Appendices

A Algorithm Details

We elaborate on the details of the proposed attack algorithm. Algorithm 2 is the same Algorithm 1 in the main paper. It is repeated for readability. This algorithm describes how a current perturbation δ_t is generated.

Algorithm 2 Predictive Attack at time t (Repeated for explanation).

1: $count \leftarrow 0$
2: **while** $count <$ MAX_COUNT **do**
3: $\mathcal{L}_{\text{total}}(\delta_t) \leftarrow \mathcal{L}_{\text{adv}}(x_t, y_t^a, \delta_t, h_t^\delta) + \sum_{i=t+1}^{t+K} \mathrm{E}_{Q_\phi(x_i|x_{:i-1})}[\mathcal{L}_{\text{adv}}(x_i, y_i^a, \delta_i, h_i^\delta)]$
using Monte-Carlo to compute $E_{Q_\phi}[\cdot]$.
4: $\forall i \in [t, t+K]$,
5: $\delta_i \leftarrow \Pi_{\|\delta_i\|_p \leq \epsilon}[\delta_i - \alpha \text{sign}(\nabla_{\delta_i} \mathcal{L}_{\text{total}}(\delta_t)]$
6: $\delta_i \leftarrow \text{clip}(x_i + \delta_i) - x_i$
It forces a valid range of perturbed inputs.
7: $count \leftarrow count + 1$
8: **end while**
9: **return** δ_t

Line 1, 2, 7: δ_t is computed for MAX_COUNT iterations, and $count$ is an iteration counter.
Line 3: Total adversarial loss is computed. To that end, we predict future inputs ($x_i, \forall i \in [t+1, t+K]$) based on the past and predicted observations $x_{:i-1}$ with a predictive model Q_ϕ. Furthermore, if we use a stochastic predictive model, we can use a mean of adversarial losses computed from multiple predictions (Monte-Carlo). The initial perturbations ($\delta_i, \forall i \in [t+1, t+K]$) are zero vectors.
Line 4, 5: For each i, we update δ_i based on Projected Gradient Descent. Π restricts the p-norm of δ_i to be less than ϵ.
Line 6: In addition, δ_i is needed to be in a valid input range. For instance, a range of pixels of image inputs is [0, 1] or [0, 255].

B Model Parameters

We provide detailed information about the parameters for models (f_θ, g_θ) and Q_ϕ. The model structure is shared for MNIST, FashionMNIST, and Mortality, while Udacity uses a different model for high dimensional inputs.

B.1 MNIST

Victim RNN (f_θ, g_θ)

- Input: $x_t \in \mathbb{R}^{28}, h_t \in \mathbb{R}^4$.
- Output: $\hat{y}_t \in \mathbb{R}^2$.

1. LSTM (in=28, hidden=4)
2. Linear (in=8, out=10, bias=True)
3. ReLU
4. Linear (in=10, out=2, bias=True)

Predictor RNN Q_ϕ

- Input: $x_t \in \mathbb{R}^{28}, h_t \in \mathbb{R}^{128}$.
- Output: $\hat{x}_{t+1} \in \mathbb{R}^{28}$.

1. LSTM (in=28, hidden=128)
2. Linear (in=128, out=150, bias=True)
3. Dropout (drop_probability=0.3)
4. ReLU
5. Linear (in=150, out=28, bias=True)

B.2 FashionMNIST

Victim RNN (f_θ, g_θ)

- Input: $x_t \in \mathbb{R}^{28}, h_t \in \mathbb{R}^8$.
- Output: $\hat{y}_t \in \mathbb{R}^{10}$.

1. LSTM (in=28, hidden=8)
2. Linear (in=8, out=10, bias=True)
3. ReLU
4. Linear (in=10, out=10, bias=True)

Predictor RNN Q_ϕ

- Input: $x_t \in \mathbb{R}^{28}, h_t \in \mathbb{R}^{128}$.
- Output: $\hat{x}_{t+1} \in \mathbb{R}^{28}$.

1. LSTM (in=28, hidden=128)
2. Linear (in=128, out=150, bias=True)
3. Dropout (drop_probability=0.3)
4. ReLU
5. Linear (in=150, out=28, bias=True)

B.3 Mortality

Victim RNN (f_θ, g_θ)

- Input: $x_t \in \mathbb{R}^{76}, h_t \in \mathbb{R}^{16}$.
- Output: $\hat{y}_t \in \mathbb{R}^2$.

1. LSTM (in=76, hidden=16)
2. Linear (in=16, out=10, bias=True)
3. ReLU
4. Linear (in=10, out=2, bias=True)

Predictor RNN Q_ϕ

- Input: $x_t \in \mathbb{R}^{76}, h_t \in \mathbb{R}^{128}$.
- Output: $\hat{x}_{t+1} \in \mathbb{R}^{76}$.

1. LSTM (in=76, hidden=128)
2. Linear (in=128, out=150, bias=True)
3. Dropout (drop_probability=0.3)
4. ReLU
5. Linear (in=150, out=76, bias=True)

Figure 11: Prediction performance of Q_ϕ on MNIST and FashionMNIST.

B.4 User

Victim RNN (f_θ, g_θ)

- Input: $x_t \in \mathbb{R}^3, h_t \in \mathbb{R}^{256}$.
- Output: $\hat{y}_t \in \mathbb{R}^{22}$.

1. LSTM (in=3, hidden=256)
2. Linear (in=256, out=200, bias=True)
3. ReLU
4. Linear (in=200, out=200, bias=True)
5. ReLU
6. Linear (in=200, out=200, bias=True)
7. ReLU
8. Linear (in=200, out=22, bias=True)

Predictor RNN Q_ϕ

- Input: $x_t \in \mathbb{R}^3, h_t \in \mathbb{R}^{128}$.
- Output: $\hat{x}_{t+1} \in \mathbb{R}^3$.

1. LSTM (in=3, hidden=128)
2. Linear (in=128, out=150, bias=True)
3. Dropout (drop_probability=0.3)
4. ReLU
5. Linear (in=150, out=3, bias=True)

B.5 Udacity

Victim RNN (f_θ, g_θ)

- Input: $x_t \in \mathbb{R}^{64\times3}, h_t \in \mathbb{R}^{256}$.
- Output: $\hat{y}_t \in \mathbb{R}$.

1. Conv. (in-channel=3, out-channel=16, kernel-size=16, stride=1)
2. ReLU
3. Conv. (in-channel=16, out-channel=16, kernel-size=16, stride=2)
4. ReLU
5. Conv. (in-channel=16, out-channel=2, kernel-size=8, stride=2)
6. LSTM (in=50, hidden=32)
7. Linear (in=32, out=50, bias=True)
8. ReLU
9. Linear (in=50, out=1, bias=True)

Predictor RNN Q_ϕ

Q_ϕ consists of two models: 1) Revertible Encoder, and 2) Frame Predictor. The Revertible Encoder is a revertible function that maps an input image to a feature vector. The Frame Predictor is a recurrent model that predicts a feature vector of the next time step, based on the current feature vector and a hidden state. After that, the Revertible Encoder reverts the predicted feature vector and obtains the predicted next input. Please refer to the original paper of CrevNet [Yu *et al.*, 2020] and the public implementation[4] for details.

- Input: $x_t \in \mathbb{R}^{64\times64\times3}, h_t \in \mathbb{R}^{512}$.
- Output: $\hat{x}_{t+1} \in \mathbb{R}^{64\times64\times3}$.

1. Revertible Encoder (in=$64 \times 64 \times 3$, out=512)
2. Frame Predictor (in=512, out=512)

B.6 Energy

Victim RNN (f_θ, g_θ)

- Input: $x_t \in \mathbb{R}^{27}, h_t \in \mathbb{R}^{16}$.
- Output: $\hat{y}_t \in \mathbb{R}$.

1. LSTM (in=27, hidden=16)
2. Linear (in=16, out=200, bias=True)
3. ReLU
4. Linear (in=200, out=200, bias=True)
5. ReLU
6. Linear (in=200, out=200, bias=True)
7. ReLU
8. Linear (in=200, out=1, bias=True)

Predictor RNN Q_ϕ

- Input: $x_t \in \mathbb{R}^{27}, h_t \in \mathbb{R}^{128}$.
- Output: $\hat{x}_{t+1} \in \mathbb{R}^{27}$.

1. LSTM (in=27, hidden=1024)
2. Linear (in=1024, out=150, bias=True)
3. ReLU
4. Linear (in=150, out=27, bias=True)

C Mathematical Formation of Metrics in Experiments

We provide mathematical formation of attack performance metric for each type of task.

[4] https://github.com/gnosisyuw/CrevNet-Traffic4cast.git

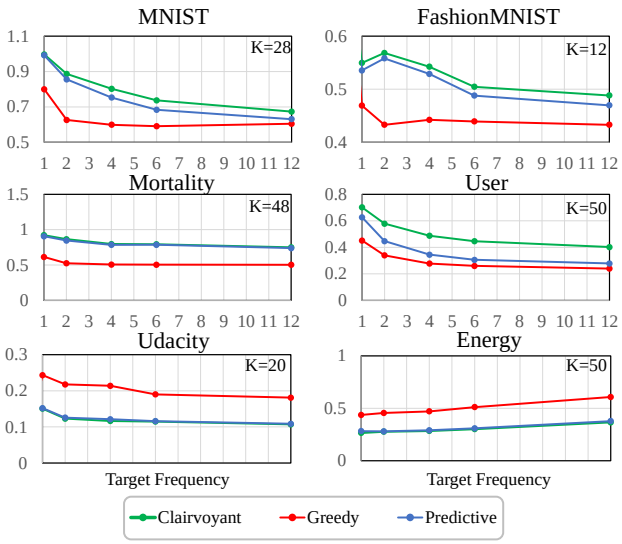

Figure 12: Attack Performances increasing the target frequency. Predictive Attack shows consistent orders of attack performances to Figure 5.

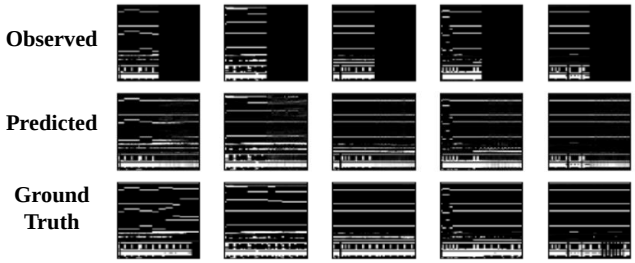

Figure 13: Prediction performance of Q_ϕ on Mortality. 76 dimensions of one-hot encoding and real-valued data, $L = 48$.

Classification Tasks. We measure Targeted Attack Success Ratio (TASR), a fraction of time steps where predicted labels are matched to target labels (y_i^a) over the number of total time steps (L). $\mathbf{1}[\cdot]$ is an indicator function.

$$\text{TASR} = \frac{\sum_{i=1}^{L} \mathbf{1}[f_\theta(x_i + \delta_i, h_i^\delta) = y_i^a]}{L}$$

Regression Tasks. We measure Targeted Mean Squared Error (TMSE) between predicted values and target values.

$$\text{TSME} = \frac{1}{L} \sum_{i=1}^{L} (f_\theta(x_i + \delta_i, h_i^\delta) - y_i^a)^2$$

D Prediction Performance

To demonstrate the performance of Q_ϕ, which is important for the Predictive Attack, we present the prediction results in Figure 11, 13, 14, 15, and 16. Given a partial early observations of a sequence $\boldsymbol{x}$, Q_ϕ predicts the rest of the sequence. For MNIST, FashionMNIST, and Mortality, we restrict the observation to the first half ($L/2$). We find Q_ϕ performs well on the datasets. Especially, Q_ϕ finds a natural extension of observed strokes of a digit. Q_ϕ also seems to learn the symmetric property of clothing. Q_ϕ finds the characteristics of Mortality: the composition of one-hot encoding (top-part) and real-valued (bottom-part) data. It produces realistic data, although some one-hot encoding is not correct due to its randomness.

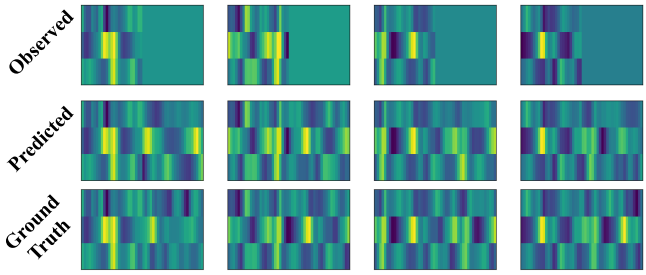

Figure 14: Prediction performance of Q_ϕ on User. 3 dimensions of real values, $L = 50$.

For Udacity, Q_ϕ observes the first five frames of road scenes and predicts the rest. We can see that Q_ϕ captures the dynamics of near vehicles and shadows.

E Adversarial Examples

To demonstrate the imperceptibility of perturbations, we present adversarial examples generated by Predictive Attack in Figure 17, 18, 19, 20, and 21. We verify that the perturbations are hard to notice, although they fool the victim RNNs, achieving 0.83, 0.80, 0.38 and 1.16 evaluation metric, respectively for MNIST, FashionMNIST, Mortality, and Udacity.

F Attack Performance with Different Targets

In order to evaluate the attack performance in more diverse targets, we increase the speed of target change. "Target frequency" refers to the speed at which the target changes. For example, the targets in Figure 4 of the main paper correspond to frequency 2. By increasing the target frequency from 1 to 12, the results are summarized in Figure 12. It is confirmed that Predictive Attack shows performance close to Clairvoyant Attack even when the target frequency is changed.

Overall, in the case of the classification task, the attack performances tend to decrease as the target frequency increases. We guess this is because the frequent target changes make the adversarial objective more difficult to achieve. On the other hand, Udacity and Energy, which are regression tasks, showed different results. We guess y ranges of each training dataset affect the results. We assume it is easier to mislead a victim to yield an observed value in training than an unobserved value. As the target frequency increases too fast to follow, misleading a victim model to yield $y = 0$ would be advantageous as it can reduce the average TMSE. However, in the case of Energy, attacks would suffer from more difficulty since the zero is not observed in the y range of the Energy training dataset. y value of Energy is an energy consumption that has only positive values, while y of Udacity is a steering angle, and it has positive and negative values crossing zero.

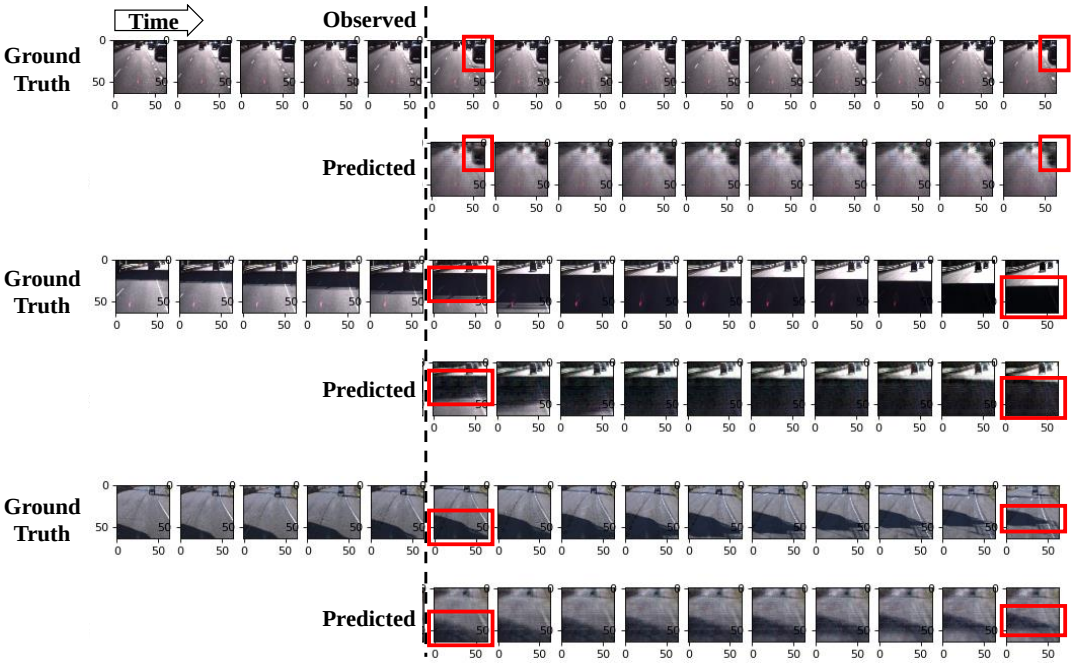

Figure 15: Prediction performance of Q_ϕ on Udacity.

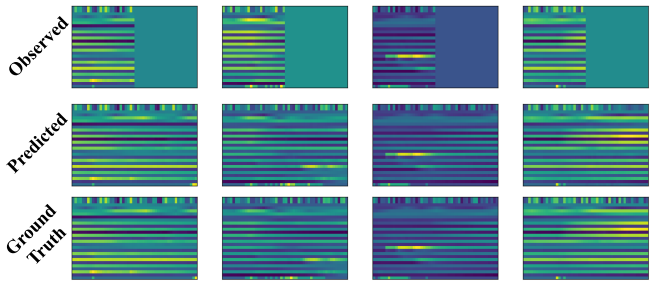

Figure 16: Prediction performance of Q_ϕ on Energy. 27 dimensions of real values, $L = 50$.

G Variability of the Achieved Results.

Figure 22 is Figure 5 of the main paper with 1 sigma performance range ($-\sigma/\sqrt{n} \sim +\sigma/\sqrt{n}, n = 3$). We exclude IID Attack to improve readability. We can check that Predictive Attack's performance ranges do not overlap with Greedy Attack's but overlap with Clairvoyant Attack's. It validates that Predictive Attack performs well in several trials consistently. Relatively, the ranges in FashionMNIST and Udacity are close compared with the other datasets. However, the actual performances of Greedy and Predictive are not close. In other words, the higher the performance of Greedy is, the higher the performance of Predictive is, and vice versa. To support that, we compute the performance correlations between Greedy and Predictive at the largest ϵ, which are 0.99 and 0.75 in FashionMNIST and Udacity, respectively.

H Additional Transferability Test

We evaluate the effectiveness of Predictive Attack in a black-box threat model, in addition to the gray-box assumption in the main paper. The attacker trains a surrogate model with a different architecture from the victim model and generates adversarial examples on the surrogate model. Then, the attacker applies the adversarial examples to a victim model.

We prepare two experiments regarding the structure of the surrogate model: 1) Different architecture in the number of the last linear layers, and 2) Different architecture in the dimension of LSTM's hidden state. We show the results on MNIST in Figure 23. We measured relative attack performance compared to the white-box performance on the surrogate model. Predictive Attack is at least 45% and 30% effective for each case, even with the different architectures.

I Impact of the Number of Sequences in Monte-Carlo Computation

To fully demonstrate the correctness of our approach (Equation 6) of Predictive Attack , we present attack performances with multi-samplings. We train new stochastic Q_ϕ that includes stochastic latent variables and, thus, produces different predictions for each sampling trial.[5] Regardless of the number of sampling trials, the attack performances are very similar. We ascribe it to the task characteristic of MNIST that is almost deterministically predictable: the next column of a digit is almost identical to the current column. We show this characteristic in Figure 25. Multiple predictions of MNIST produce similar images.

[5] Chung , J. et al. 2015, "A Recurrent Latent Variable Model for Sequential Data"

Figure 17: Adversarial examples of Predictive Attack on MNIST and FashionMNIST. ℓ_∞ $\epsilon = 0.15$.

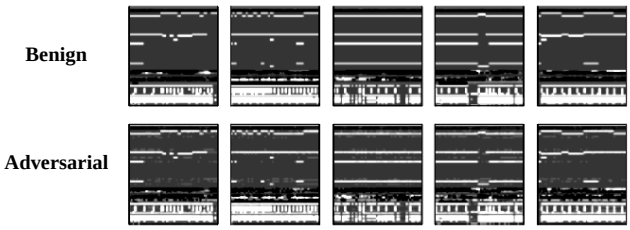

Figure 18: Adversarial examples of Predictive Attack on Mortality. ℓ_∞ $\epsilon = 0.15$.

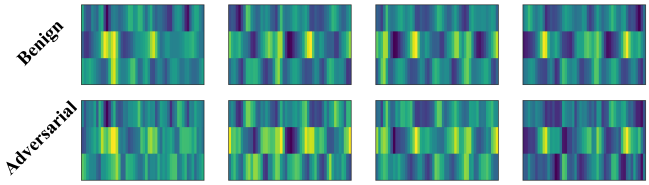

Figure 19: Adversarial examples of Predictive Attack on User. ℓ_∞ $\epsilon = 0.3$.

J Impact of MAX_COUNT.

Figure 26 illustrates speed of convergence in terms of attack performances. x-axis is MAX_COUNT, and y-axis is attack performance. We can check the gradients of Predictive Attack's attack performances, which mean speed of convergence, are close to those of Clairvoyant Attack's.

K Validation of Real-time Attack Implementation

We reimplement an imitation learning-based real-time attack [Gong *et al.*, 2019] based on the public implementation[6] because the original implementation has problem-specific constraints such as perturbation being restricted to five subsets of the entire time periods, while we consider the entire period for an attack. We replace the problem-specific expert [Storn and Price, 1997] originally used for audio data with the general expert Clairvoyant (K=L). To verify our implementation is correct, we measured the relative performance of the attack to an expert. In particular, the performance is measured at ϵ where a performance of the expert converged (Figure 27). The average relative performance of our implementation (right yellow, 57%) is higher than that of the public implementation (left straight lines, 40∼41%). This difference comes from the number of time steps where perturbations are generated. While the original implementation allowed only five perturbation steps, our implementation generates perturbations for all time steps (L).

[6]https://github.com/YuanGongND/realtime-adversarial-attack

L Online Evasion Attack on Online Training

It is meaningful to consider online training in an online evasion attack since it is frequently used to overcome the inefficiency of offline training in online tasks. Theoretically, we show online evasion attack on a victim changing with online training is the same as attacking a victim without online training. From a practical point of view, we also examine the challenge of attacking a recurrent model that changes its parameter.

Assuming the whitebox threat model as in the main paper, we assume that an attacker knows the parameter update rule $U(x_i, \theta_i) : \mathbb{R}^n \times \mathbb{R}^N \to \mathbb{R}^N$ of the victim's online training where N is the number of model parameters. In this setup, an online evasion attack can be similarly formulated by incorporating U to the original problem.

$$\boldsymbol{\delta} = \underset{\boldsymbol{\delta}=(\delta_1,\cdots,\delta_L)\in\Delta}{\arg\min} \ \mathrm{Agg}\left(\mathcal{L}_1^{\mathrm{adv}}, \cdots, \mathcal{L}_L^{\mathrm{adv}}\right), \ \text{where} \quad (10)$$

$\mathcal{L}_i^{\mathrm{adv}}$ is the loss at time i: $\mathcal{L}_i^{\mathrm{adv}} = \mathcal{L}(f(x_i + \delta_i, h_i^\delta, \theta_i), y_i^a)$, and:

$$h_i^\delta = g(x_{i-1} + \delta_{i-1},\ h_{i-1}^\delta), \quad (11)$$

$$\theta_i = U(x_{i-1} + \delta_{i-1},\ \theta_{i-1}). \quad (12)$$

From this point of view, parameter θ_i can be considered as an additional hidden state. If we denote (h_i^δ, θ_i) with $h_i^{\delta'}$ and denote g and U as g', we can rewrite the equation 2 and 3 as $h_i^{\delta'} = g'(x_{i-1} + \delta_{i-1},\ h_{i-1}^{\delta'})$. It reduces to the original problem (Equation $1 \sim 2$ of the main paper). As long as we can predict future inputs, we can compute an exact model parameter θ_i as well as hidden state h_i^δ; thus, we can conduct the proposed online evasion attack.

M Possible Defense

We propose a simple idea of using adversarial training [Mądry *et al.*, 2018], which is an effective defense against offline evasion attacks, as a defense against online evasion attacks. Firstly, we assume that the length of the input (the number of time steps) is a constant L. Under this assumption, we can consider the L-step unfolded recurrent victim model as an offline model. Therefore, as with adversarial training on an offline model, we can perform adversarial training on the unfolded victim model against L-step adversarial examples, using a PGD attack that minimizes an adversarial loss at each time step.

However, this defense would not be effective against attacks over L-step. The reason is that the adversarial error on hidden states of a victim model accumulates as an attack continues.

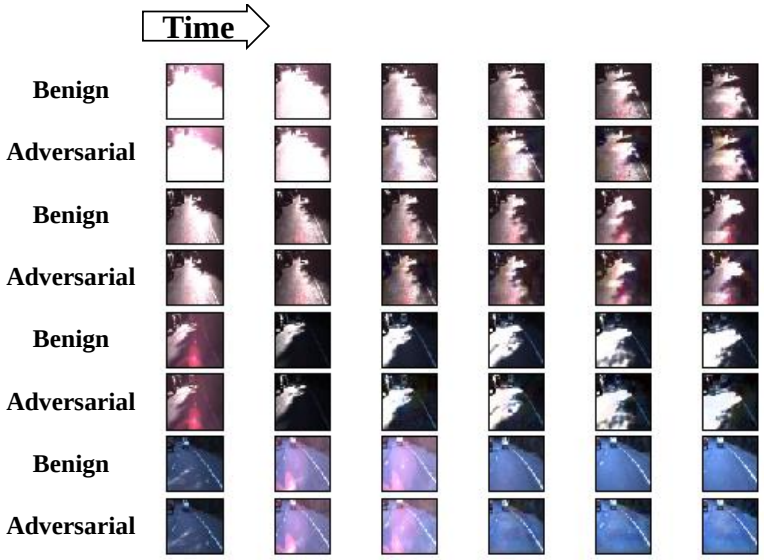

Figure 20: Adversarial examples of Predictive Attack on Udacity. ℓ_∞ $\epsilon = 0.05$.

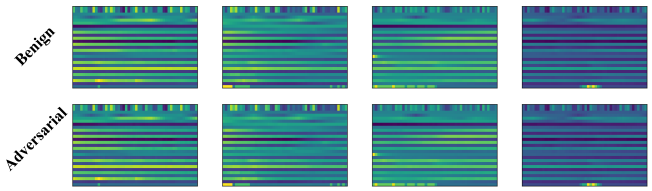

Figure 21: Adversarial examples of Predictive Attack on Energy. ℓ_∞ $\epsilon = 0.02$.

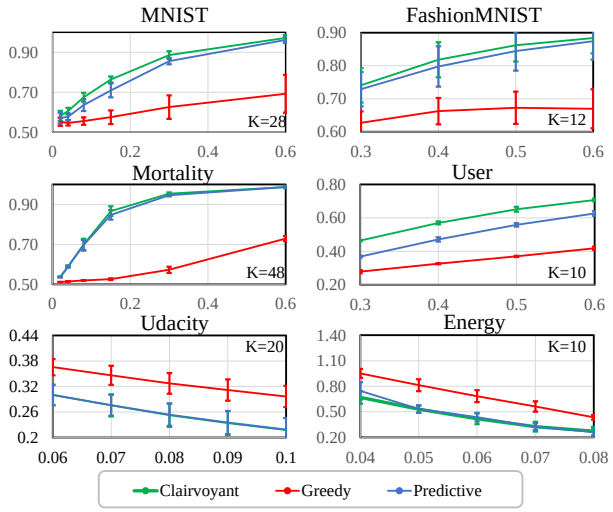

Figure 22: Variability of the achieved results (Figure 5 in the main paper). We visualize 1σ ranges of the results.

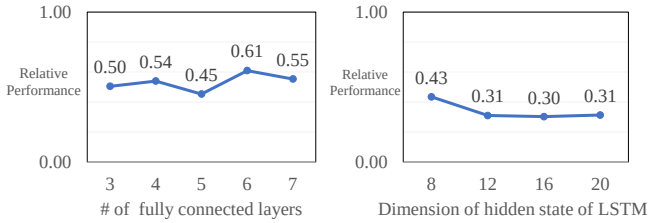

Figure 23: The relative performance of transferred adversarial examples compared to white-box attack fool rates on MNIST. Attacking a surrogate model with $\epsilon = 0.3$ constraints, Predictive Attack ($K = 28$) generated the adversarial examples. Whitebox fool rate on the surrogate model is 0.98. We find Predictive Attack can achieve 30$\sim$61% of white-box performance although the victim model has different architectures. Performances of five trials are averaged.

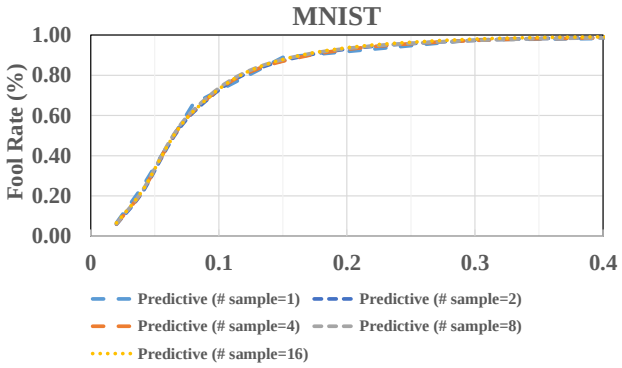

Figure 24: Impact of the number of sequences in Monte-Carlo simulation. Since there is no uncertainty in the dataset, so does its stochastic Q_ϕ. Therefore, the performance does not depend on the number of samples.

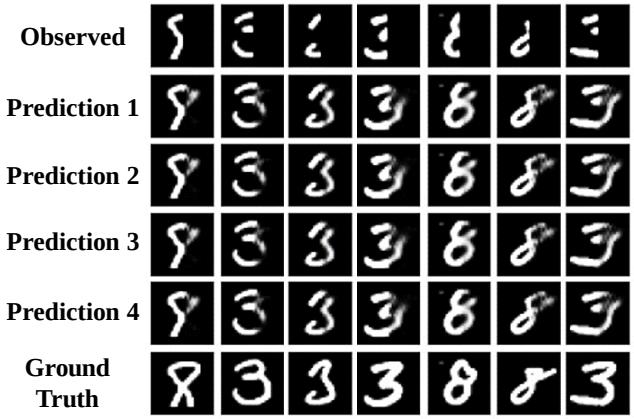

Figure 25: Prediction performance of a stochastic Q_ϕ on MNIST. We can see that multiple sampling produces similar digits.

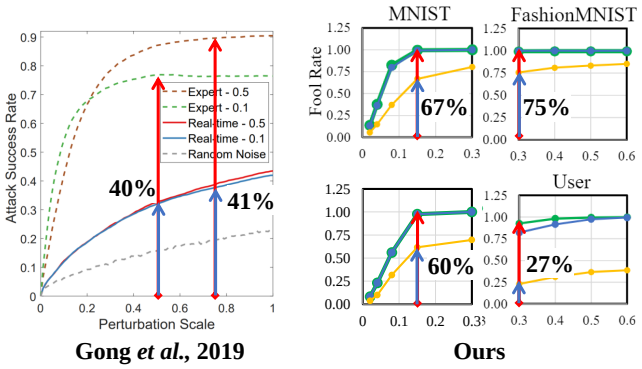

Figure 27: Implementation validation of the imitation learning-based real-time attack. We demonstrate how the relative performances of imitation learning-based attacks to that of an expert are computed. Our implementation (right, yellow) achieves higher performances, compared to the original implementation (left, straight lines), because of the larger number of perturbation steps.

The attacker would eventually succeed in manipulating hidden states to produce wrong victim outputs. A victim might initialize a hidden state to a value (e.g., zero vector) to prevent the accumulation; however, it would degrade victim's clean task performance.

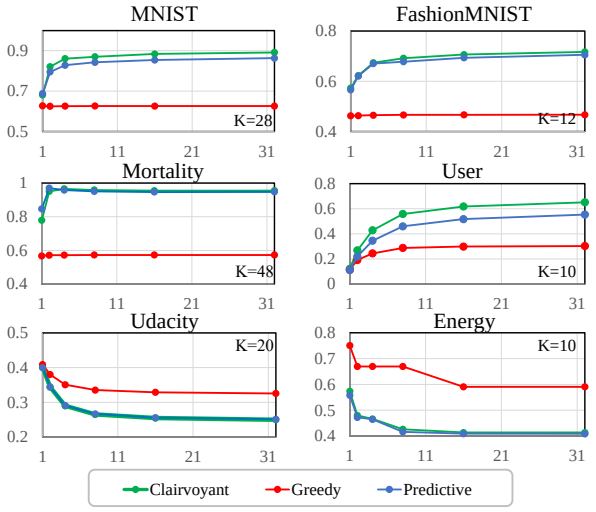

Figure 26: Impact of MAX_COUNT. Predictive Attack's attack performances converge as fast as those of Clairvoyant Attack.