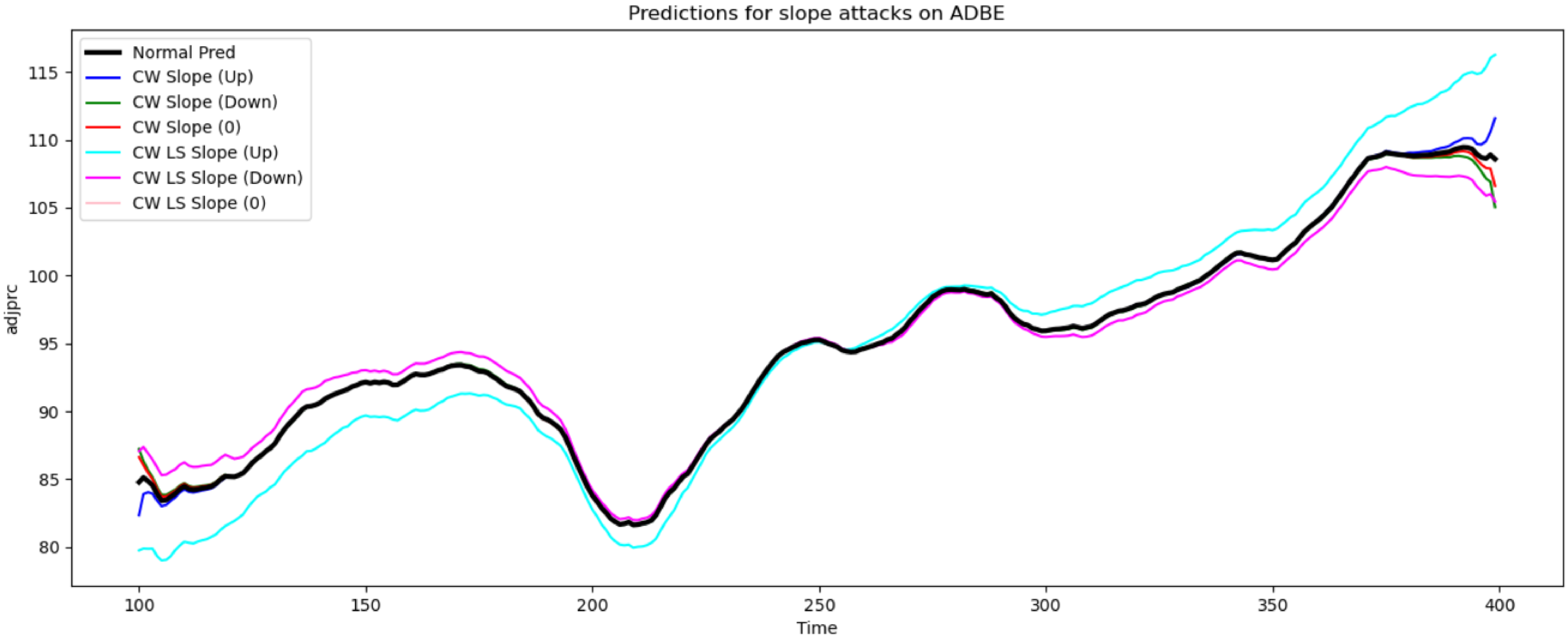
CSCD94 Week 12 Update

Dominik Luszczynski

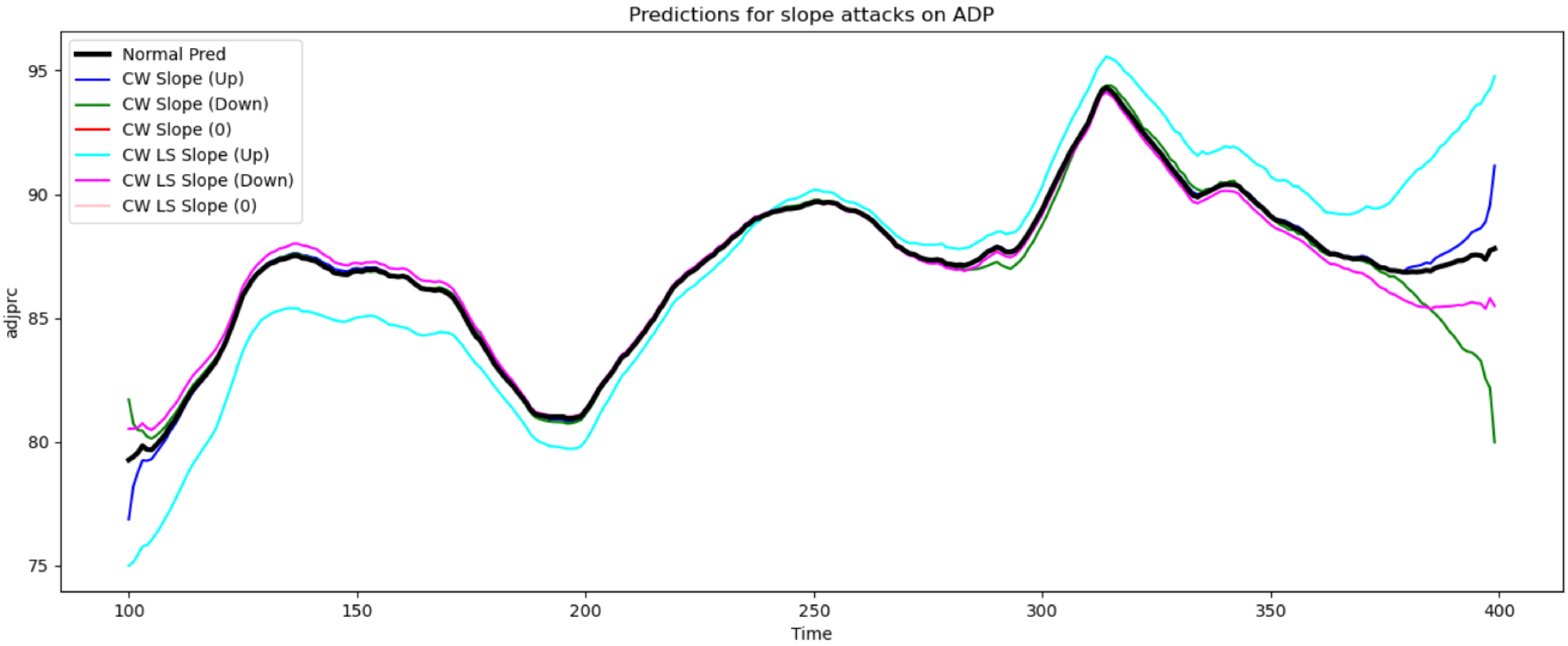
Next Steps (Last Week)

- Try to improve the GAN with the 3 solutions discussed before.
- Finish the malware.
- Implement basic adversarial training (time constraint).
- Implement C&W versions of the slope attacks.
- Start to write final report:
  - Narrative would be similar to [1] with some extra steps:
    - Discuss attacks on time series and where the pitfalls are with implementing image-based adversarial attacks on time-series.
    - Discuss how these “new” attack methods are typically performed on simple models like 3-layered CNNs, and how N-HiTS would be a better baseline since it is more production-ready.
    - Discuss new slope-based attacks.
    - Discuss the adversarial GAN (hopefully it works)
    - Discuss how easy it would be to inject malware, and how adversarial attacks need to be taken more seriously.
    - Effects of adversarial training.

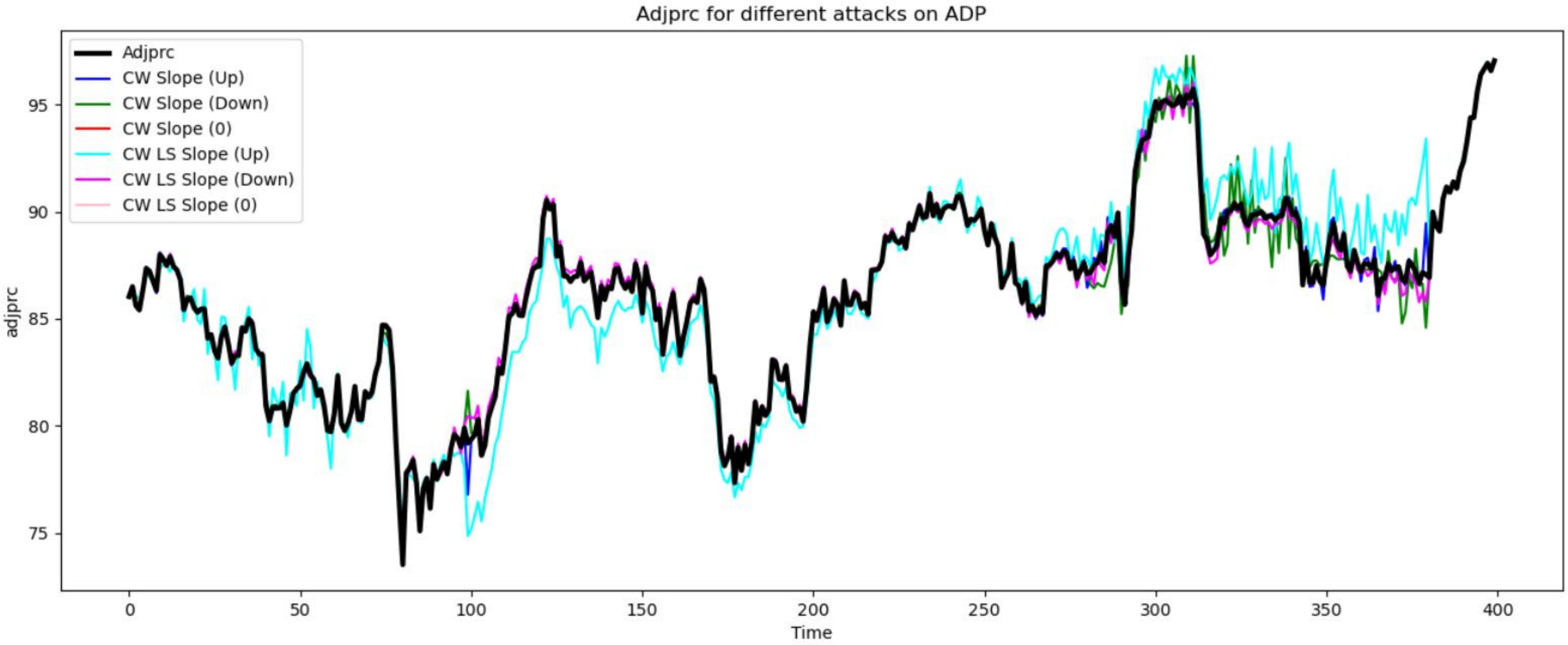
1. C&W Version of the Slope Attacks - DONE



1. C&W Version of the Slope Attacks - DONE



1. C&W Version – Easily Detectable (Expected, no clamping)



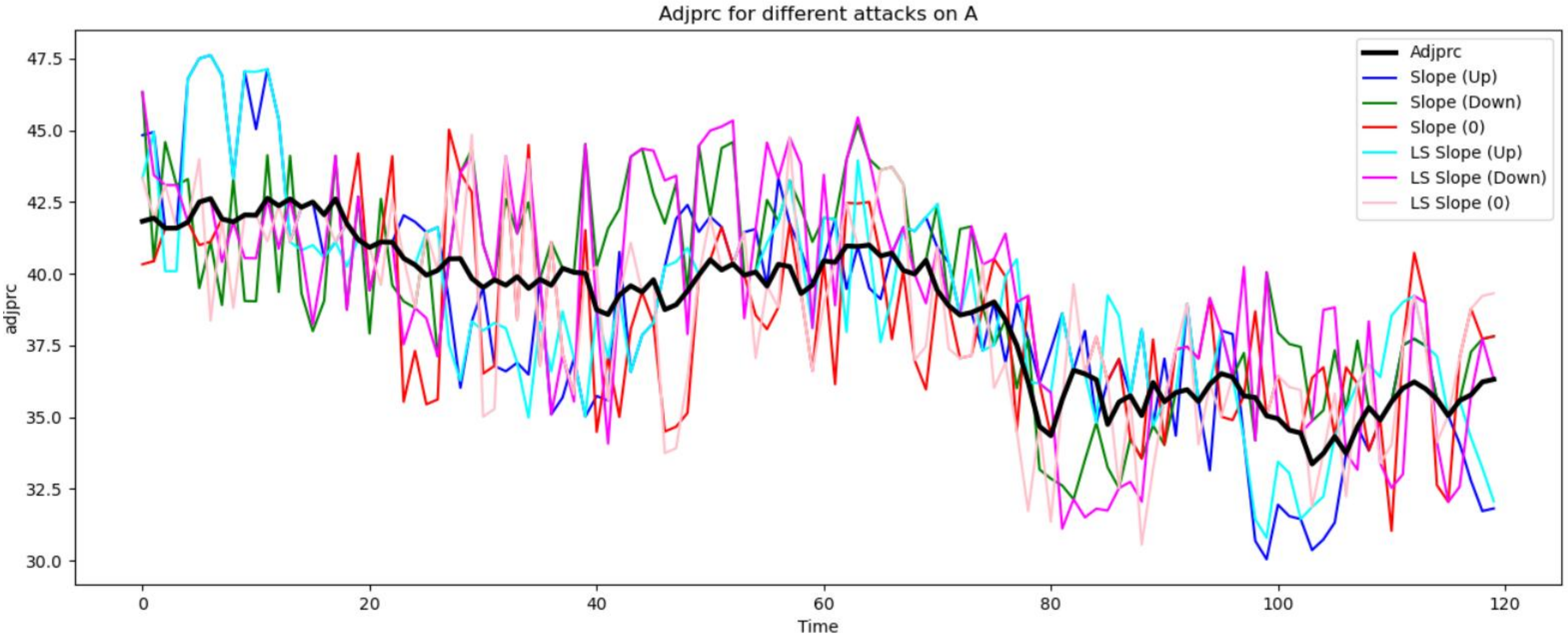
2. Adversarial Training

- Limitation: Not necessarily feasible with the N-HiTS forecasting model
  - Adversarial training: Using adversarial inputs as training data (with the real ground truth)
  - Problem: Adjprc is used as both a feature and a label. Furthermore, the N-HiTS model performs training through rolling windows.
    - Thus, I would need to generate n files with 120 days, with the first 100 days being the adversarial input, and the last 20 days being the real data (any more would cause a mix between the real and adversarial data). However, I have 350+ recordings with 10 years of data.

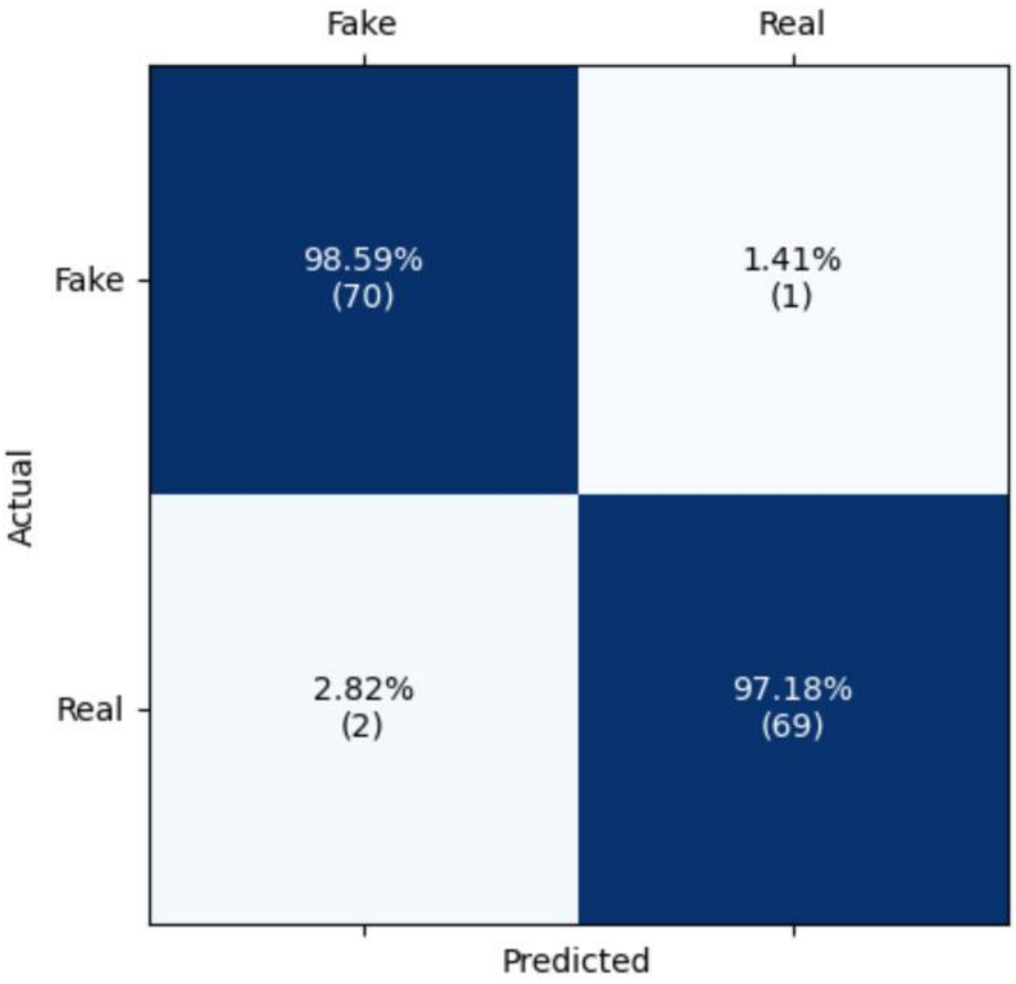
2. Change of Plans: Discriminator

- Another common approach to prevent adversarial attacks is to create a discriminator that filters inputs before sending it to the model [1].
- Therefore, I created a 4-layered CNN and trained it to differentiate real and adversarial examples.
- Three models were trained for 3 different adversarial attacks for the first 120 days of each recording (for now).
  - All models were trained on the “Up” versions of the adversarial attack.
- The same training split was used from the N-HiTS model.

2. a) Discriminator: LS-Slope Attack (Eps = 5)



2. a) Discriminator: LS-Slope Attack (Eps = 5)

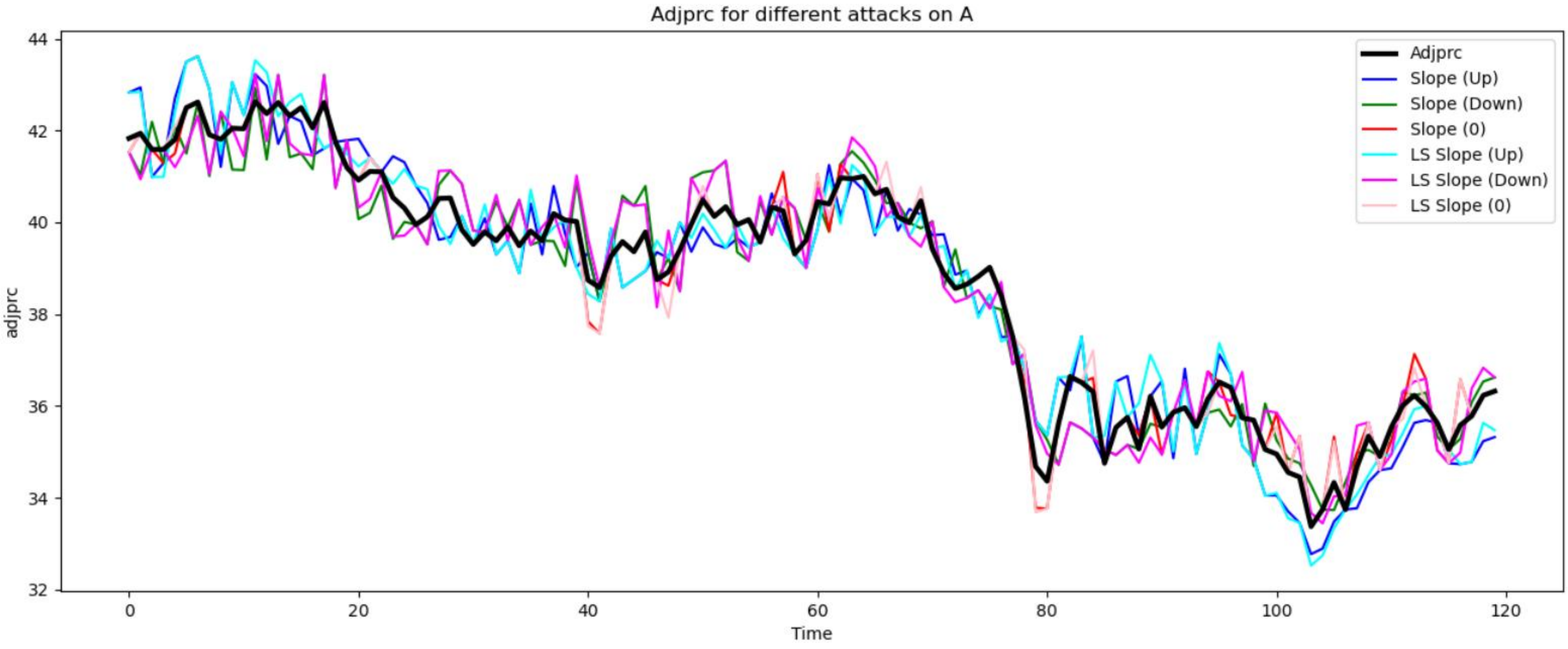


Accuracy: 97.89

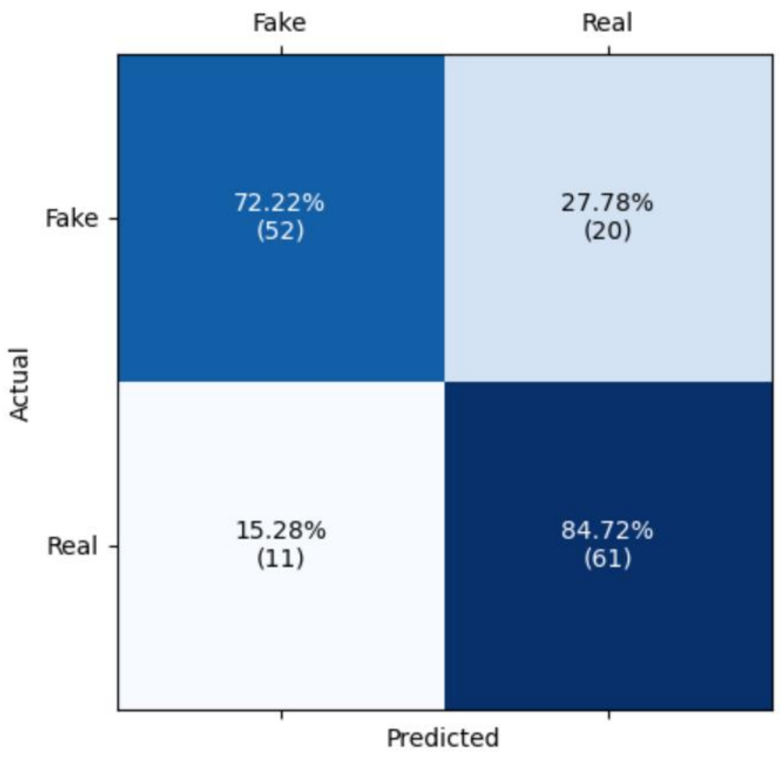
F1: 97.87

Kappa: 0.9577

2. a) Discriminator: LS-Slope Attack (Eps = 1)



2. a) Discriminator: LS-Slope Attack (Eps = 1)

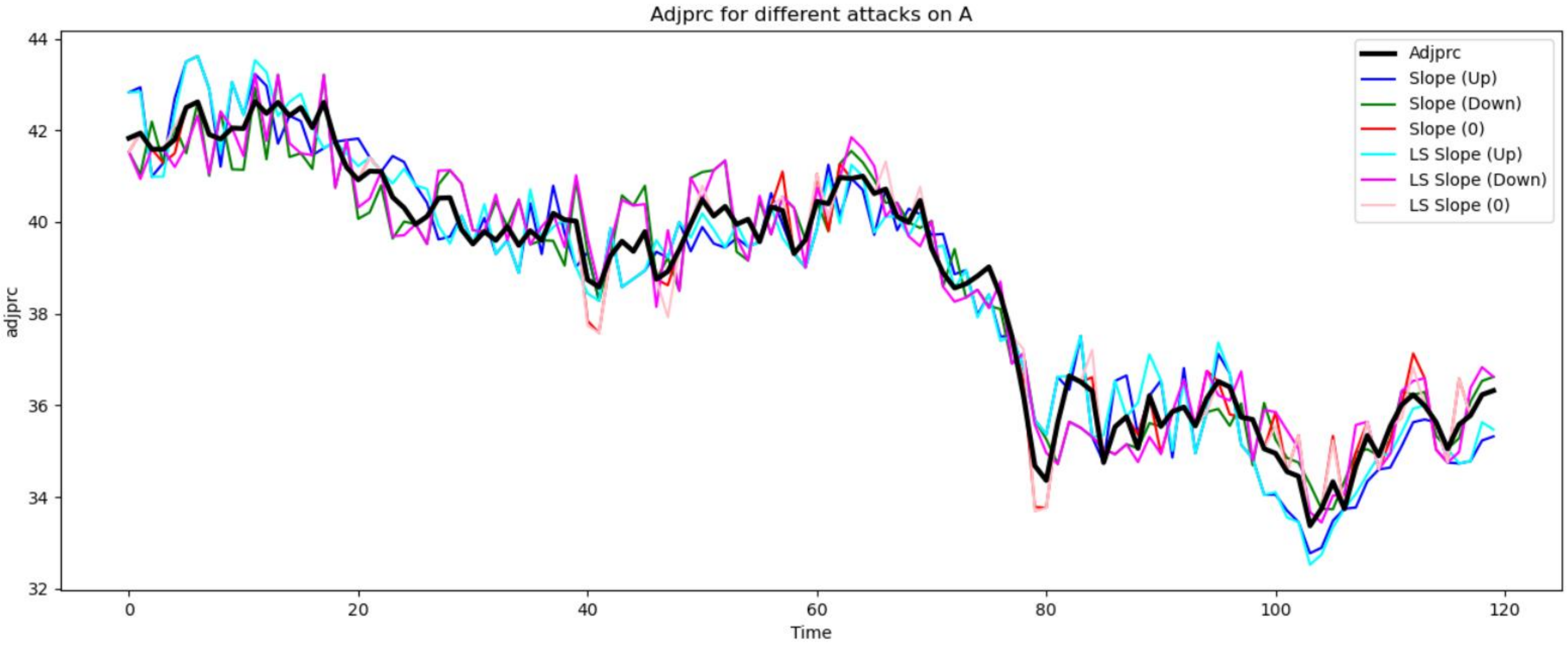


Accuracy: 78.47

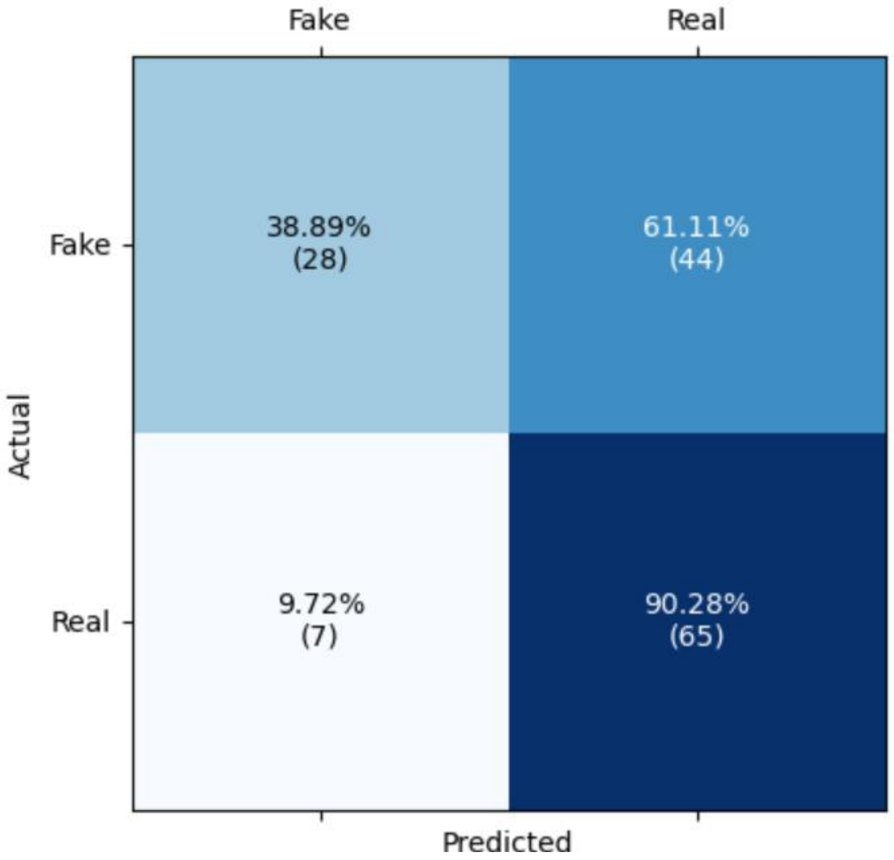
F1: 79.74

Kappa: 0.5694

2. a) Discriminator: Endpoint-Slope Attack (Eps = 1)



2. a) Discriminator: Endpoint-Slope Attack (Eps = 1)



Accuracy: 64.58

F1: 71.82

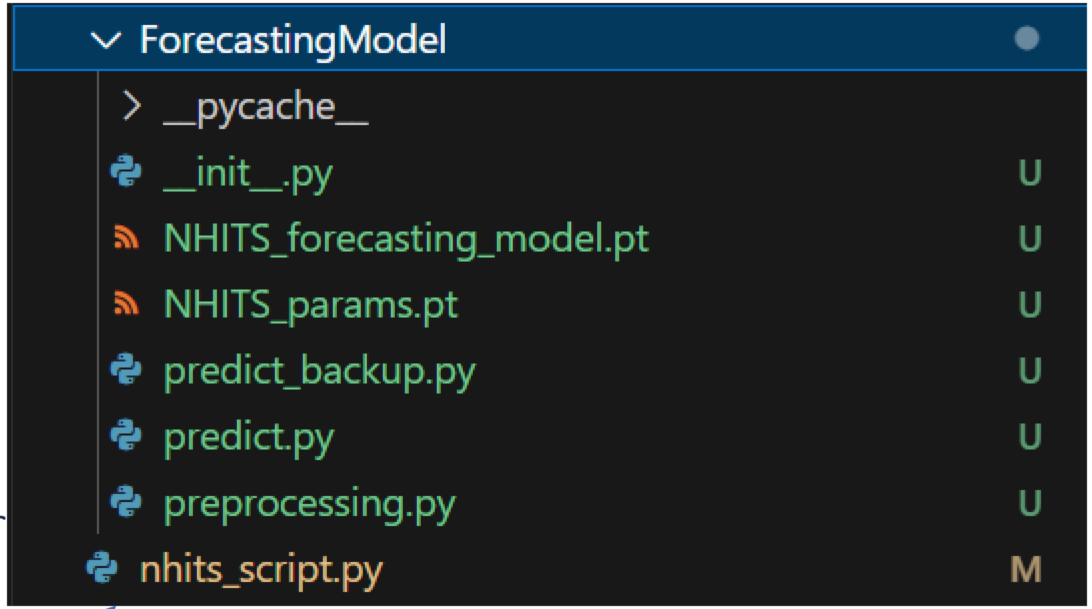
Kappa: 0.2916

Example Model Pipeline

1. Input data to predicting script
2. (Optional) Run discriminator on the input to verify it is not adversarial
3. Run and preprocessing on input
4. Run model(x)
5. Return prediction/output

3. Trojan Malware

Library



Script that uses model
(can be anywhere)

Demo

3. Trojan Malware

1. The __init__ function is run whenever you import something from the 'ForecastingModel' library.
    1. Because it is run right as you import something, we can modify any file we want inside the 'ForecastingModel' folder (these files have not been executed yet).
2. Find where we call model(x) (extra work if you have something different)
3. Read the file and save a backup (preferably a string)
4. Go through each line of the file
    1. If it is torch.no_grad line -> remove it and fix indentation on lines after
    2. If it is a 'model(x)' line -> insert our adversarial attack
    3. If normal line, keep as is
5. Write the new lines to the file
6. After execution, restore the backup

3. Trojan Malware

- Why is it problematic?
  - White/black box adversarial attacks (gradients vs no gradients) does not matter in this situation as we can easily remove any torch.no_grad().
  - We have bypassed any security the model may have (like a discriminator or any integrity checks on the model).
  - If the attack is implemented by the developer of the model, then it can be virtually undetectable unless someone explicitly reviews the code provided.
  - Don't need to touch the model itself.
  - Self-cleaning -> would need to know to inspect the __init__.py file
- Current research typically focuses on trying to secure the model (like preventing model tampering attacks [2]) or making the model robust rather than looking at the entire pipeline as a whole.

3. Trojan Malware Defenses

1. If external attacker:
   - The developer would need to hash the directory, and some installer (PyPI) would need to verify the hash matches.
   - However, if your trust a malicious installer, then there isn't anything you can do.
2. If internal attacker (developer):
   - Code review?
   - If the user uses torch.no_grad outside the corrupted library, then you could prevent gradients from being used.
     - But how likely would a non-ML developer know to do this?

4. Adversarial GAN

- Work in progress….
- Worst case, I would try to improve some of the previous GAN versions I have, run them through the adversarial discriminator to determine how “stealthy” they are.

Next Steps

- Try and get a semi-decent GAN.
- Write paper.
- Do a bit more research to make a stronger case for the malware example.

References

[1] G. Pialla *et al.*, “Time series adversarial attacks: an investigation of smooth perturbations and defense approaches,” *International journal of data science and analytics*, vol. 19, no. 1, pp. 129–139, 2025, doi: 10.1007/s41060-023-00438-0.

[2] Z. Che *et al.*, “Model Tampering Attacks Enable More Rigorous Evaluations of LLM Capabilities,” 2025, doi: 10.48550/arxiv.2502.05209.