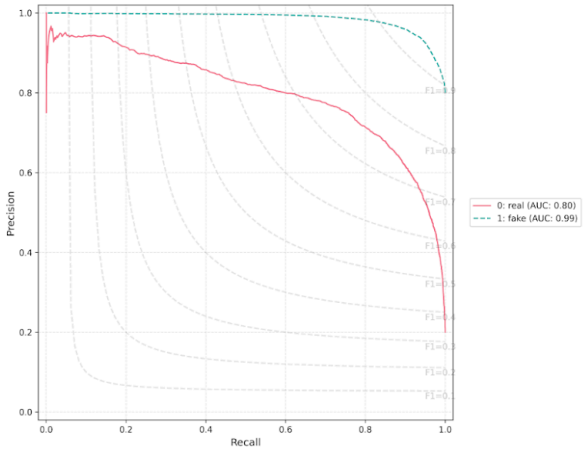

Precision
Recall
0: real (AUC: 0.74)
1: fake (AUC: 0.98)

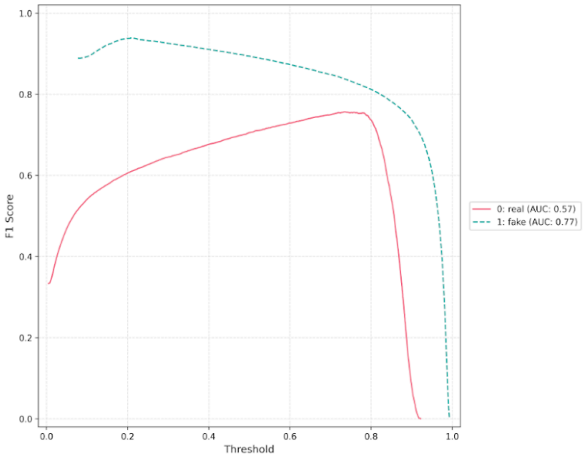

F1 Score
Threshold
0: real (AUC: 0.52)
1: fake (AUC: 0.72)

FF++









Overall Stats

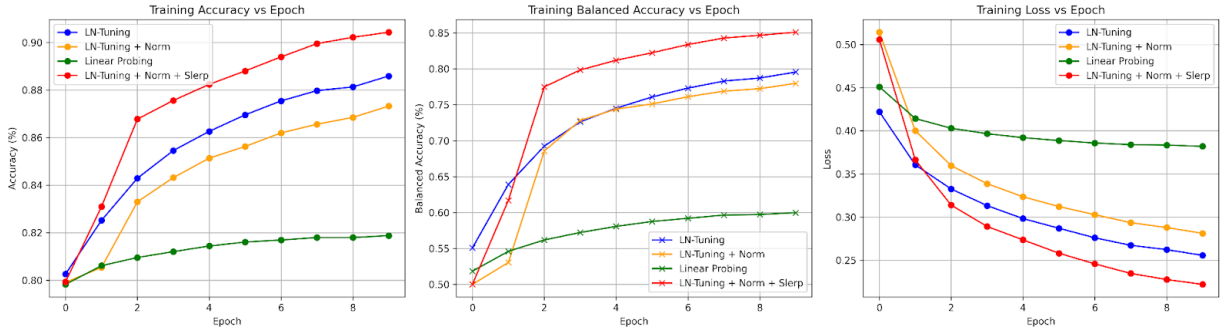
Dataset	Precision	Recall	F1 score	Accuracy
FF25	0.93	0.83	0.88	0.82
FF50	0.94	0.85	0.90	0.84
FF75	0.95	0.90	0.92	0.88
FF++	0.96	0.90	0.93	0.89

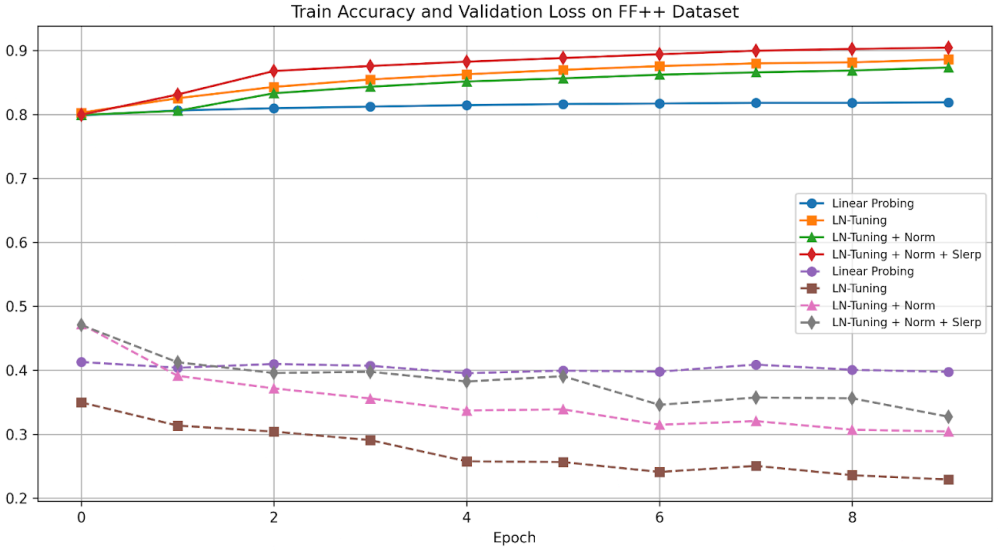
Justification

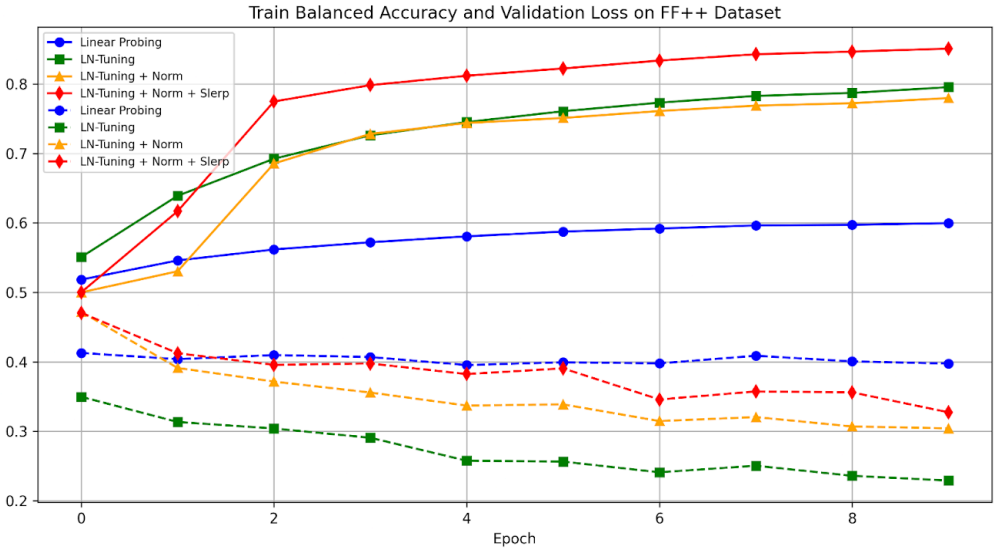
The model trained on FF++ 25% exhibits clear signs of struggle. Its learning is initially very slow, and despite a late surge in training accuracy, its validation loss remains stubbornly high and flat. This is a strong indication that 25% of the data is grossly insufficient for generalization and leads to severe overfitting (or failure to learn generalizable features) by epoch 9. The model is effectively memorizing the training set or failing to learn meaningful representations.

Comparison 1

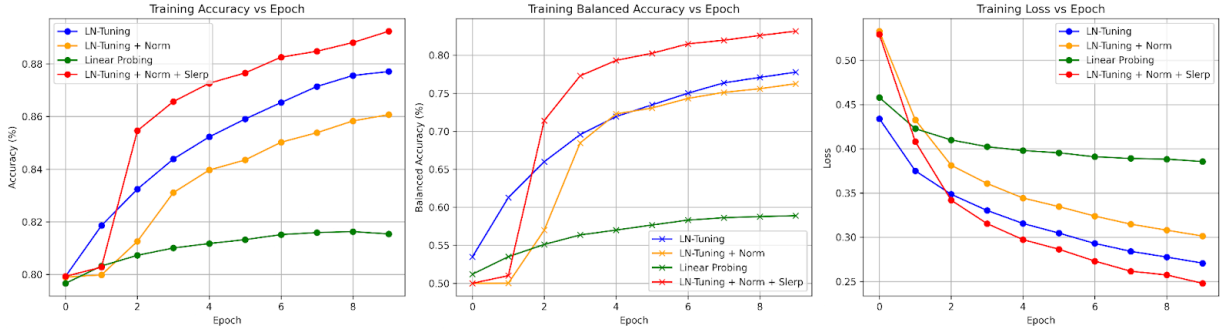
Comparison between techniques on FF++ dataset

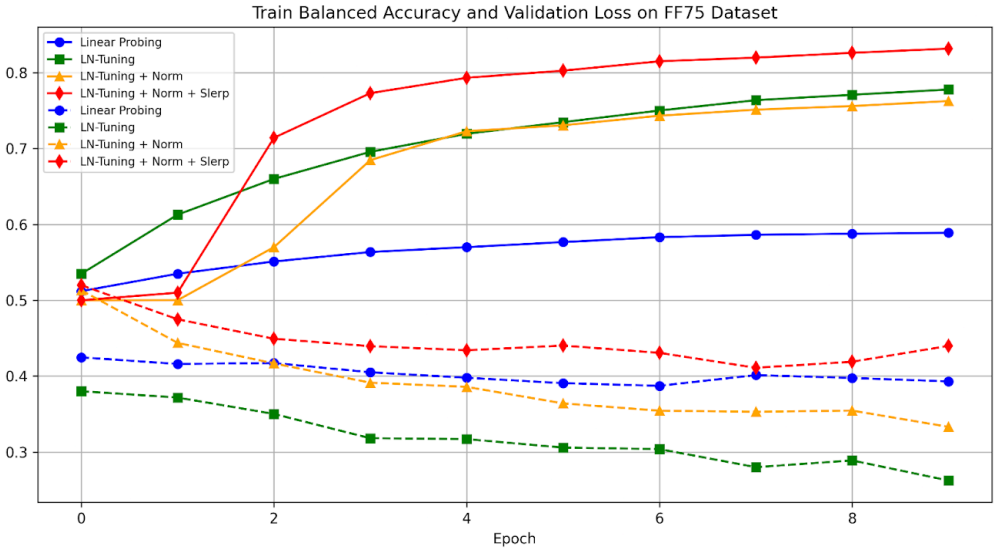


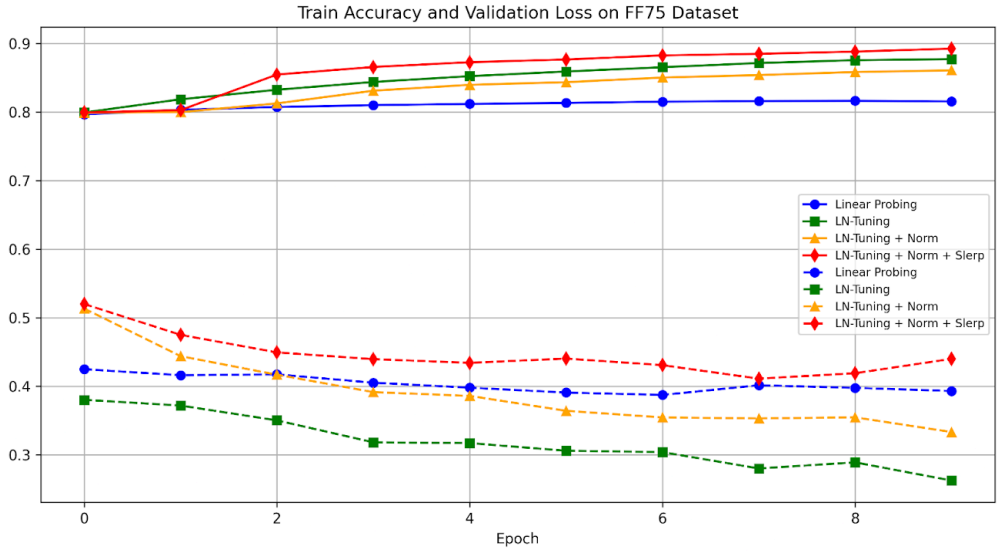




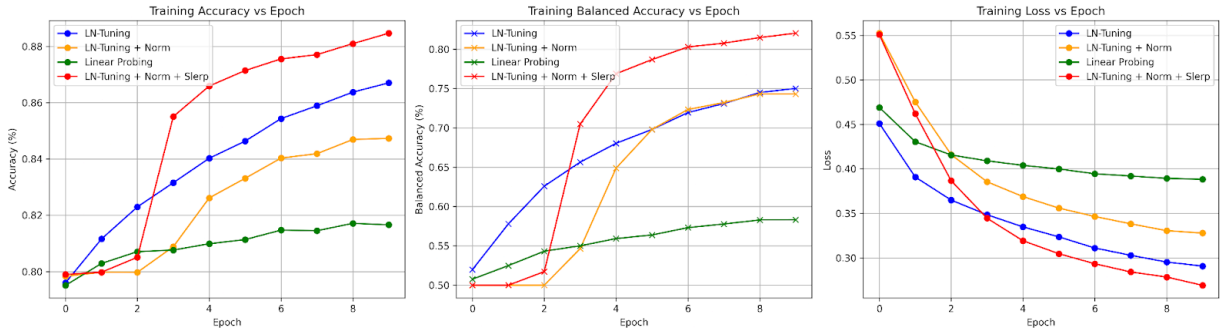
Comparison between techniques on FF75 dataset

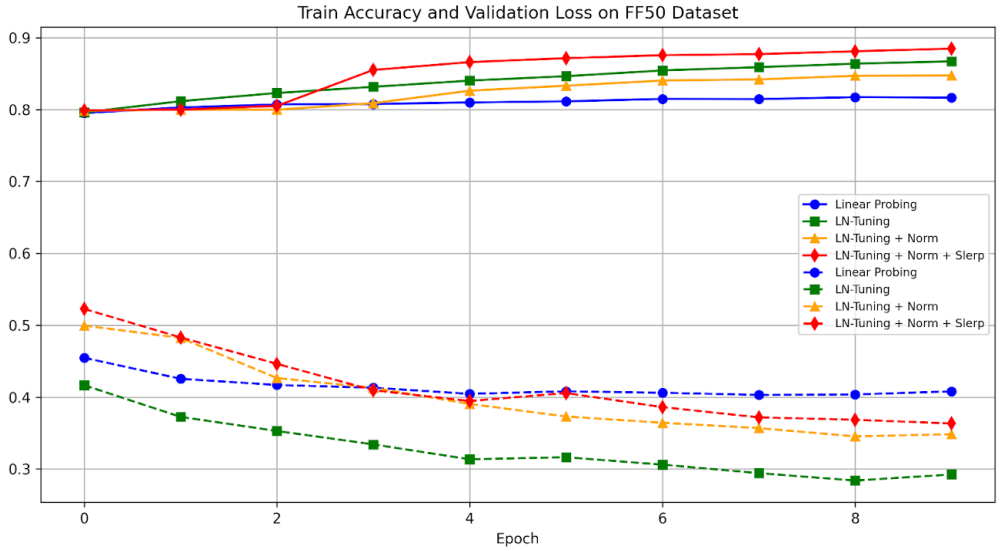


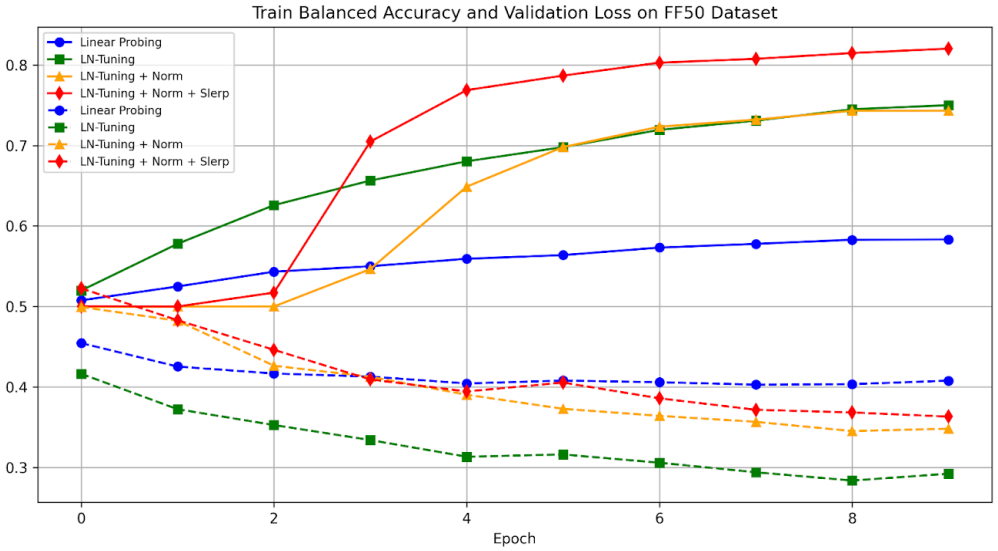




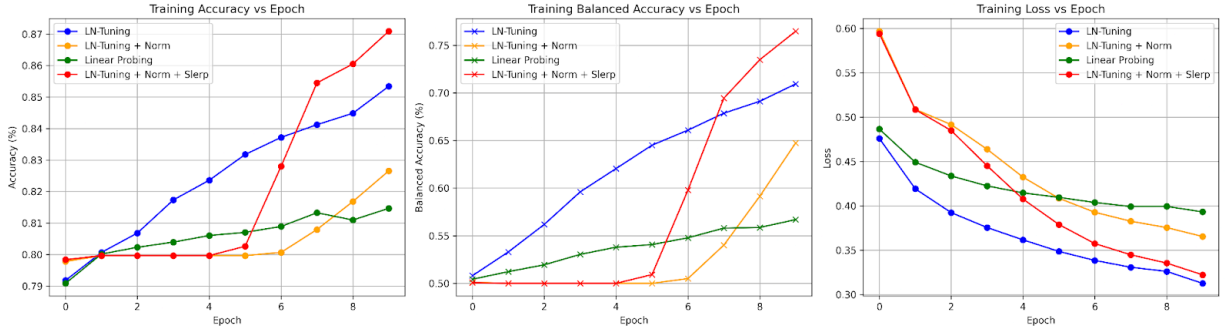
Comparison between techniques on FF50 dataset

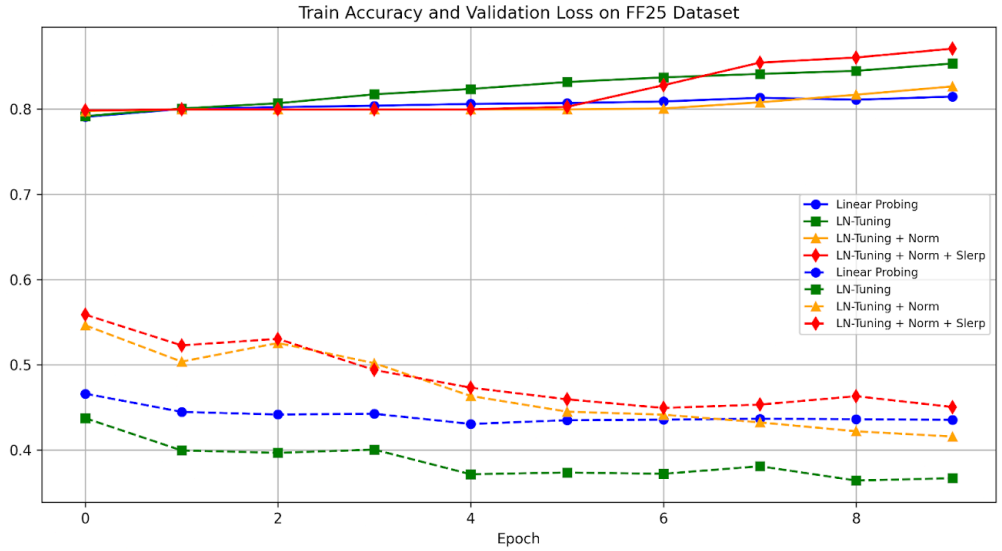


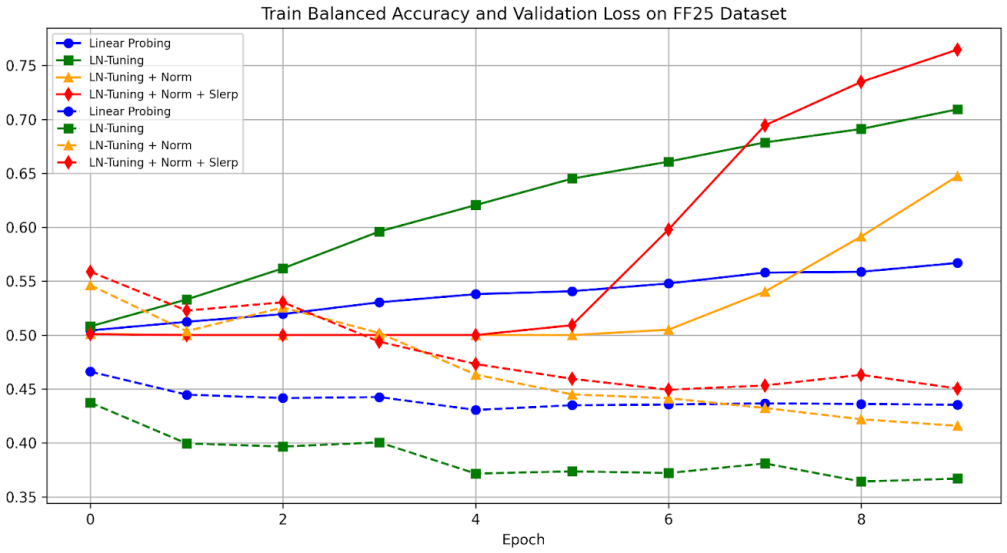




Comparison between techniques on FF25 dataset



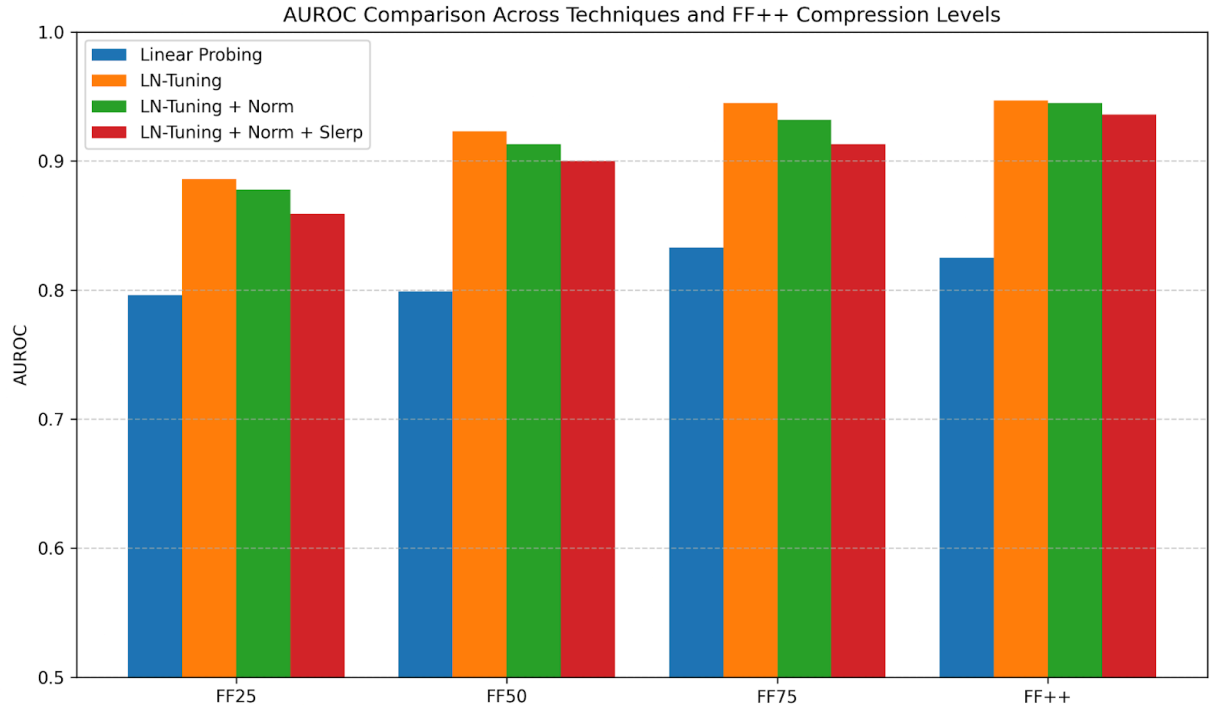

Train Accuracy and Validation Loss on FF25 Dataset
Linear Probing
LN-Tuning
LN-Tuning + Norm
LN-Tuning + Norm + Slerp
Linear Probing
LN-Tuning
LN-Tuning + Norm
LN-Tuning + Norm + Slerp
0.8
0.7
0.6
0.5
0.4
0
2
4
6
8
Epoch


Train Balanced Accuracy and Validation Loss on FF25 Dataset
Linear Probing
LN-Tuning
LN-Tuning + Norm
LN-Tuning + Norm + Slerp
Linear Probing
LN-Tuning
LN-Tuning + Norm
LN-Tuning + Norm + Slerp
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
2
4
6
8
Epoch

Comparison 2

AUROC Comparison Across Techniques on Varying size of FF++

	FF25	FF50	FF75	FF++
Linear Probing	0.796	0.799	0.833	0.825
LN Tuning only	0.886	0.923	0.945	0.947
LN Tuning+Norm	0.878	0.913	0.932	0.945
LN+Norm+Slerp	0.859	0.90	0.913	0.936



Justification

Observations:

1. **General Trend:** Across all methods, there's a clear trend of **increasing AROC scores with larger dataset sizes**. The "FF++" column (presumably 100% data) generally shows the highest scores, while "FF25" (25% data) shows the lowest. This reinforces the importance of data volume.

2. **Linear Probing (Baseline):**

    - Achieves the lowest AROC scores across all dataset sizes (ranging from 0.796 to 0.833).
    - Shows a modest improvement with more data, but caps out around 0.833.
3. **LN Tuning Only (Method 2):**
    - **Consistently achieves the highest AROC scores** among all methods.
    - Scores are significantly higher than Linear Probing, especially for smaller datasets (e.g., 0.886 for FF25 vs. 0.796).
    - Reaches impressive AROC values of 0.945 (FF75) and 0.947 (FF++).
4. **LN Tuning + Norm (Method 3):**
    - Performs very well, considerably better than Linear Probing.
    - AROC scores are slightly *lower* than "LN Tuning only" for FF25, FF50, and FF75.
    - For FF++, its AROC (0.945) is almost identical to "LN Tuning only" (0.947).
5. **LN + Norm + Slerp (Method 4):**

    - Performs better than Linear Probing, but notably **underperforms both "LN Tuning only" and "LN Tuning+Norm"** across all dataset sizes.
    - Its AROC scores are the lowest among the three advanced fine-tuning methods (LN Tuning variants).
    - For example, for FF25, its AROC is 0.859, while "LN Tuning only" is 0.886. For FF++, it's 0.936 compared to 0.947 for "LN Tuning only".

LayerNorm Tuning is the Most Effective Strategy:

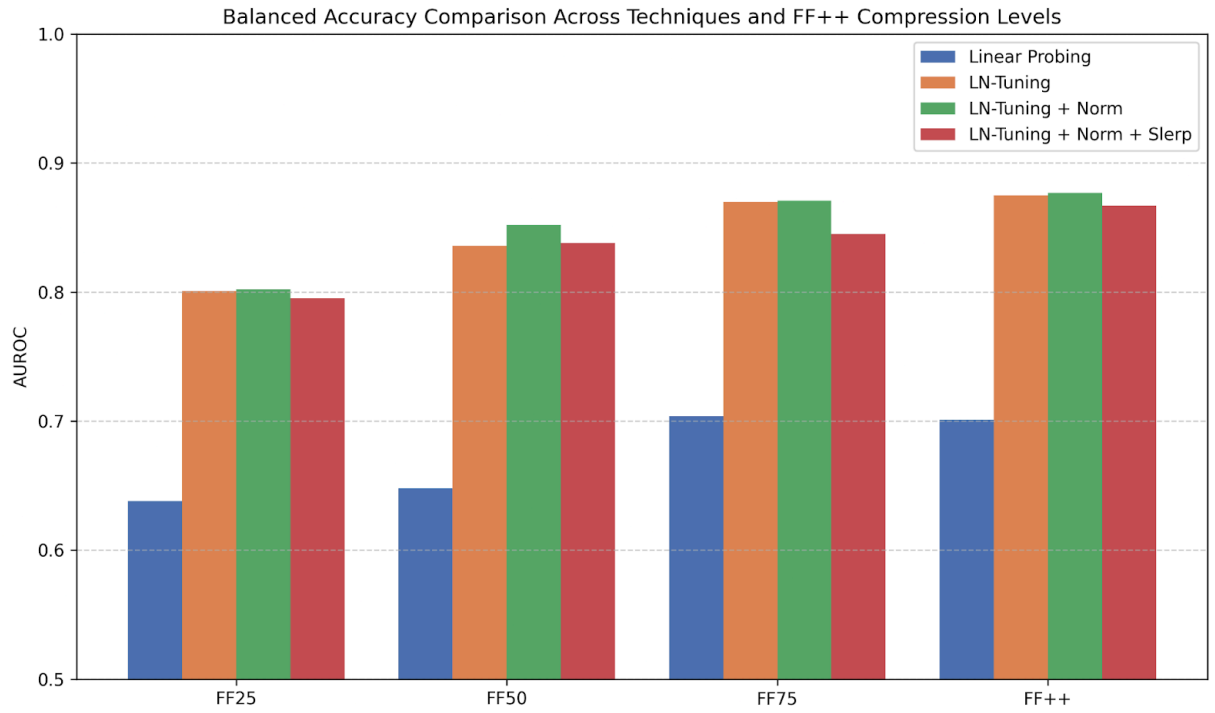
L2 Normalization (Method 3) Offers Marginal Benefits (or even Slight Detriment) to AROC:

L2 normalization on the CLS token did not consistently improve the overall discriminative ability (AROC) when compared to just LN Tuning. In fact, for most dataset sizes, it slightly reduced the AROC. This indicates that while it might shift the feature distribution and affect threshold-dependent metrics like accuracy, it doesn't necessarily improve the overall separability across all thresholds.

Slerp (Method 4) Does Not Translate to Higher AROC:

Synthetic Data Limitations: While Slerp generates more data, this synthetic data might not truly represent the complex variations of real-world test data as effectively as the model adapting its LayerNorms directly with real data does, particularly when judging overall separability.

Balanced Accuracy Comparison Across Techniques on Varying size of FF++



	FF25	FF50	FF75	FF++
Linear Probing	0.638	0.648	0.704	0.701
LN only	0.801	0.836	0.870	0.875
LN + Norm	0.802	0.852	0.871	0.877
LN +Norm +Slerp	0.795	0.838	0.845	0.867

Performance Interpretation Summary

Observations:

1. **General Trend:** Consistent with previous observations and the AROC table, **larger dataset sizes generally lead to higher Balanced Test Accuracy** across all methods.

2. **Linear Probing (Baseline):**
   - Again, it consistently performs the lowest, with balanced accuracies ranging from 0.638 to 0.704.
   - This confirms its limited effectiveness as a standalone strategy on frozen features.

3. **LN Tuning only (Method 2):**
    - Shows a significant jump in performance over Linear Probing (e.g., 0.801 for FF25 vs. 0.638).
    - Achieves high balanced accuracies, reaching 0.870 for FF75 and 0.875 for FF++.

4. **LN + Norm (Method 3):**
    - Performs very similarly to "LN Tuning only" for FF25 (0.802 vs. 0.801).
    - **Notably, it achieves slightly higher balanced accuracy than "LN Tuning only" for FF50 (0.852 vs. 0.836), FF75 (0.871 vs. 0.870), and FF++ (0.877 vs. 0.875).** This is a key difference from the AROC table.

5. **LN + Norm + Slerp (Method 4):**
    - Also performs significantly better than Linear Probing.
    - For FF25, its balanced accuracy (0.795) is slightly lower than "LN Tuning only" (0.801) and "LN + Norm" (0.802).
    - For FF50, it's very similar to "LN Tuning only" (0.838 vs. 0.836) but slightly lower than "LN + Norm" (0.852).
    - For FF75 and FF++, its balanced accuracies (0.845 and 0.867 respectively) are consistently the lowest among the LN tuning variants, though still very high overall.

LayerNorm Tuning Remains Highly Effective

L2 Normalization (Method 3) Shows Benefits in Balanced Accuracy

Slerp (Method 4) Does Not Consistently Lead to Top Balanced Accuracy

Potential Overfitting to Augmented Data

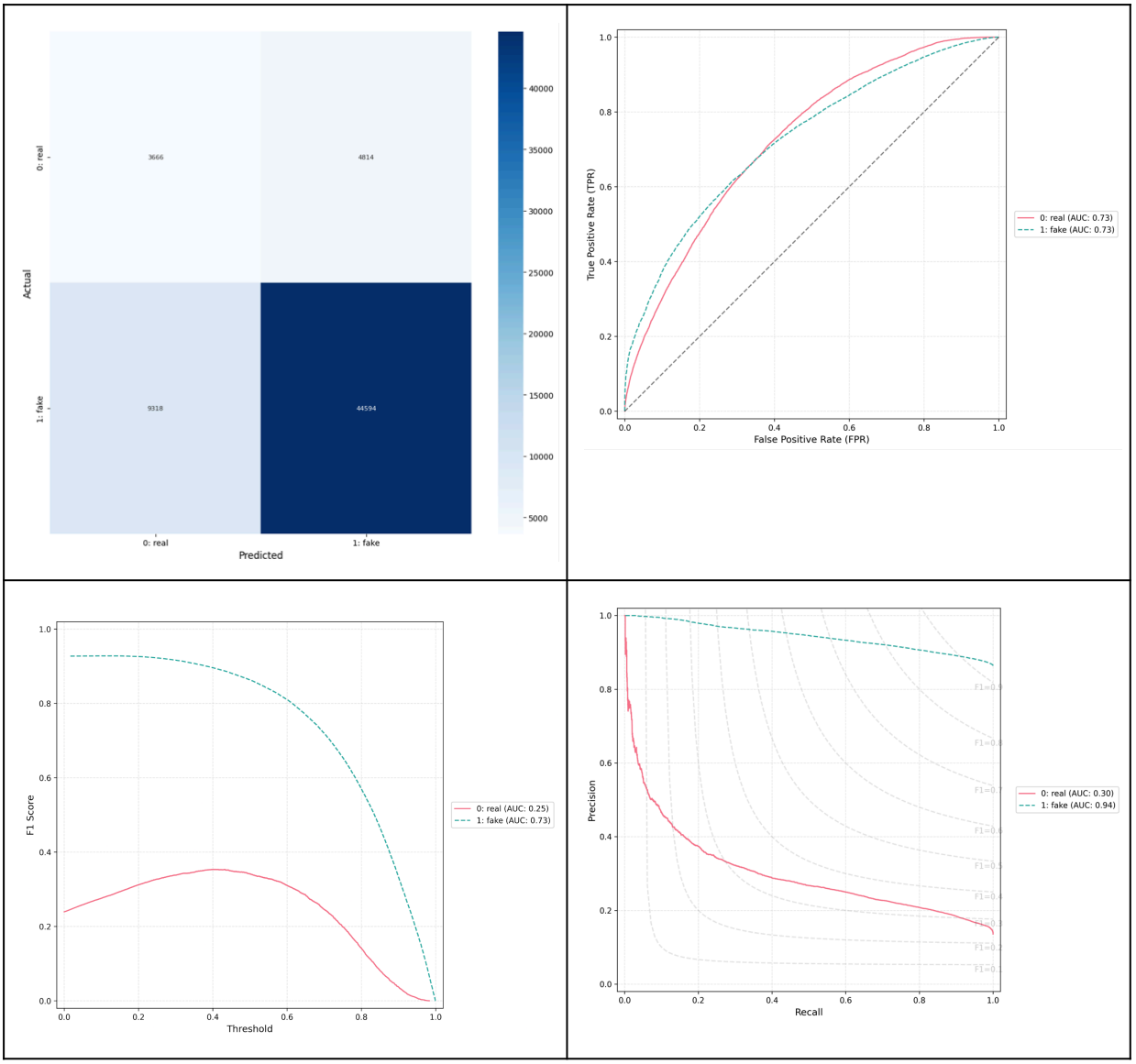
Trade-off in Generalization: The additional complexity from Slerp and the multiple loss terms might create a model that is very good at specific aspects of the training data but not as robust across the full spectrum of unseen test samples for balanced classification.

Cross-Dataset Evaluation:

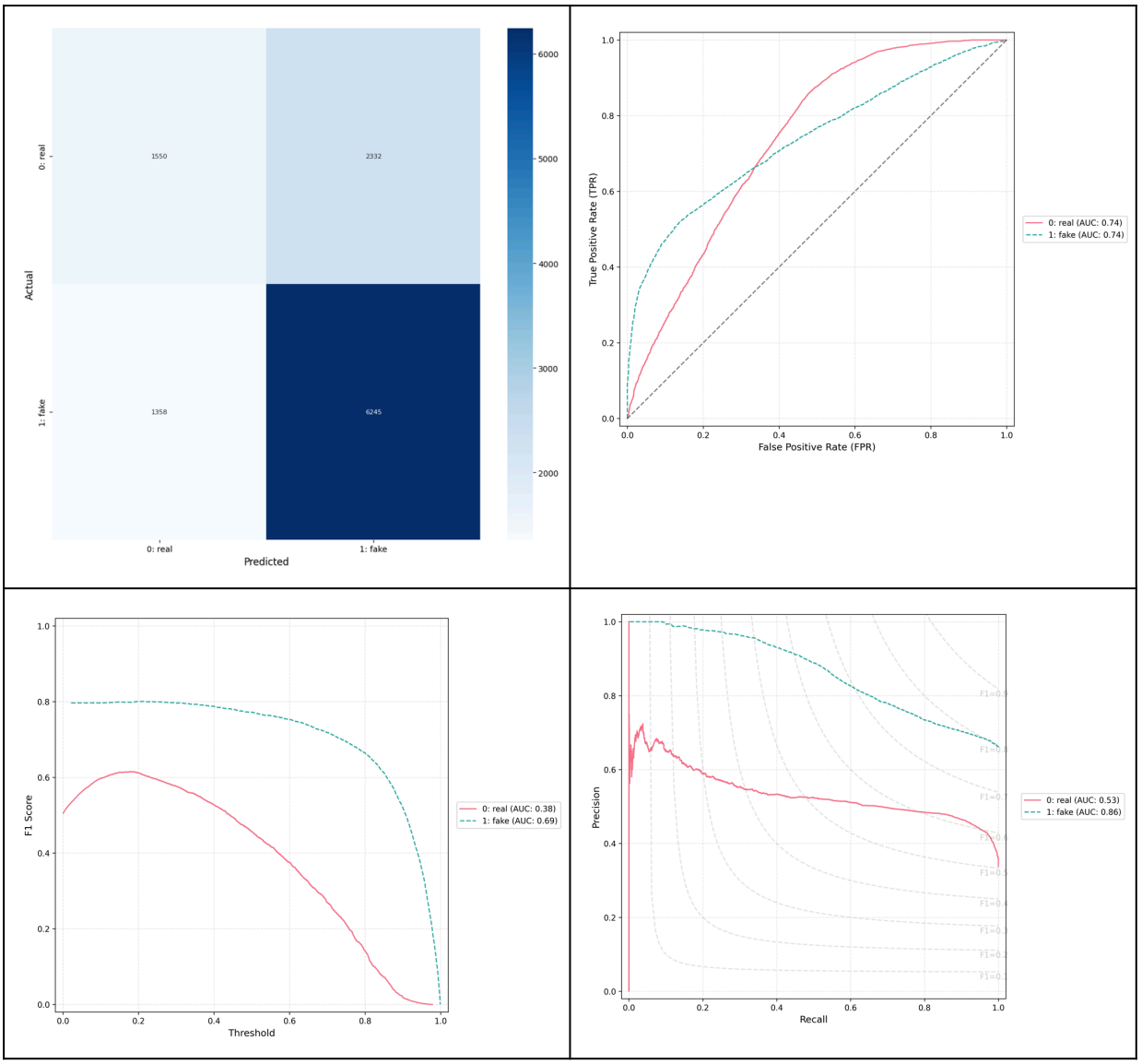
- To assess the generalization capability of each technique, trained the model on the FaceForensics++ (FF++) dataset and evaluated it on three unseen datasets: Celeb-DF v1, Celeb-DF v2, UADFV & DFD
- This cross-dataset evaluation was conducted for all four methods — Linear Probing, LN-Tuning, LN-Tuning + Norm, and LN-Tuning + Norm + SLERP — to benchmark their robustness against domain shifts and unseen deepfake distributions.

Method I - Linear Probing

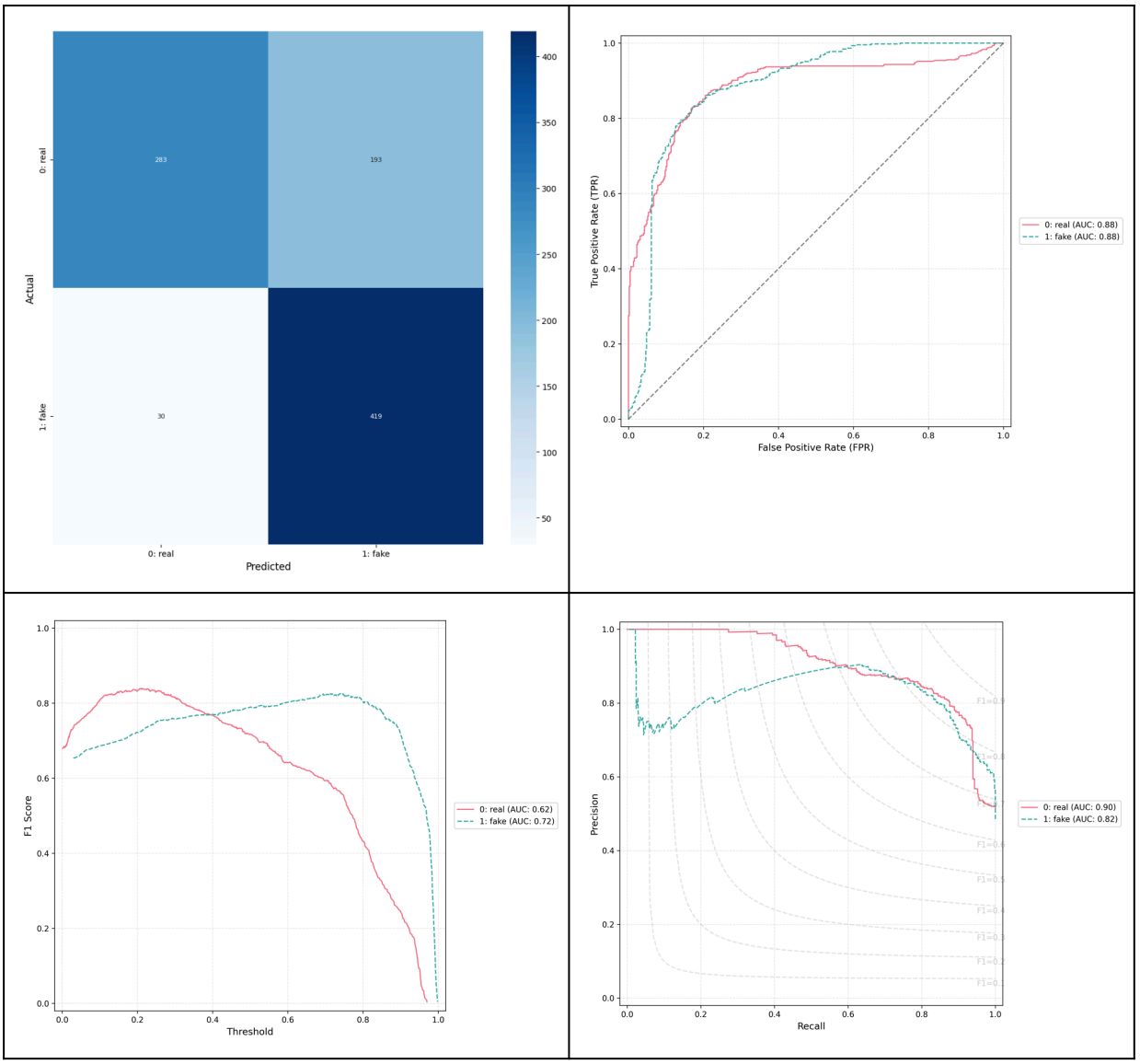
Result on Celebv1



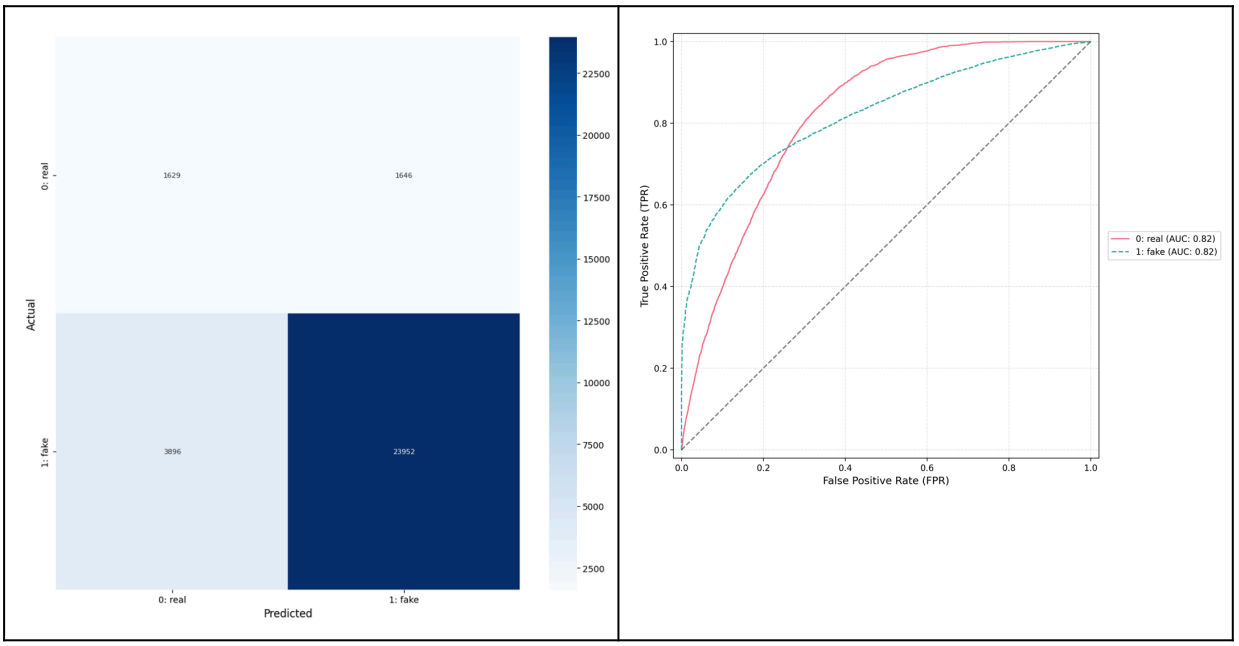
Result on Celebv2 Dataset

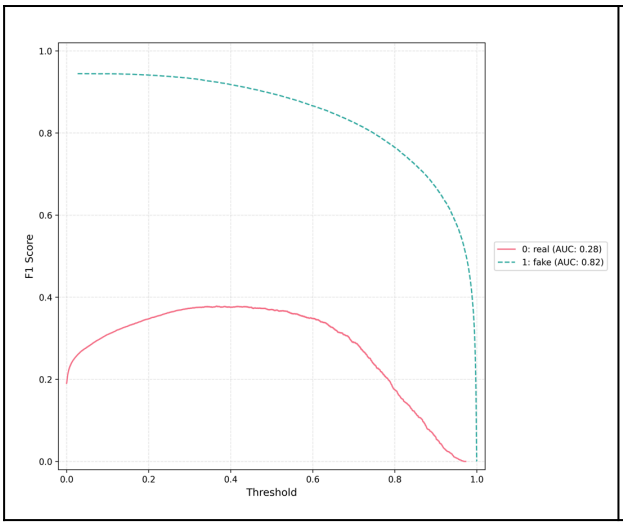


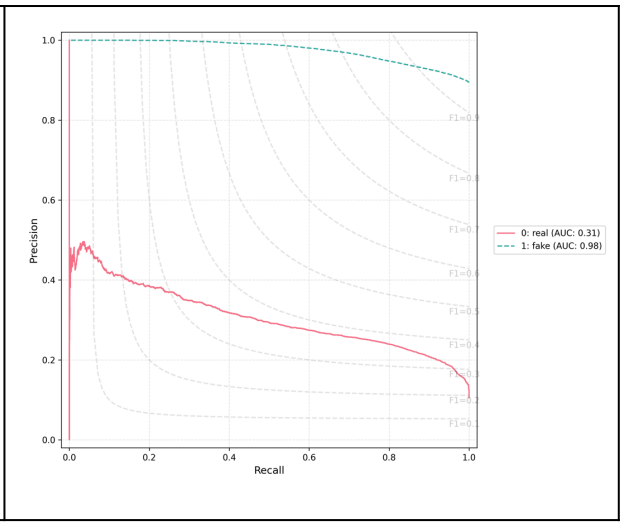
Result on UADFV Dataset



Result on DFD Dataset

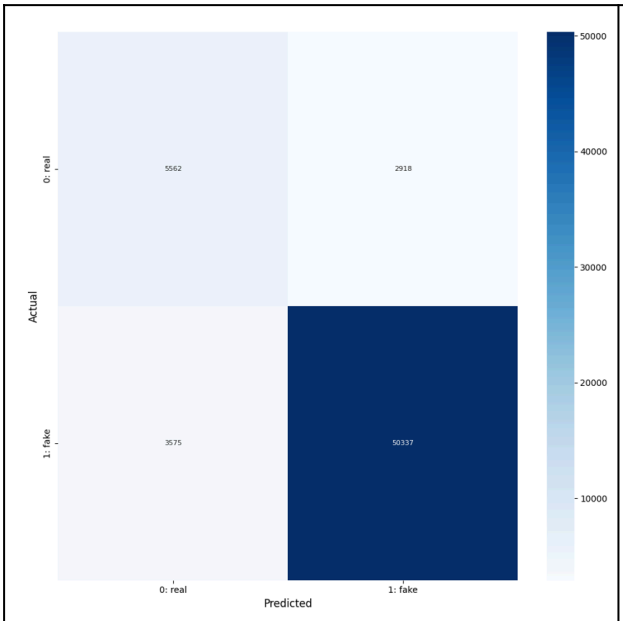


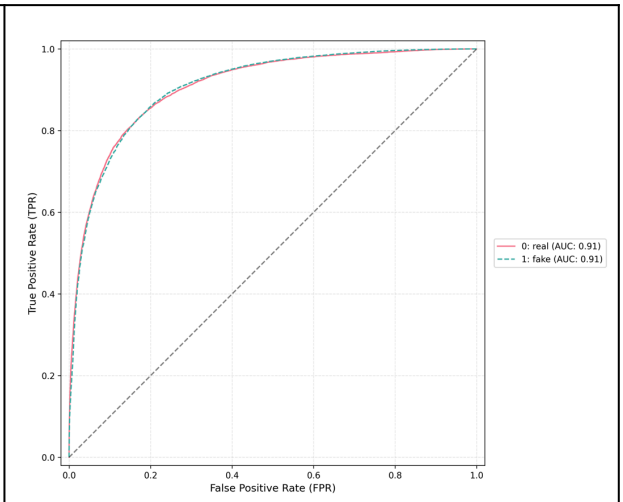


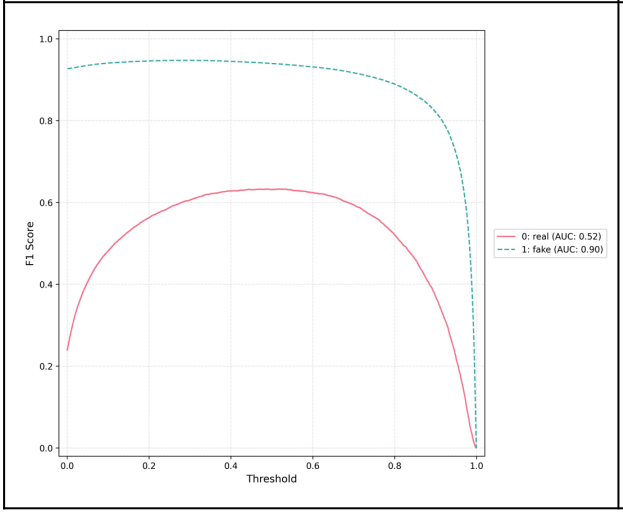


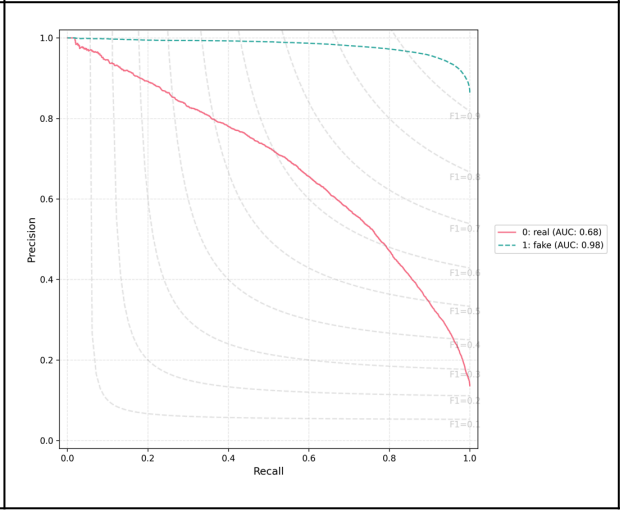
Method II - LN-Tuning

Result on Celebv1 Dataset

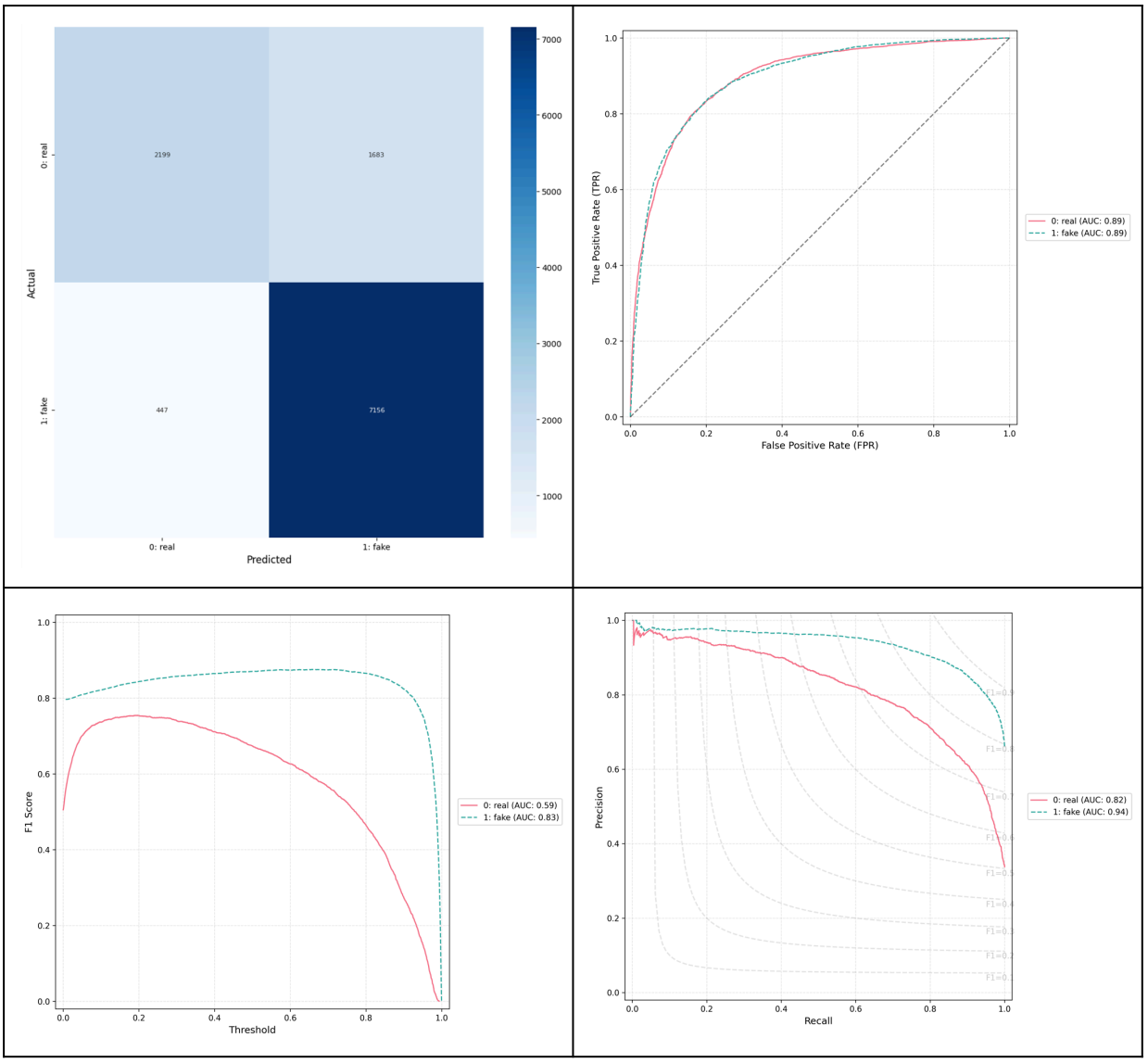




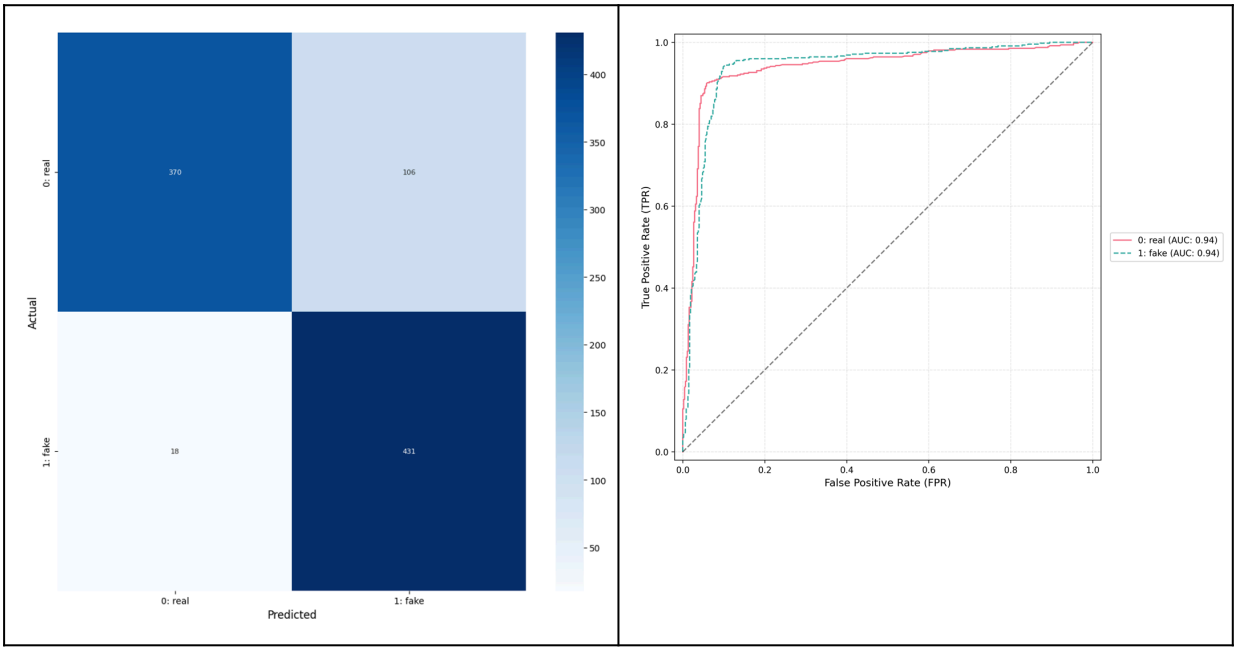


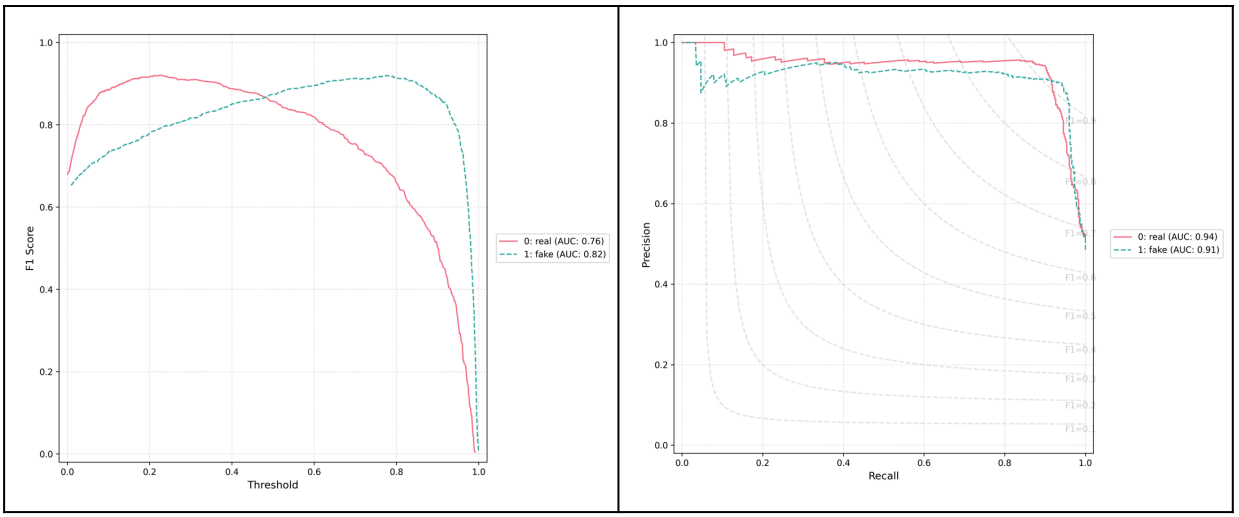


Result on Celebv2 Dataset

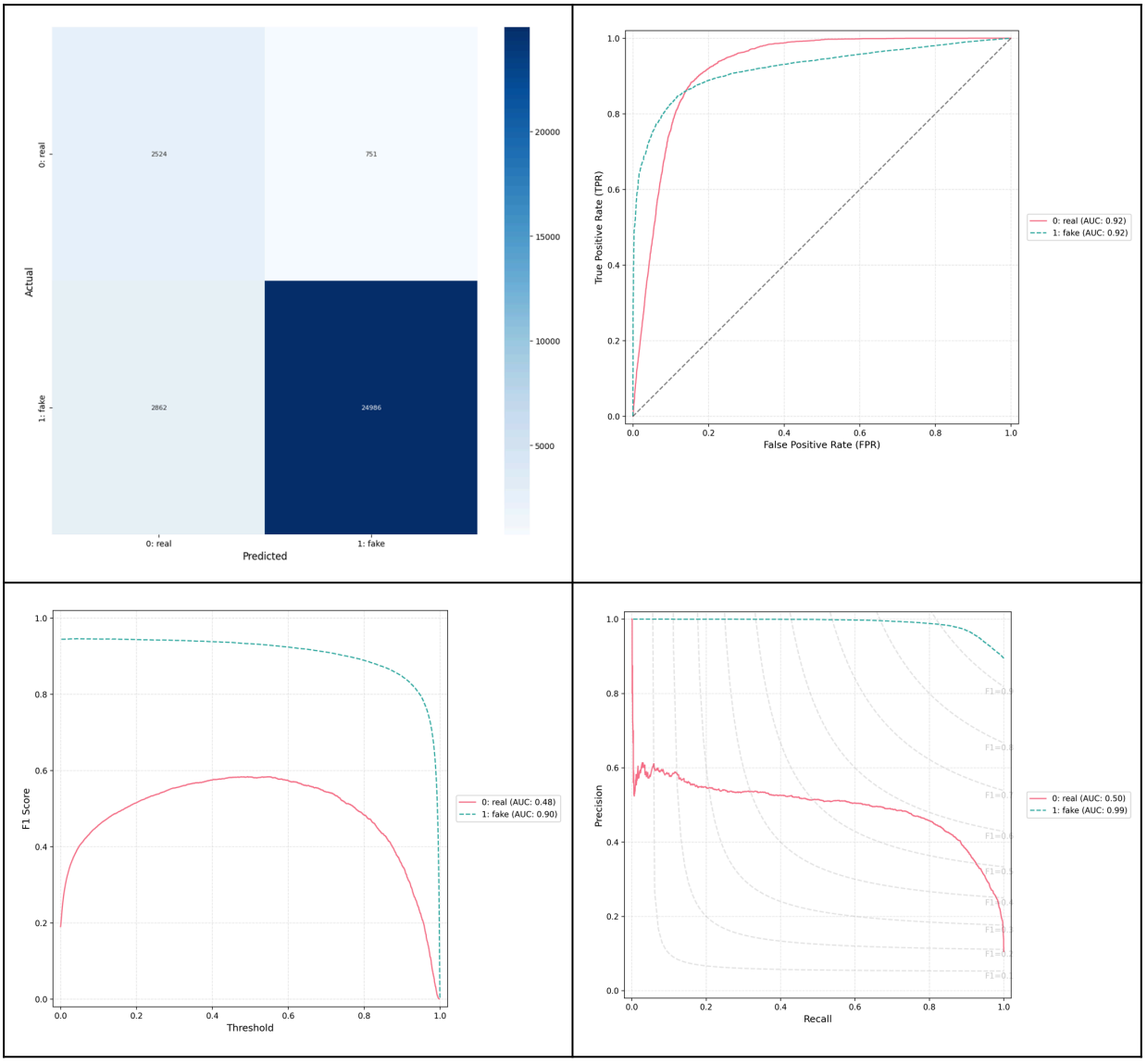


Result on UADFV Dataset



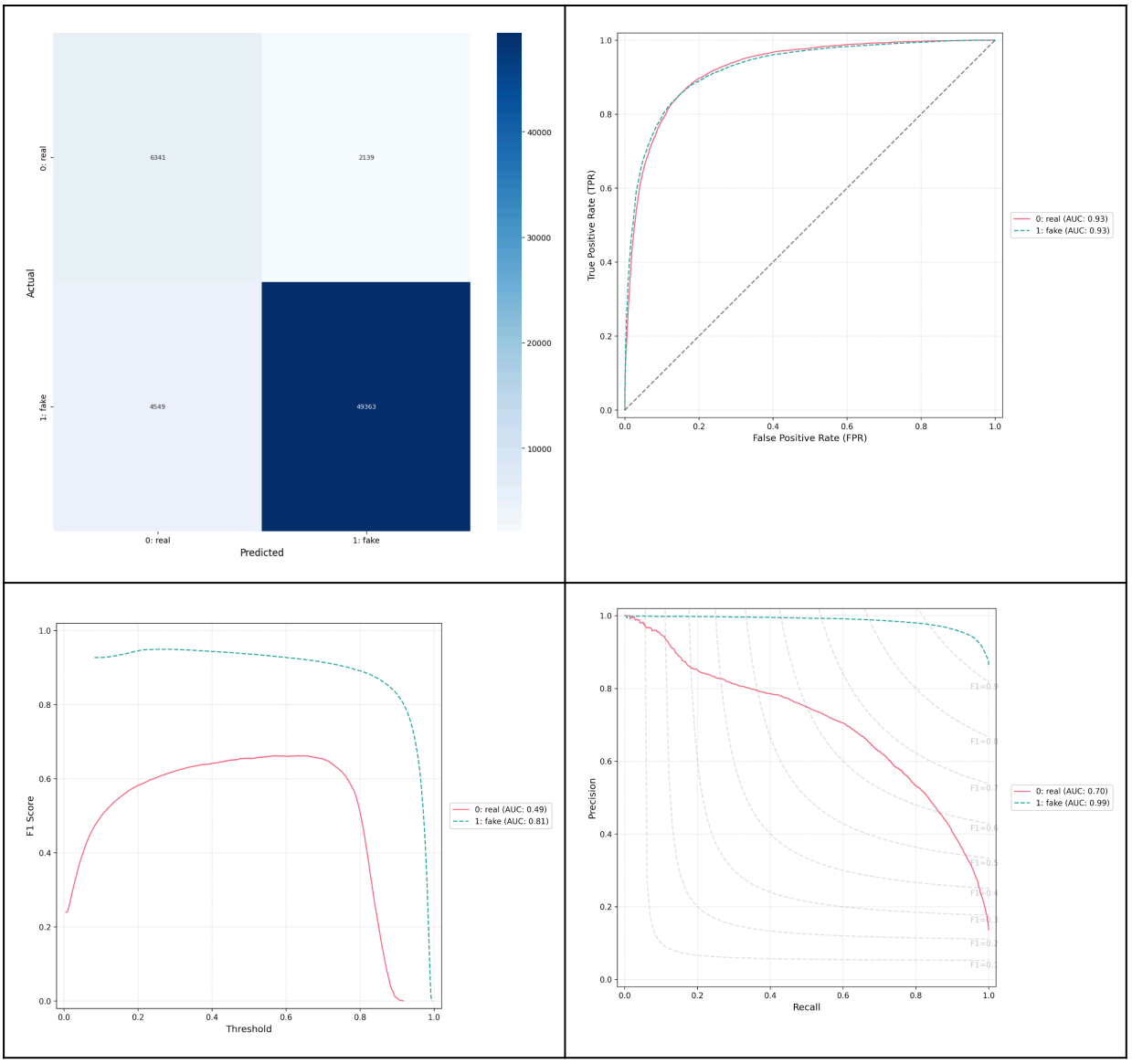


Result on DFD Dataset

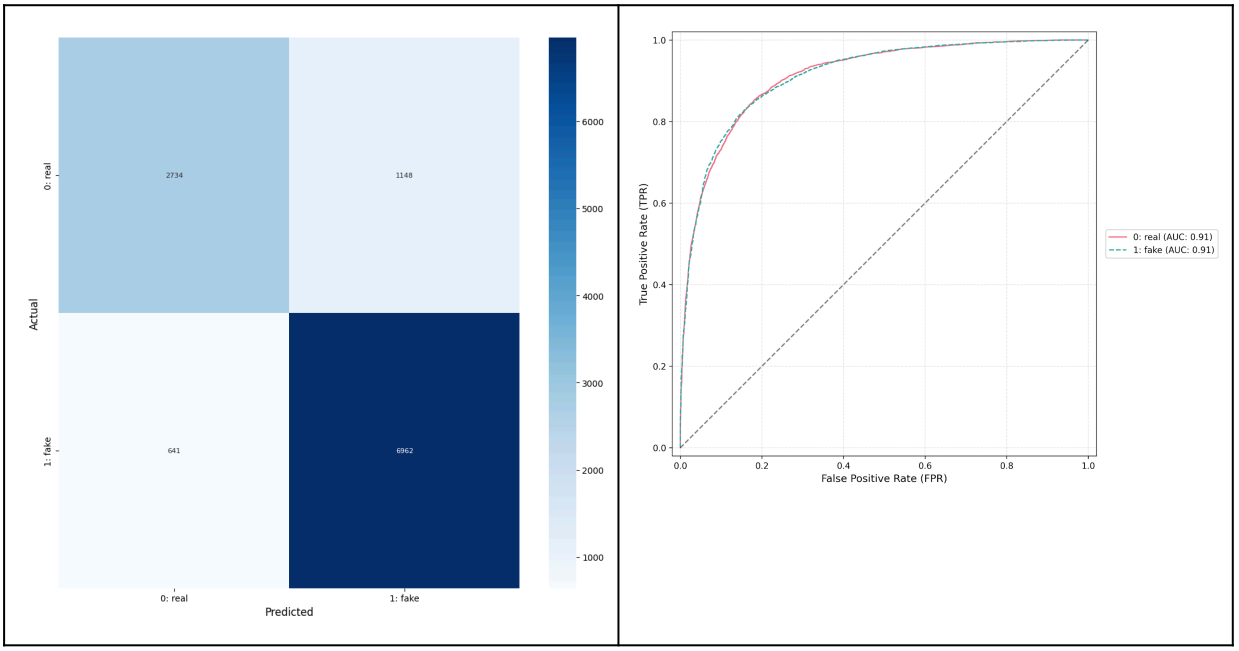


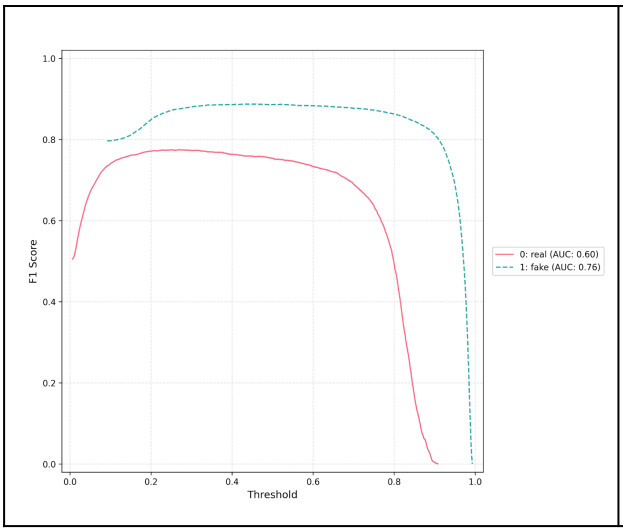
Method III - LN-Tuning+Norm

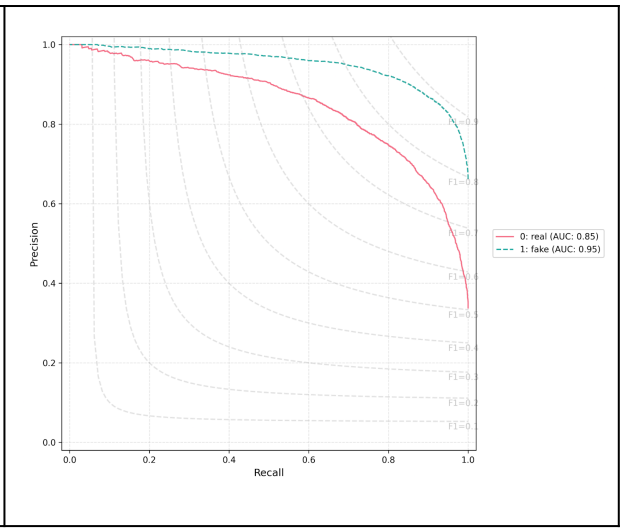
Result on Celebv1 Dataset



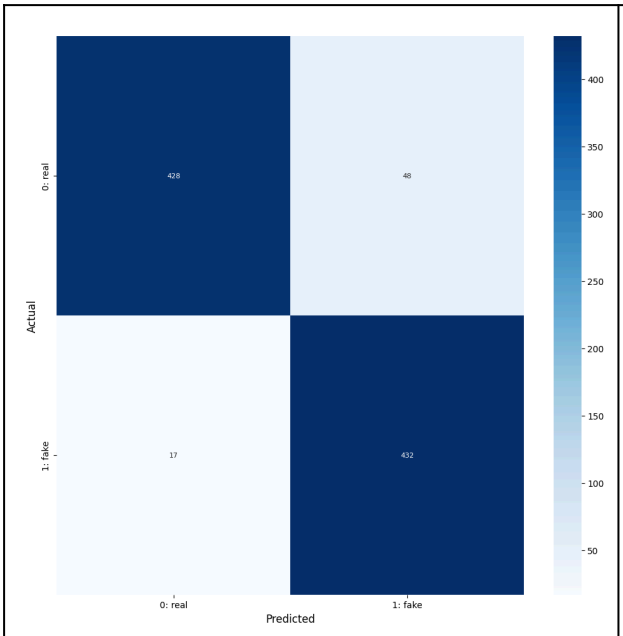
Result on Celebv2 Dataset

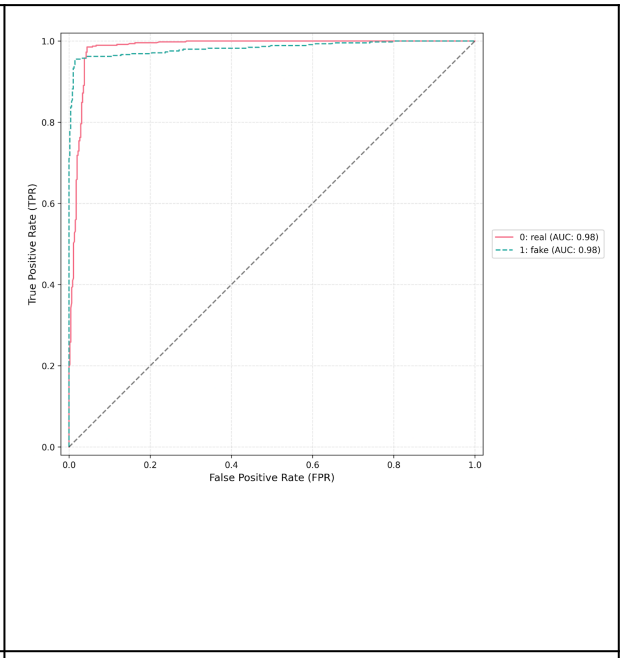


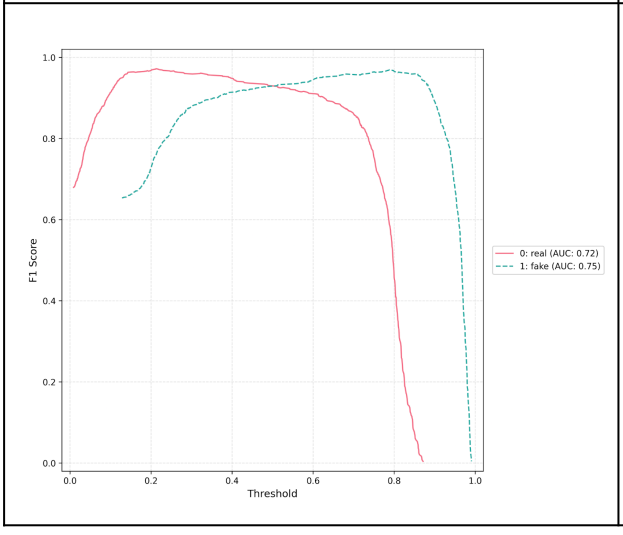


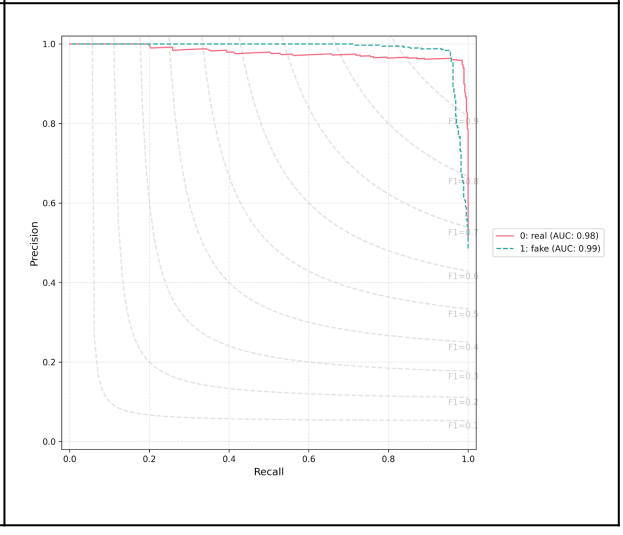


Result on UADFV Dataset

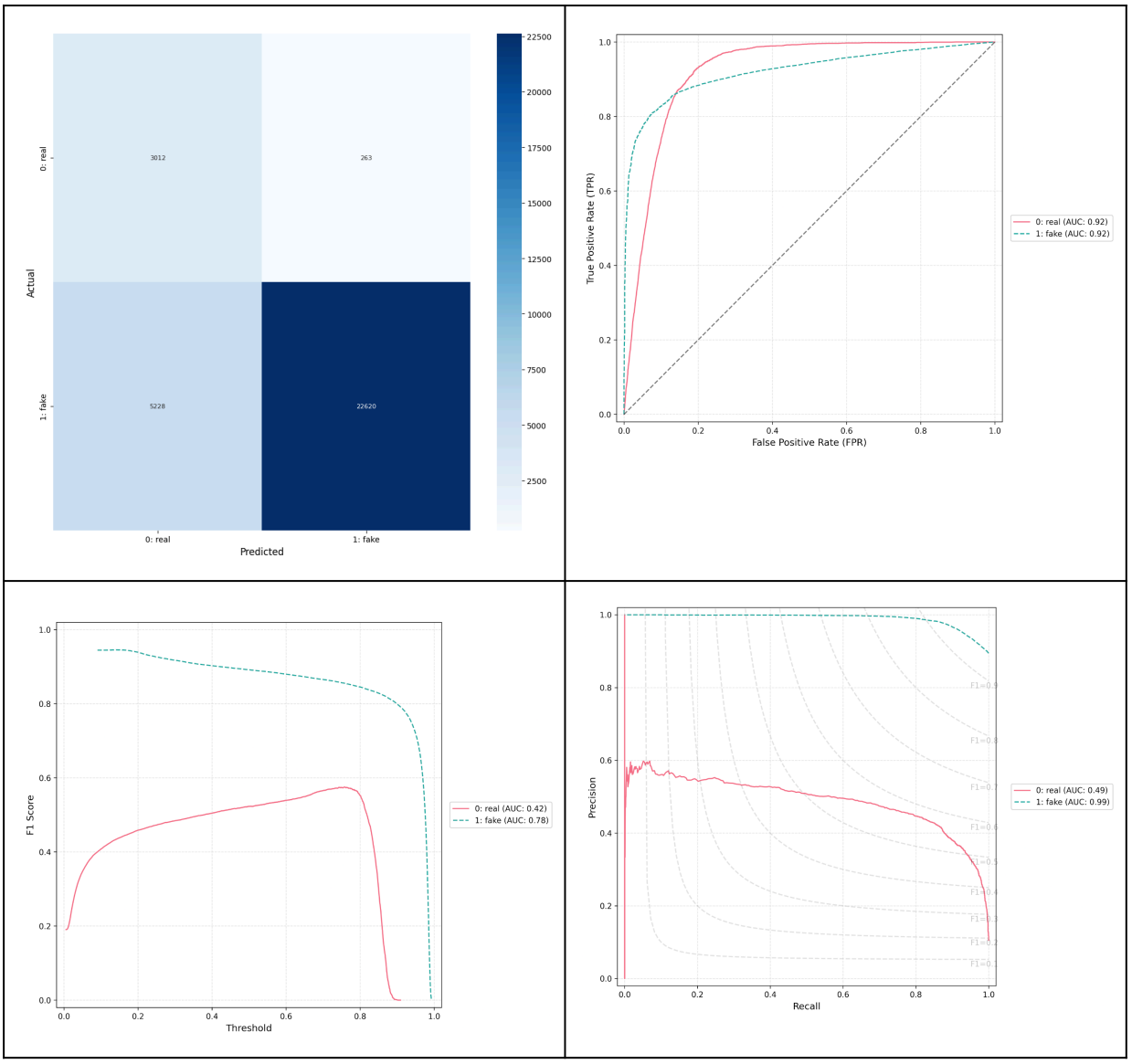






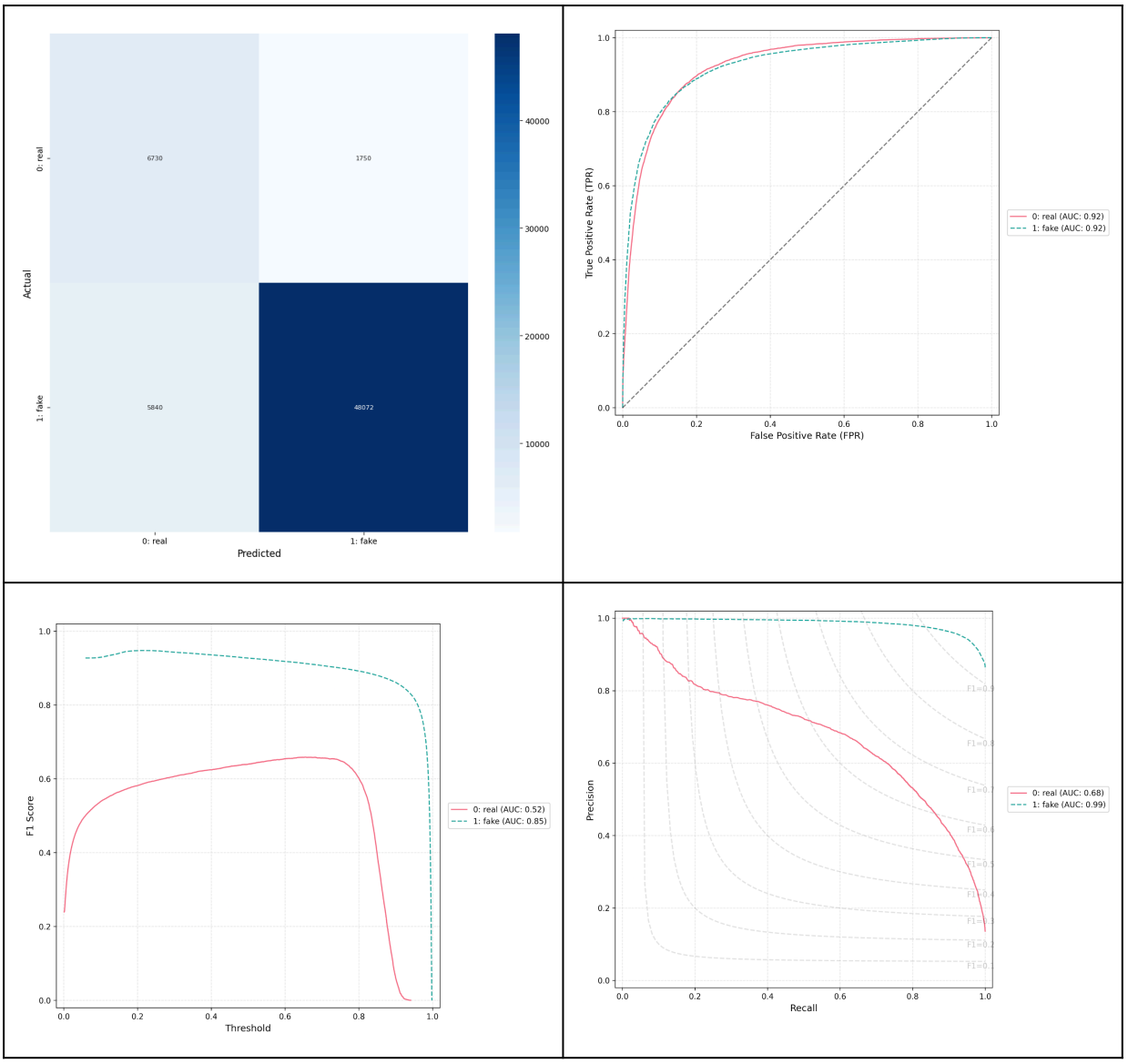


Result on DFD Dataset

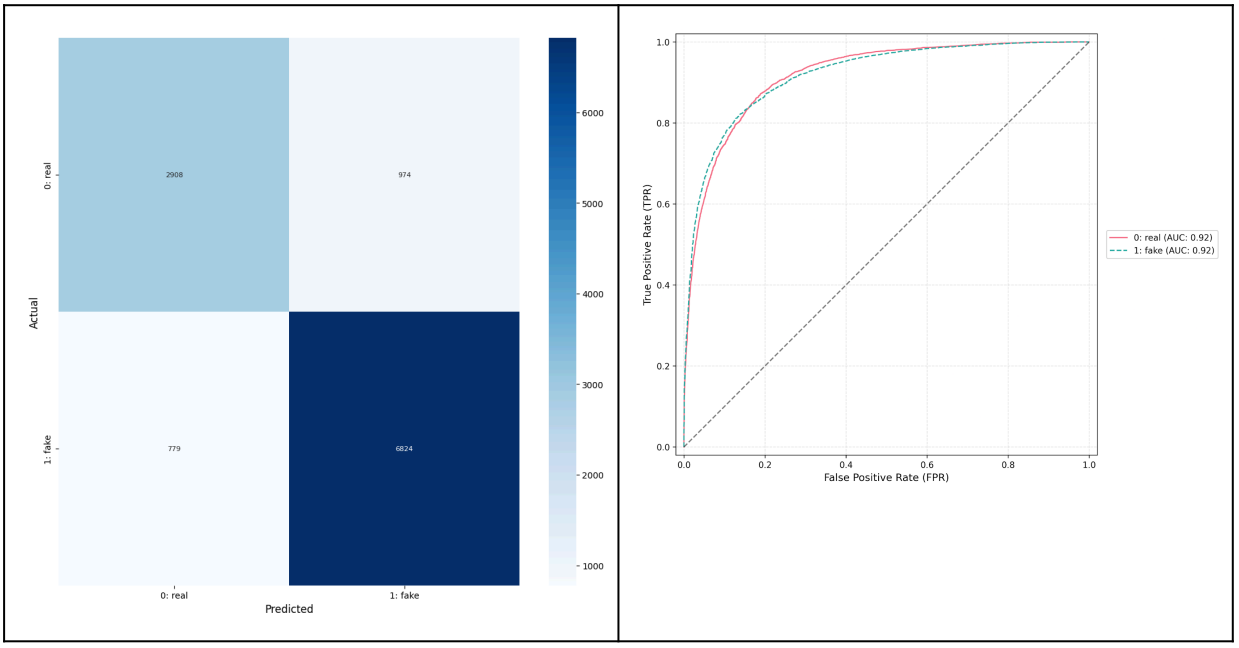


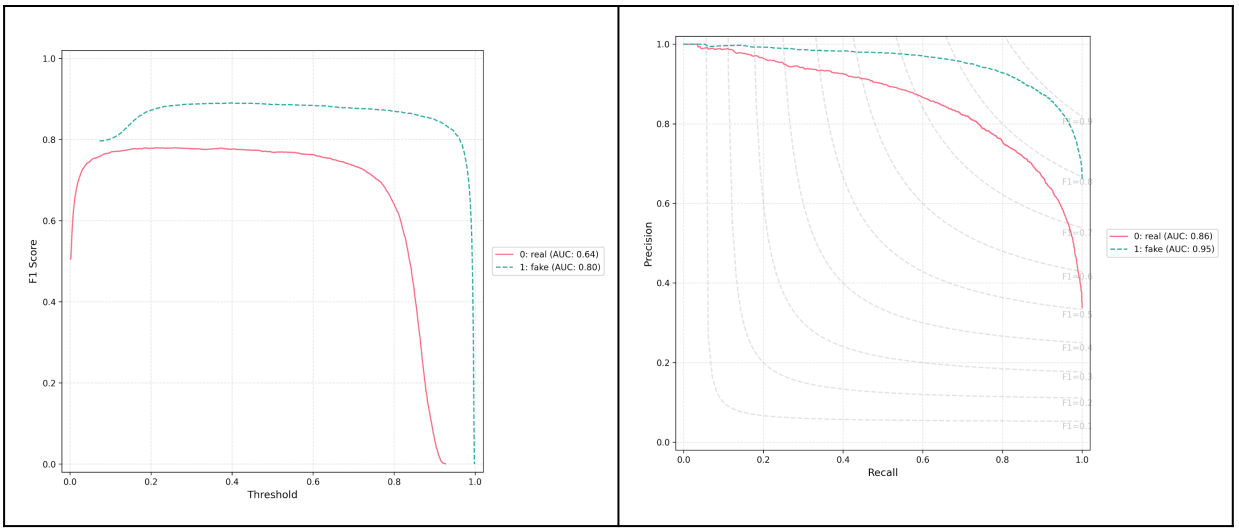
Method IV - LN-Tuning+Norm+Slerp

Result on Celebv1 Dataset

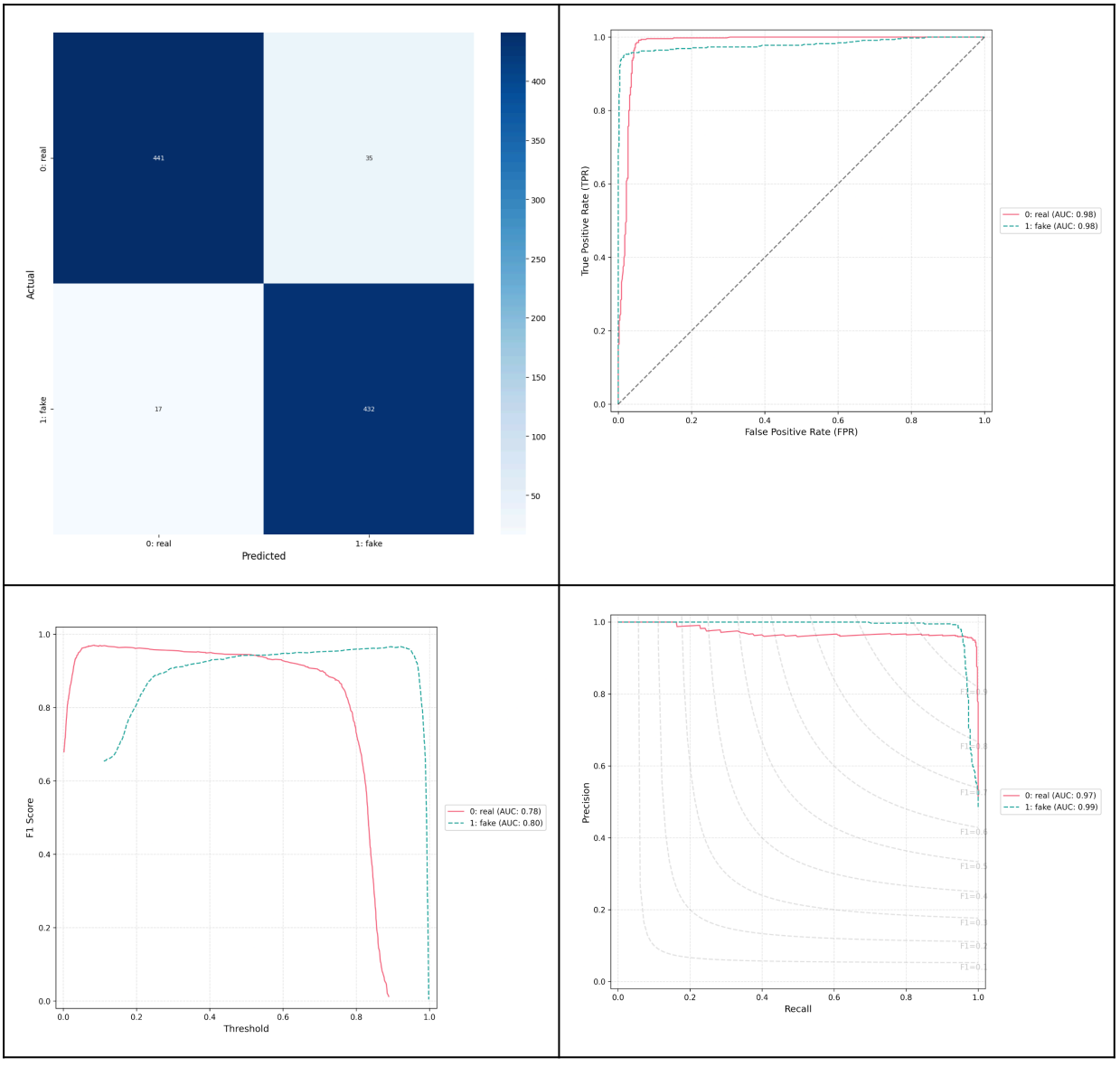


Result on Celebv2 Dataset

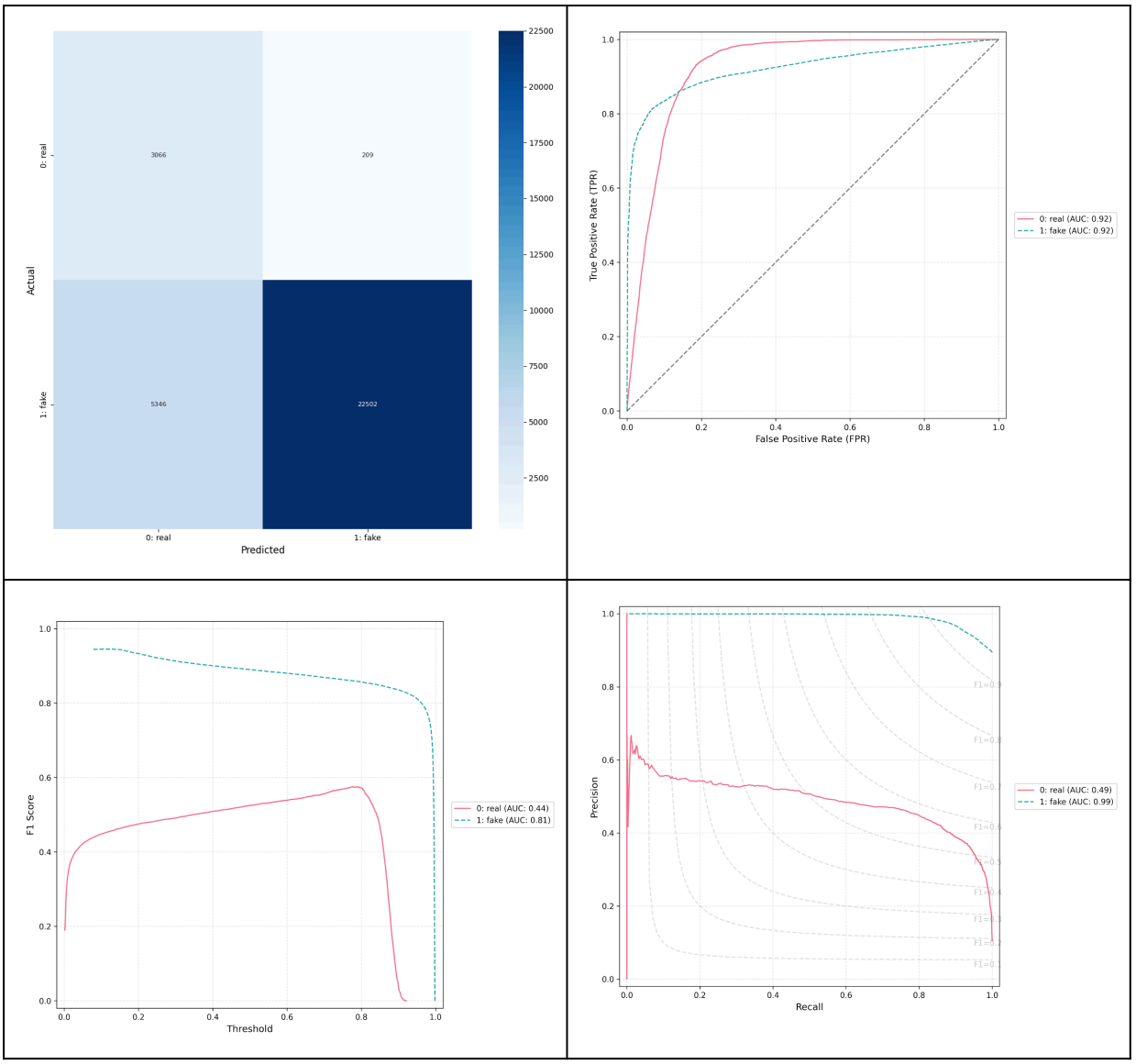




Result on UADFV Dataset

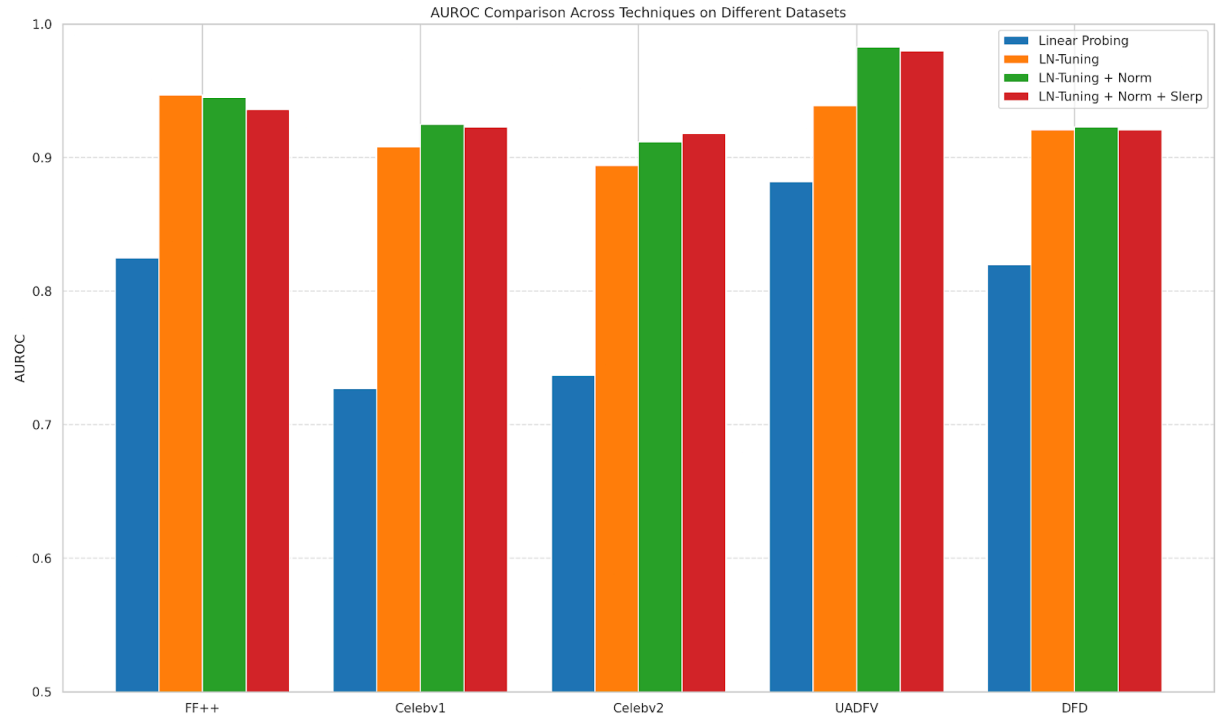


Result on DFD Dataset



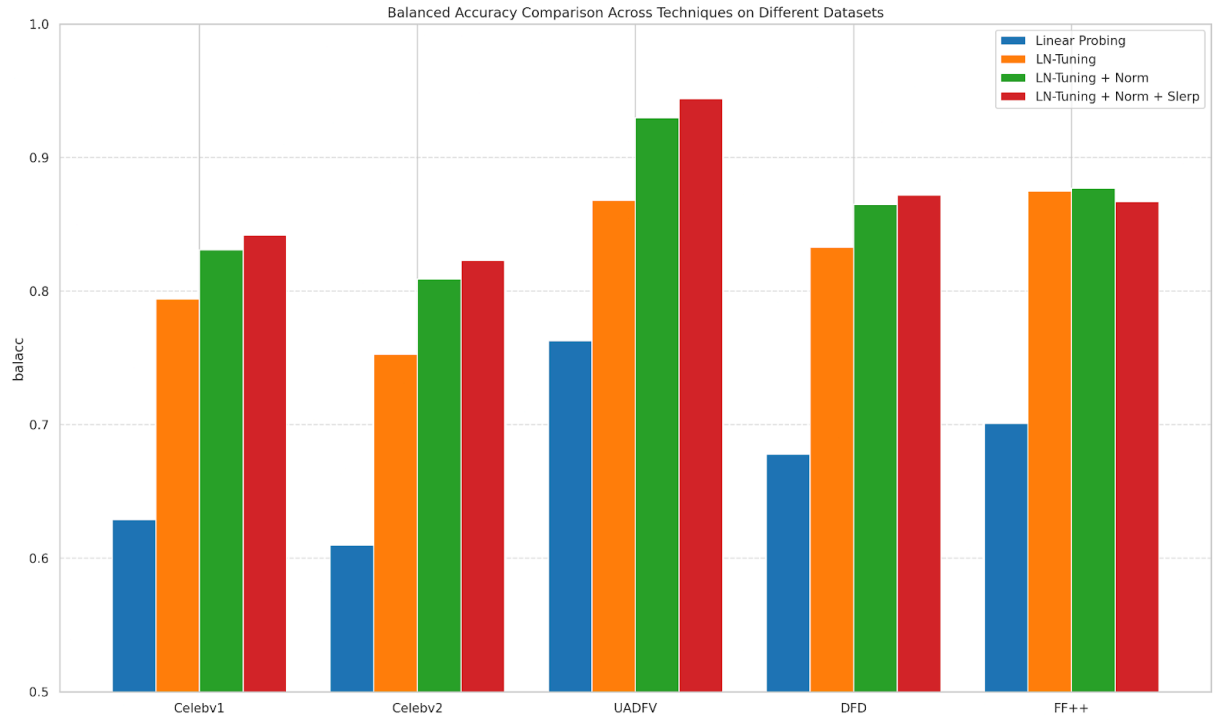
AUROC Comparison Across Techniques on Different Dataset

	FF++	Celebv1	Celebv2	UADFV	DFD
Linear Probing	0.825	0.727	0.737	0.882	0.820
LN only	0.947	0.908	0.894	0.939	0.921
LN+Norm	0.945	0.925	0.912	0.983	0.923
LN+Norm+Slerp	0.936	0.923	0.918	0.980	0.921



Balanced Accuracy Comparison Across Techniques on Different Dataset

	Celebv1	Celebv2	UADFV	DFD	FF++
Linear Probing	0.629	0.610	0.763	0.678	0.701
LN only	0.794	0.753	0.868	0.833	0.875
LN + Norm	0.831	0.809	0.930	0.865	0.877
LN +Norm +Slerp	0.842	0.823	0.944	0.872	0.867



Justification

The Slerp method (LN+Norm+Slerp) sacrifices a tiny bit of "in-domain" FF++ balanced accuracy (0.867 vs 0.877) to gain substantial improvements in **cross-dataset balanced accuracy** (e.g., 0.842 on Celebv1 vs 0.831 for LN+Norm). This confirms that Slerp's primary strength lies in **enhancing the model's ability to generalize to deepfakes generated by entirely different methods or from different sources**, even if it means a very minor dip in performance on the source dataset itself. It makes the model more robust to shifts in data distribution, which is a highly desirable trait for real-world deepfake detection.

Note - fake(label = 1) , real(label = 0)

Examples for Qualitative evaluation.

Sample Test Image taken from FF++ Dataset !

Ground Truth = 1 (fake) 	prob_real_class_0	prob_fake_class_1
Linear Probing	0.0107	0.9893
LN-Tuning	0.0048	0.9952
LN-Tuning + Norm	0.0759	0.9241
LN-Tuning + Norm + Slerp	0.0316	0.9684

Ground Truth = 0 (real) 	prob_real_class_0	prob_fake_class_1
Linear Probing	0.3789	0.6211
LN-Tuning	0.6114	0.3886

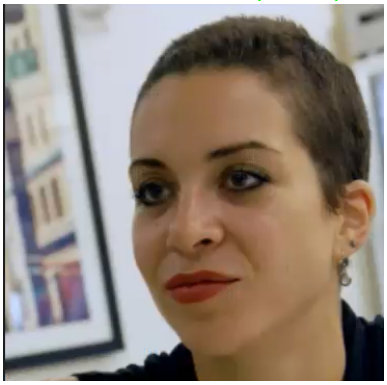
LN-Tuning + Norm	0.7937	0.2063
LN-Tuning + Norm + Slerp	0.8529	0.1471

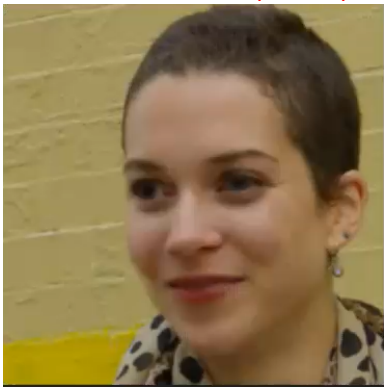
Sample Test Image taken from Celebv1 Dataset !

Ground Truth = 0 (real) 	prob_real_class_0	prob_fake_class_1
Linear Probing	0.3221	0.6779
LN-Tuning	0.6947	0.3053
LN-Tuning + Norm	0.7194	0.2806
LN-Tuning + Norm + Slerp	0.7886	0.2114

Ground Truth = 1 (fake) 	prob_real_class_0	prob_fake_class_1
Linear Probing	0.3372	0.6628
LN-Tuning	0.0532	0.9468
LN-Tuning + Norm	0.0197	0.9803
LN-Tuning + Norm + Slerp	0.0031	0.9969

Sample Test Image taken from DFD Dataset !

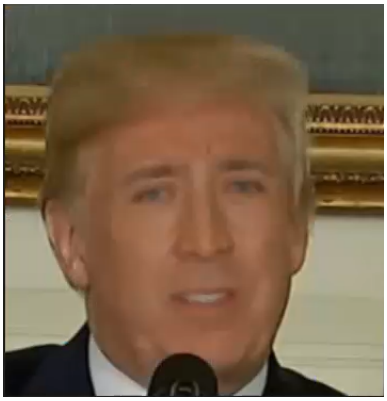
Ground Truth = 0 (real) 	prob_real_class_0	prob_fake_class_1
Linear Probing	0.6541	0.3459
LN-Tuning	0.9349	0.0651
LN-Tuning + Norm	0.7663	0.2337
LN-Tuning + Norm + Slerp	0.8273	0.1727

Ground Truth = 1 (fake) 	prob_real_class_0	prob_fake_class_1
Linear Probing	0.2237	0.7763
LN-Tuning	0.0178	0.9822
LN-Tuning + Norm	0.0138	0.9862

LN-Tuning + Norm + Slerp	0.0030	0.9970

Sample Test Image taken from UADFV Dataset !

Ground Truth = 0 (real) 	prob_real_class_0	prob_fake_class_1
Linear Probing	0.3142	0.6858
LN-Tuning	0.7033	0.2967
LN-Tuning + Norm	0.7994	0.2006
LN-Tuning + Norm + Slerp	0.8311	0.1689

Ground Truth = 1 (fake) 	prob_real_class_0	prob_fake_class_1
Linear Probing	0.0443	0.9557
LN-Tuning	0.0145	0.9855

LN-Tuning + Norm	0.0321	0.9679
LN-Tuning + Norm + Slerp	0.0077	0.9923