

Input-Specific and Universal Adversarial Attack Generation for Spiking Neural Networks in the Spiking Domain

Spyridon Raptis and Haralampos-G. Stratigopoulos
Sorbonne Université, CNRS, LIP6, Paris, France

Abstract—As Spiking Neural Networks (SNNs) gain traction across various applications, understanding their security vulnerabilities becomes increasingly important. In this work, we focus on the adversarial attacks, which is perhaps the most concerning threat. An adversarial attack aims at finding a subtle input perturbation to fool the network’s decision-making. We propose two novel adversarial attack algorithms for SNNs: an input-specific attack that crafts adversarial samples from specific dataset inputs and a universal attack that generates a reusable patch capable of inducing misclassification across most inputs, thus offering practical feasibility for real-time deployment. The algorithms are gradient-based operating in the spiking domain proving to be effective across different evaluation metrics, such as adversarial accuracy, stealthiness, and generation time. Experimental results on two widely used neuromorphic vision datasets, NMNIST and IBM DVS Gesture, show that our proposed attacks surpass in all metrics all existing state-of-the-art methods. Additionally, we present the first demonstration of adversarial attack generation in the sound domain using the SHD dataset.

Index Terms—Spiking neural networks, neuromorphic computing, adversarial attack.

I. INTRODUCTION

Neuromorphic computing has emerged as a revolutionary paradigm, inspired by the structure and functionality of the human brain, to address the growing demand for low-power and low-latency inference in Artificial Intelligence (AI). Neuromorphic computing is based on Spiking Neural Networks (SNNs), which represent a biologically plausible model of neural computation, leveraging discrete spikes to process information. SNNs have shown significant promise in various applications, such as vision, speech processing, and robotics [1], [2].

With the growing reliance on AI systems, unique security risks and challenges emerge, underscoring the need to understand the vulnerabilities and threat landscape in neuromorphic computing as the field continues to evolve rapidly. Perhaps the most preoccupying threat is the adversarial attack [3], which aims at exploiting subtle perturbations in the input to manipulate the network’s behavior, leading to incorrect predictions. While adversarial attacks have been studied extensively for Artificial Neural Networks (ANNs) [4], transferring

these attacks to SNNs presents unique challenges due to their discrete and temporal characteristics.

Adversarial attacks on SNNs can be classified in various ways. They presume a real-valued input source, i.e., image frames, which is then encoded into spike trains [5]–[8], or they operate directly on spiking datasets, for example extracted by a Dynamic Vision Sensor (DVS) [9]–[12]. Although spike encoding bridges the gap between conventional data and SNNs, SNNs are inherently designed to work with spiking input datasets because such datasets align naturally with their event-driven, time-based computational paradigm. This is where SNNs showcase their advantages, as opposed to converting static frames to spiking data, which results in energy cost, loss of temporal richness, and reduced biological fidelity. Therefore, techniques designed for spiking input sources [9]–[12] are necessary to accommodate the most efficient and natural application of SNNs.

Another categorization of existing techniques is trial-and-error [6], [9], [10] versus gradient-based [7], [8], [10]–[12]. Trial-and-error refers to an iterative process where the input sample is repeatedly modified until the desired adversarial behavior is achieved. This approach is not effective because it is difficult to find the optimal perturbation in the limited search time that the attacker has.

Gradient-based approaches rely on the model’s internal gradients to quickly determine the optimal perturbation. When applied to SNNs [10]–[12], each iteration of the algorithm is composed of three stages: (i) forward pass to obtain the model prediction and compute the loss; (ii) backward pass to compute the gradient of the loss with respect to the input of the first layer; and (iii) update of the spiking input perturbation. The backward pass is performed in the spiking domain using surrogate gradients to address the non-differentiability of the spiking activation function [13], [14]. The proposed techniques mainly differ in step (iii), in particular how the continuous gradients at the input of the first layer, which are typically with respect to the membrane potential of the neurons, are converted to perturbations in the spiking domain, with the challenge being to maintain the spatiotemporal gradient information during the conversion.

A third categorization is input-specific versus universal adversarial attacks. Input-specific attacks generate perturbations that are specifically crafted for each individual input. In contrast, universal attacks generate a single perturbation that

This work was supported by the French National Research Agency (ANR) through the European CHIST-ERA program under the project TruBrain (Grant N° ANR-23-CHR4-0004-01) and by the European Network of Excellence dAIEDGE (Grant N° 101120726).

can fool the model across many inputs. Input-specific attacks are less plausible as it is assumed that the attacker can capture the running input, generate the adversarial version offline, and modify the input, all in real-time without introducing any delay. Universal attacks are more practical and efficient in real-world scenarios. The vast majority of works propose input-specific attacks [5]–[12] with only one work addressing the universal attack problem [11].

Metrics used to evaluate the efficiency of adversarial attacks include: (i) adversarial accuracy or attack success rate (ASR) defined as the percentage of adversarial examples that successfully cause the model to produce an incorrect answer; (ii) perturbation size defined as the percentage of input spikes across the complete input duration that are flipped; and (iii) average elapsed time to generate the adversarial example.

In this work we make the following contributions:

- 1) We propose novel gradient-based input-specific and universal adversarial attacks that operate exclusively in the spiking domain, effectively preserving the spatiotemporal information within the gradients.
- 2) Our input-specific attack outperforms all prior state-of-the-art gradient-based attacks in the spike domain in all metrics [10]–[12].
- 3) Our universal attack outperforms the only existing approach [11] as it is more successful across the entire dataset and is inherently more stealthy.
- 4) While previous works focus on vision datasets [10]–[12], such as the NMIST [15] and IBM DVS Gesture [16], we demonstrate for the first time adversarial attacks on the Spiking Heidelberg Digits (SHD) sound dataset [17].

The rest of the article is structured as follows. In Section II, we review the state-of-the-art on adversarial attacks on SNNs following the aforementioned categorization. In Section III, we discuss the generation of spiking datasets. In Section IV, we describe the proposed adversarial attacks generation. In Section V, we describe the experimental setup and, in Section VI, we present the results. Section VII concludes the paper by pointing to future work ideas.

II. STATE-OF-THE-ART ADVERSARIAL ATTACKS ON SNNs

A. Input-specific adversarial attacks

1) *Real-valued input*: In [5], an ANN model with the same topology as the SNN is randomly initialized and its weights are overwritten with the weights of the SNN. Next, the real-valued input is converted into the spiking domain using Poisson rate encoding and converted back to a real-valued rate input by averaging the spikes for each pixel over the sample duration. The classical Fast Gradient Sign Method (FGSM) [18] is used to generate an adversarial input for the ANN starting from the rate input. The ANN adversarial input is then converted to the adversarial SNN input using the same Poisson rate coding.

In [6], the algorithm follows a trial-and-error approach applying a perturbation in a subset of pixels in the middle of the image that is imperceptible to the human eye so as to maximize the difference between the target class probability and the maximum class probability considering all other classes.

In [7], the classical adversarial generation algorithms FGSM [18] and Projected Gradient Descent (PGD) [19] proposed for ANNs are adapted in the spiking domain. A similar approach is used in [8] using PGD, but the focus was to study how the spiking structural parameters, such as neuron threshold and inference window, can affect the adversarial robustness.

2) *Spiking input*: In [9], an adversarial input is generated by adding or removing spikes in a trial-and-error fashion, under the constraint that the number of perturbed spikes stays confined within a fraction ϵ of the number of input spikes.

In [10], both trial-and-error and gradient-based attacks are proposed for vision datasets generated by a DVS. Trial-and-error attacks perturb spikes in the perimeter of the frame (*frame* attack), in the corners (*corner* attack) or in two pixels at a time (*dash* attack) for the whole duration of the sample. The gradient-based attack, called *sparse* attack, works by updating the input perturbation based on the gradient of the loss with respect to the input.

In [11], *SpikeFool* is proposed, which is an adapted version of the *SparseFool* attack [20] in the spiking domain. *SparseFool* finds a small perturbation on the input in the direction orthogonal to the decision boundary.

In [12], a gradient-based attack is proposed specific to SNNs that leverages the spatiotemporal backpropagation SNN training [13]. In the backward pass, the real-valued gradients with respect to the membrane potential of the input neurons is generated. Then, a gradient-to-spike (G2S) converter using a series of mathematical operations is proposed to convert the first-layer input gradient map to input spiking train updates in each iteration. For some inputs, the gradient vanishing problem was encountered, i.e., the resultant gradient map had all zeros. For such inputs, to circumvent this issue, the restricted spike flipper (RSF) converter is proposed that re-initializes the gradient map and is combined with increasing the firing rate of neurons in the penultimate layer during the attack.

B. Universal adversarial attacks

In [11], the universal patch is restricted in the area of the input that is key for correct prediction. For example, in the case of a vision input, it is restricted in the area where the action is performed. The proposed algorithm is adapted from [21]. The patch is optimized iteratively on the inputs using the PGD gradient-based method to maximize the misclassification rate across inputs.

III. SPIKING DATASETS

Spiking inputs are binary signals that represent discrete events over time, mimicking the way biological neurons communicate with one another.

In vision tasks [15], [16], these inputs are typically generated by a DVS, which captures changes in pixel brightness rather than absolute brightness [22]. Each event encodes the location of a pixel, a timestamp, and a polarity indicating whether the brightness increased or decreased. This event-driven approach produces sparse and temporally precise data,

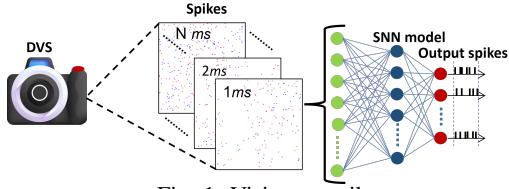


Fig. 1: Vision to spikes.

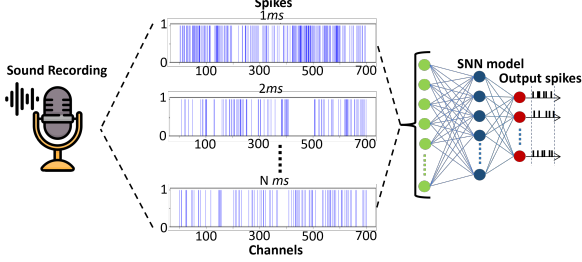


Fig. 2: Sound to spikes.

making it ideal for low-power and high-speed applications. Fig. 1 shows the spike events produced from a DVS and fed to an SNN model. Red and blue dots represent the positive and negative polarity of the events accordingly.

In sound tasks [17], spiking inputs are derived from recording devices that convert continuous auditory signals into spikes. This is often achieved using a bank of filters or channels that decompose the audio signal into different frequency bands, similar to the cochlea in the human ear [17]. Each channel generates spikes when the energy in its frequency band exceeds a certain threshold, preserving both the temporal and spectral structure of the auditory signal. Fig. 2 shows the spike events produced from a sound recording device that has 700 channels.

These spiking representations enable SNNs to process temporal data efficiently while aligning with the asynchronous nature of neuromorphic hardware [1], [2].

IV. PROPOSED ADVERSARIAL ATTACKS ON SNNs

As SNNs operate with binary inputs (spike or no spike), classic gradient-based techniques cannot be utilized. Hence, we designed a strategy that allows for direct gradient-based optimization exclusively in the spiking domain. The techniques used to bridge the gap between continuous gradient updates and binary spiking inputs are the Gumbel-Softmax [23], [24] and the Straight-Through-Estimator (STE) [25]. The proposed algorithms make no assumption about the architecture of the SNN, i.e., fully connected, convolutional or recurrent, and no assumption about the information coding scheme, i.e., rate coding or time-to-first-spike coding.

Input optimization in the spiking domain has also been explored in the contexts of SNN hardware accelerator testing [26] and hardware Trojan attacks [27], each using tailored real-to-binary tensor conversions and task-specific loss functions. In the first scenario, the goal is to generate a minimal-duration input sequence that maximizes fault coverage, whereas in the second, the objective is to design a trigger input capable of activating the hardware Trojan.

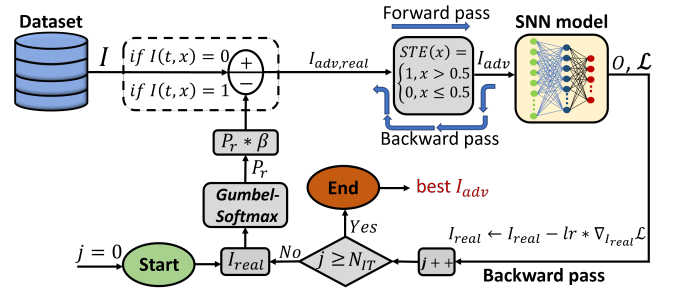


Fig. 3: Input-specific adversarial attack generation flow.

A. Terminology and notation

We interpret the input as being decomposed into frames of size $W \times H$ over time and we denote it by $I(t_k, x_{ij})$, where t_k denotes a discrete time point and x_{ij} denotes the spatial location on the frame, $i = 1, \dots, W$ and $j = 1, \dots, H$. If the input has duration T and T_f denotes the global clock period, then $k = 1, \dots, T/T_f$. At time point t_k , $I(t_k, x_{ij}) = 1$ if pixel/channel x_{ij} carries a spike, otherwise $I(t_k, x_{ij}) = 0$ denotes no spike.

Let the SNN have L layers with N^ℓ neurons in layer ℓ , $\ell = 1, \dots, L$, with N^L being the number of output classes.

Let $O^{\ell i}(I)$ denote the output of neuron i in layer ℓ for input I and let $O^L = [O^{L1}, \dots, O^{LN^L}]$ denote the output of the last layer L . The winning class neuron of the output layer L is the one that produces the largest number of spikes within a time window:

$$w = \arg \max_i \|O^{Li}(I)\|_1, \quad (1)$$

where $\|\cdot\|_1$ is the ℓ^1 norm and $\|O^{Lw}(I)\|_1$ is the spike count of the winning class neuron w . For a given dataset sample I , the goal of the adversarial attack is to find a slightly perturbed sample I_{adv} such that a different neuron wins:

$$\arg \max_i \|O^{Li}(I)\|_1 \neq \arg \max_i \|O^{Li}(I_{adv})\|_1. \quad (2)$$

Defining the variable:

$$y(k, i, j) = |I(t_k, x_{ij}) - I_{adv}(t_k, x_{ij})|, \quad (3)$$

the perturbation size, denoted by $PS(I, I_{adv})$, is given in % by:

$$PS(I, I_{adv}) = 100 \cdot \frac{\sum_{k=1}^{T/T_f} \sum_{i=1}^W \sum_{j=1}^H y(k, i, j)}{W \times H \times T}. \quad (4)$$

B. Input-specific adversarial attack

1) *Algorithm:* Fig. 3 shows the flowchart of the input-specific adversarial attack algorithm. We start with a randomly initialized real-valued tensor I_{real} of the same size as I . I_{real} is converted into a probabilistic tensor Pr with values in the range $(0, 1)$ using the Gumbel-Softmax function:

$$Pr = \text{GumbelSoftmax}(I_{real}, \tau), \quad (5)$$

where the temperature parameter τ controls the sharpness of the probability distribution. A high temperature produces a softer binary distribution where probabilities are closer to each other, while a lower temperature produces a sharper distribution where probabilities are closer to 0 or 1.

We generate a real-valued perturbed representation $I_{adv,real}$ of I as follows:

$$I_{adv,real}(t, x) = \begin{cases} I(t, x) + Pr(t, x) * \beta & \text{if } I(t, x) = 0 \\ I(t, x) - Pr(t, x) * \beta & \text{if } I(t, x) = 1 \end{cases} \quad (6)$$

where β is a scaling factor controlling the level of perturbation and is initially set to 1.

Next, the STE function is applied that transforms $I_{adv,real}$ using a threshold 0.5 into the adversarial input I_{adv} :

$$I_{adv} = \text{STE}(I_{adv,real}). \quad (7)$$

Combining Eqs. (6) and (7), we note that Pr represents the probability of each event being perturbed. It is added to the elements of I that are 0 and subtracted from the elements of I that are 1. For $\beta = 1$, the event is perturbed if $Pr > 0.5$, i.e., a spike is added if $I(t, x) = 0$ and a spike is removed if $I(t, x) = 1$.

Now, the attack can be formulated as an optimization problem:

$$\min_{I_{real}} \mathcal{L}(I, I_{adv}, O^L), \quad (8)$$

where the minimization of the loss function \mathcal{L} achieves the desired adversarial objective. The loss function \mathcal{L} is defined as the weighted sum of three loss functions L_i , $i = 1, \dots, 3$ that will be described in detail in Section IV-B2:

$$\mathcal{L} = \sum_{k=1}^3 \alpha_k * L_k, \quad (9)$$

where α_i are the weights for scalarizing the three loss functions and aggregating them into a single one. An optimizer (e.g. Adam [28]) is set to apply changes to I_{real} with adaptive learning rate lr towards achieving the optimization objective.

Each iteration of the algorithm involves two stages. In the first stage, forward pass is performed with I_{adv} and the loss function \mathcal{L} is computed. In the second stage, gradient-based backpropagation is performed to compute the gradient of the loss with respect to the membrane potential of neurons in the input layer using the same backpropagation pipeline as during the training of the SNN. In our implementation, we used the SLAYER framework [14] and did not notice the extreme gradient vanishing problem as in [12]. When, we reach the input layer, the STE function passes on the incoming gradient as if it was an identity function. Essentially, the STE function generates the spiking input in the forward pass while allowing the necessary continuous gradient flow during backpropagation. Thereafter, the gradients are computed normally because all tensors are real-valued and differentiable. Using the chain rule, we obtain:

$$\nabla_{I_{real}} \mathcal{L} = \nabla_{I_{adv,real}} \mathcal{L} \cdot \frac{\partial I_{adv,real}}{\partial Pr} \cdot \frac{\partial Pr}{\partial I_{real}}, \quad (10)$$

where $\frac{\partial A}{\partial B}$ represents the Jacobian of the transformation from A to B . Finally, I_{real} is updated through the optimizer as follows:

$$I_{real}^{(l+1)} \leftarrow I_{real}^{(l)} - lr * \nabla_{I_{real}} \mathcal{L}, \quad (11)$$

where $I_{real}^{(l+1)}$ is the optimized I_{real} after iteration l of the optimization loop. The I_{real} is being continuously optimized

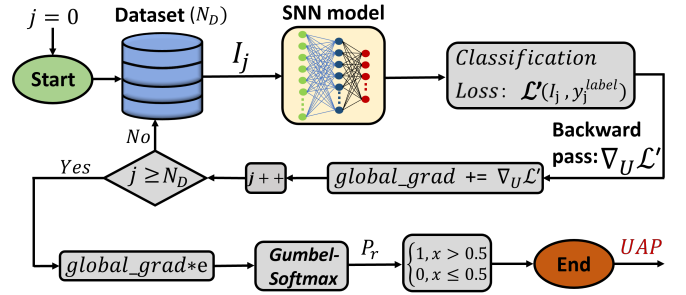


Fig. 4: Universal adversarial attack generation flow.

such that it leads to the ideal P_r and eventually to the optimal I_{adv} . The loop terminates when a specific number of iterations N_{IT} is reached and the best I_{adv} that produced the minimum loss is returned, as well as the elapsed time for its generation.

2) *Loss functions*: The three loss functions are as follows:

- *Spatiotemporal similarity loss* (L_1): Targets minimizing the spatiotemporal similarity between the original and perturbed samples. For this purpose, it uses the sample variance of the variable $y(k, i, j)$ in Eq. (3) denoted by $\text{Var}(y)$:

$$L_1 = \max(0, \text{Var}(y) - r_1). \quad (12)$$

Relaxation parameter r_1 is adaptive starting from 0 and increasing every few iterations if there is no improvement in the loss. We use the sample variance instead of the perturbation size in Eq. (4) as we experimentally found that the optimization converges faster.

- *Winning class loss* (L_2): Let w denote the winning class neuron when forward passing sample I , i.e., $w = \arg \max_i \|O^{Li}(I)\|_1$. This loss function measures the spike counts of this winning neuron when forward passing the adversarial sample I_{adv} :

$$L_2 = \|O^{Lw}(I_{adv})\|_1. \quad (13)$$

The idea behind this loss function lies in the fact that if the spikes of the originally winning class are gradually reduced, then eventually the winning class will change achieving the adversarial objective.

- *Confidence margin loss* (L_3): At every iteration, I_{adv} is optimized to force the network into misclassifying it. When the winning class changes from w to w_{adv} , this loss function ensures that there is a minimum spike count difference d between the original and new winning classes:

$$L_3 = \max(0, \|O^{Lw}(I_{adv})\|_1 - \|O^{Lw_{adv}}(I_{adv})\|_1 + d). \quad (14)$$

The direct gradient application and adaptive hyperparameters (e.g., lr , τ , β , r_1 , d) make the optimization to converge quickly and always lead to the generation of an adversarial example.

C. Universal Adversarial Attack

Fig. 4 shows the flowchart of the proposed universal adversarial attack generation algorithm. The steps are as follows:

- 1) The dataset of size N_D used to train and validate the SNN model is being loaded and the model is set again to training mode.
- 2) A global tensor $global_grad$ is defined having the same size as input I . $global_grad$ is initialized with zeros:

$$global_grad = 0. \quad (15)$$

The goal is to progressively update this tensor with the gradient information gathered from the dataset samples.

- 3) Every sample I_j from the dataset is forward passed through the network in order to calculate the classification loss $\mathcal{L}'(I_j, y_j^{label})$, where y_j^{label} is the target class of I_j .
- 4) Backpropagation is performed to obtain the gradients of the loss with respect to the membrane potential of the neurons of the first layer, denoted by U . These gradients, denoted by $\nabla_U \mathcal{L}'$, indicate the direction and the magnitude in which the membrane potential of the neurons should change to either increase or decrease the likelihood of a correct classification. By extension, the membrane potential is defined by the number and rate of incoming spikes, thus $\nabla_U \mathcal{L}'$ indirectly indicates the change in the input spike train.
- 5) During the inference, the gradients are accumulated in the global gradient tensor $global_grad$:

$$global_grad += \nabla_U \mathcal{L}'. \quad (16)$$

The purpose of accumulating the gradients is to capture global patterns of sensitivity. Certain input neurons will consistently show higher gradient magnitude across multiple samples, which suggests that they are critical to the decision making. These critical input neurons are the primary target for perturbing their incoming spike trains.

- 6) After the inference over the complete dataset is finished, the Gumbel-Softmax function is applied to convert the real-value global gradient tensor into the range $(0, 1)$:

$$Pr = \text{GumbelSoftmax}(global_grad \times e, \tau), \quad (17)$$

where e controls the perturbation magnitude and initially is set to 1. Pr here represents the probabilities of input spikes being perturbed. Values closer to 1 mean that the corresponding spike is highly likely to be perturbed, whereas values close to 0 indicate that the corresponding spike most likely will remain unchanged. τ is crucial as it controls the sharpness of the perturbations. Higher τ leads to smoother perturbations whereas lower τ leads to more sharp perturbations.

- 7) In order to apply Pr to a sample of the dataset, it needs to be in the binary domain. A rounding function gives the final universal adversarial patch UAP as follows:

$$UAP = \text{round}(Pr). \quad (18)$$

The stealthiness of the UAP is defined based on the spatiotemporal sparsity of its spikes. It becomes a perturbation when added to the incoming input. The perturbation can be

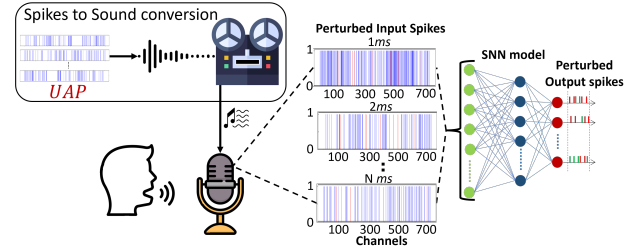


Fig. 5: Example universal adversarial attack application in the sound domain.

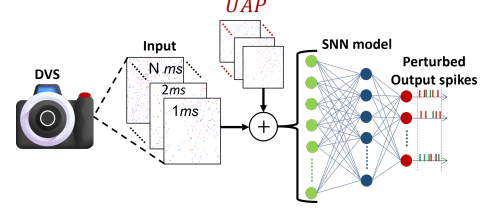


Fig. 6: Example universal adversarial attack application in the vision domain.

expressed using the same metric in Eq. 4 by setting $I = 0$. A single dataset inference suffices to generate the UAP, thus the adversarial example generation time is fixed and equals the dataset inference time. A second inference is performed to validate the ASR of the UAP.

The final UAP can be added to any input I during the deployment of the SNN model to fool the prediction. In the sound domain, the spike-based information from the UAP can be transformed into an audio signal by mapping the spike events to corresponding audio frequencies. This perturbation can then be recorded and replayed to a sound-to-spike sensor, causing the target SNN to misinterpret the input. Fig. 5 illustrates this attack flow, where human voice commands are perturbed by the UAP, leading to confusion in the SNN. In the vision domain, the spike-based information from the UAP can be injected into the event stream generated by a DVS. This modification alters the spatiotemporal event patterns before they reach the SNN, resulting in incorrect model outputs. Fig. 6 illustrates this attack flow.

V. EXPERIMENTAL SETUP

The adversarial attack algorithms are evaluated on three datasets, namely MNIST, IBM DVS Gesture, and SHD. The SNN training and gradient-based backpropagation for attack generation were performed using the SLAYER framework [14]. MNIST is a spiking version of MNIST that consists of 60K training and 10K testing images of handwritten digits, captured by a DVS while it views MNIST images on an LCD monitor [15]. The IBM DVS Gesture dataset contains 1341 samples of 11 hand and arm gestures performed by 29 individuals under three lighting conditions, and is also recorded via a DVS [16]. The SHD dataset includes 8332 training and 2088 testing samples of spoken English and German digits, converted into spike trains using a neuromorphic cochlea model, spanning in total 20 classes [17]. The corresponding SNN architectures are shown in Figs. 7-9. Table I summarizes the SNN characteristics, including the nominal

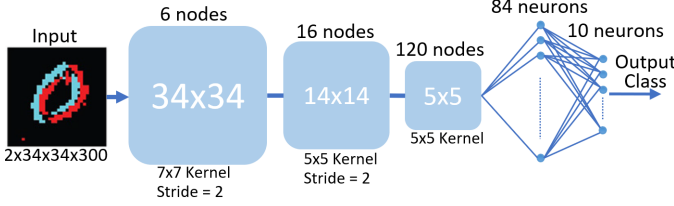


Fig. 7: SNN architecture for the MNIST dataset.

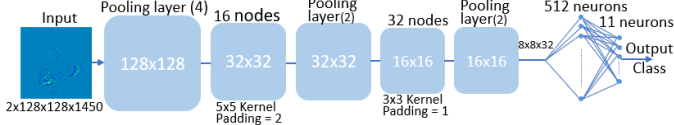


Fig. 8: SNN architecture for the IBM DVS Gesture dataset.

prediction accuracy, the input spatial and temporal dimensions, and the input spikes size considering an $1ms$ timestep.

VI. RESULTS

A. Input-specific adversarial attack

Table II summarizes the results of the input-specific adversarial attack for the three case studies. We evaluated only the samples that were correctly classified, as generating an adversarial example for a misclassified sample would be meaningless. The algorithm successfully generates an adversarial example for all tested samples, achieving an ASR of 100%. The maximum average perturbation across the three case studies is 0.0585%, rendering the attacks extremely stealthy. Table II also shows the minimum, maximum, and average adversarial example generation time, as well as the average number of iterations across the tested samples. Figs. 10 and 11 illustrate for the MNIST and IBM DVS Gesture case studies, respectively, the result for two tested samples. Each column corresponds to one tested sample. The first three subplots show three snapshots of the adversarial input, where, for the purpose of better visualization, each snapshot projects the spikes of 10 consecutive timesteps onto one plane. Black and red dots indicate original and perturbed spikes, respectively. The subplot at the bottom of the column shows the spike count of output neurons corresponding to the different classes for the original dataset sample and its adversarial example. The results demonstrate that minor perturbations in spike trains, forming an adversarial example nearly indistinguishable from the original sample, consistently caused misclassification. This underscores both the effectiveness of the proposed algorithm in terms of ASR and stealthiness, and the susceptibility of SNNs to the introduced adversarial attack.

B. Universal adversarial attack

Table III presents the performance of the proposed universal adversarial attack across the three case studies. As it can be seen, mixing inputs with the UAP results in low perturbation and consistently high ASR. Fig. 12 visualizes the spike pattern of the UAP for each case study. For the MNIST and IBM we show three snapshots that accumulate the spikes across 10 timesteps, while for the SHD we show the spike pattern across the channels for the first 4 timesteps. The UAP shows a sparse

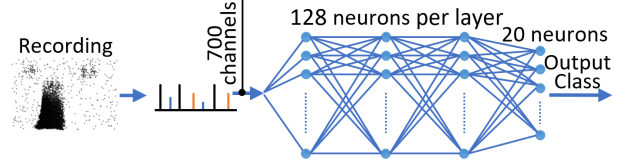


Fig. 9: SNN architecture for the SHD dataset.

TABLE I: Benchmark SNNs characteristics.

	NMNIST	IBM	SHD
Prediction accuracy	98.19%	86.36%	76.59%
# Output classes	10	11	20
Input spatial dimension	$2 \times 34 \times 34$	$2 \times 128 \times 128$	$700 \times 1 \times 1$
Input temporal dimension	300 ms	1.45 s	1 s
Input spikes size	693600	47513600	700000
Size training set	60K	1080	8332
Size testing set	10K	261	2088

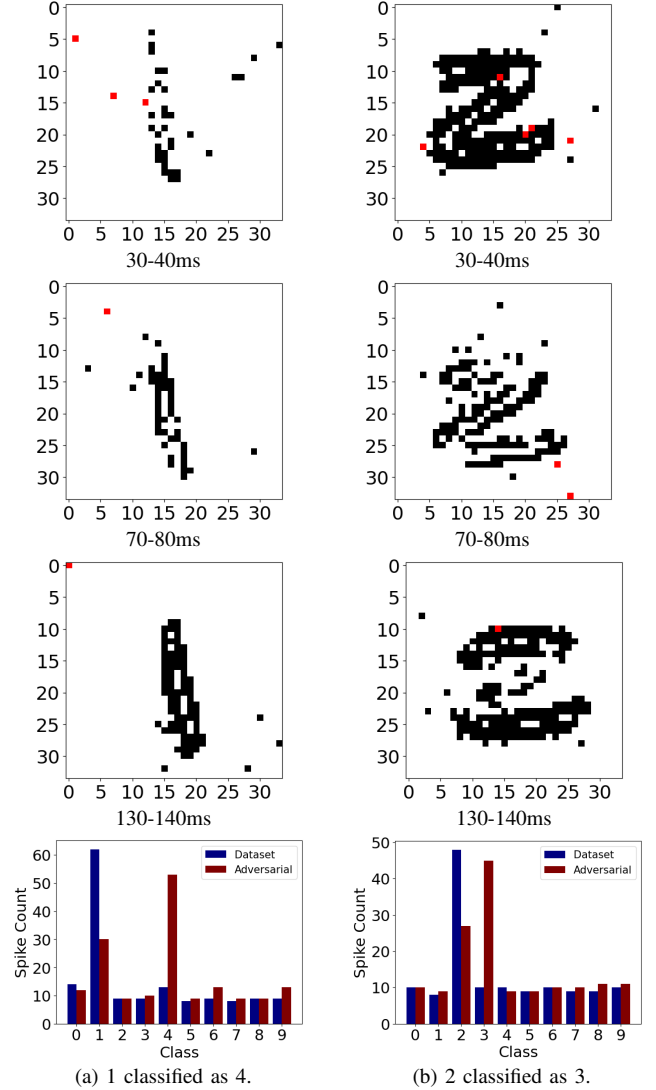
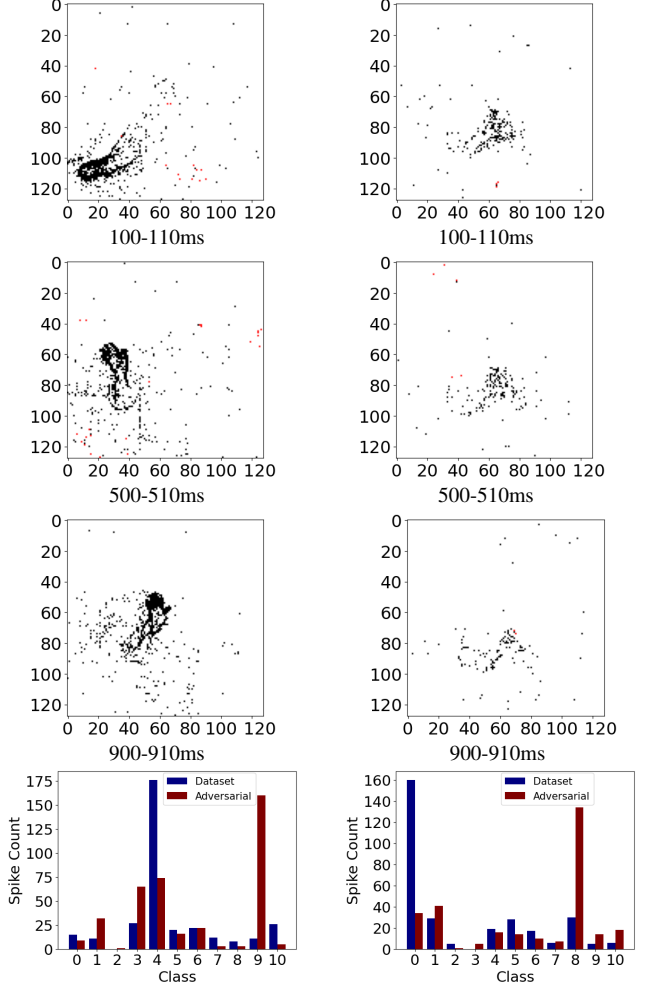


Fig. 10: Adversarial examples for the N-MNIST dataset.

spike pattern and when added to the incoming sample becomes imperceptible. Although the perturbation is larger compared to the input-specific adversarial attack, the advantage of the UAP is that it can manipulate the vast majority of inputs in real-time, fooling the network's decision.

TABLE II: Input-specific adversarial attack results.

	NMNIST	IBM	SHD
Samples tested	69369	1304	9921
ASR	100%	100%	100%
Average perturbation	0.0305%	0.0018%	0.0585%
Minimum generation time	0.0121s	0.122s	0.0061s
Average generation time	2.277s	2.465s	9.7688s
Maximum generation time	6.3085s	4.682s	36.4226s
Average number of iterations	114	74	1008



(a) Right arm counter clockwise classified as playing guitar. (b) Hand clapping classified as playing drums.

Fig. 11: Adversarial examples for the IBM DVS Gesture dataset.

C. Comparison with the state-of-the-art

1) *Input-specific adversarial attack*: Tables IV and V present for the NMNIST and IBM DVS Gesture case studies, respectively, a comparative analysis with the respect to the state-of-the-art gradient-based attacks, namely the *sparse* attack [10], the *SpikeFool* attack [11], and the attack in [12]. For the *SpikeFool* attack, more than one result is shown exploring the trade-off between ASR and perturbation. The evaluation metrics for these prior attacks were taken from [11]. Therein, the spikes of each recording were binned into a reduced set of timesteps (e.g. 60 for the NMNIST and 145 for the IBM DVS Gesture), capping the maximum number of spikes to 1 per pixel. This reduced the input spikes size significantly from

TABLE III: Universal adversarial attack results.

	NMNIST	IBM	SHD
Samples tested	69369	1304	9921
ASR	81.66%	87.66%	78.35%
Average perturbation	0.45%	0.079%	0.57%

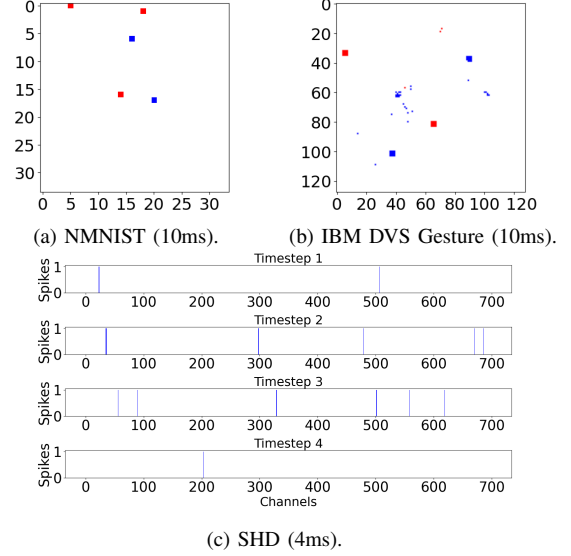


Fig. 12: UAP illustration.

$2 \times 34 \times 34 \times 300 = 693600$ to $2 \times 34 \times 34 \times 60 = 138720$ for the NMNIST and from $2 \times 128 \times 128 \times 1450 = 47513600$ to $2 \times 128 \times 128 \times 145 = 4751360$ for the IBM DVS Gesture. Moreover, only 1000 samples were tested. In contrast, the results for our proposed method are produced considering the full input spike size and all samples in the dataset that are correctly classified. In other words, the ASR for the prior attacks is likely to be optimistic as it is computed on a sub-dataset and, in addition, our algorithm was executed searching in the much larger raw input spikes space, thus solving a more challenging optimization. Still, the results show that are our attack outperforms all prior attacks in all metrics.

2) *Universal adversarial attack*: Only [11] proposes a solution providing results for the IBM Gesture IBM. The comparison is given in Table VI which shows the ASR per class, as well as the average ASR across all classes in the last column. As it can be seen, our attack outperforms [11] in 8 out of 11 classes (while achieving 100% ASR on 5 classes) and on average across all classes with 87.6% as opposed to 77%. The perturbation size is not reported in [11], while we report it for our attack in Table III. Qualitatively, our attack is more stealthy as the spike pattern of the *UAP* is sparse and is distributed both spatially and temporally, while in [11] the patch is limited to the area where the actual gesture is performed making it noticeable.

VII. CONCLUSION

We introduced two innovative adversarial attack methods for SNNs leveraging spatiotemporal gradients in the spiking domain: an input-specific attack and a universal adversarial attack. The input-specific attack generates adversarial examples from any dataset sample while introducing minimal, imperceptible perturbations. The universal adversarial attack

TABLE IV: Comparison of input-specific adversarial attacks on the NMNIST.

Attack	Samples tested	ASR	Perturbation	Average generation time/frame	Input spikes size
[10]	1000	74.44%	23.28%	0.0106 s	138720
[11] (1)	1000	99.53%	0.1845%	0.4145 s	138720
[11] (2)	1000	99.88%	0.196%	0.2 s	138720
[12]	1000	100%	0.6237%	0.015 s	138720
This work	69369	100%	0.0305%	0.00759 s	693600

TABLE V: Comparison of input-specific adversarial attacks on the IBM DVS Gesture.

Attack	Samples tested	ASR	Perturbation	Average generation time/frame	Input spikes size
[10]	1000	92.44%	4.78%	$9.65e^{-4}$ s	4751360
[11] (1)	1000	100%	0.0065%	0.0176 s	4751360
[11] (2)	1000	99.87%	0.0042%	0.0165 s	4751360
[11] (3)	1000	97.61%	0.0024%	0.0196 s	4751360
[12]	1000	99.77%	0.028%	0.0036 s	4751360
This work	1304	100%	0.0018%	0.0017 s	47513600

crafts a single reusable perturbation that significantly degrades accuracy across the vast majority of inputs, making it highly efficient for real-time deployment. Results on SNNs trained for the NMNIST and IBM DVS Gesture vision datasets, show that the proposed attacks outperform across all evaluated metrics all prior state-of-the-art attacks. Furthermore, for the first time we demonstrate adversarial attacks on the SHD neuromorphic auditory dataset. Overall, this work provides new insights into adversarial robustness of SNNs and highlights critical security implications for neuromorphic computing, emphasizing the need for dedicated defense mechanisms tailored to SNNs. Existing defenses, such as adversarial training [9], [11] and input filtering [10], offer potential countermeasures but often introduce trade-offs like increased computational overhead or reduced accuracy on clean inputs. As future work, we aim to develop SNN-specific defense mechanisms that effectively balance robustness, efficiency, and real-world applicability.

REFERENCES

- [1] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, Nov. 2019.
- [2] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nat. Comput. Sci.*, vol. 2, no. 1, pp. 10–19, Jan. 2022.
- [3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv:1312.6199v4*, 2014.
- [4] J. C. Costa, T. Roxo, H. Proença, and P. R. M. Inácio, "How deep learning sees the world: A survey on adversarial attacks & defenses," *IEEE Access*, vol. 12, pp. 61113–61136, Apr. 2024.
- [5] S. Sharmin, P. Panda, S. S. Sarwar, C. Lee, W. Ponghiran, and K. Roy, "A comprehensive analysis on adversarial robustness of spiking neural networks," in *Proc. Int. Jt. Conf. Neural Netw. (IJCNN)*, Jul. 2019.
- [6] A. Marchisio, G. Nanfa, F. Khalid, M. A. Hanif, M. Martina, and M. Shafique, "Is spiking secure? a comparative study on the security vulnerabilities of spiking and deep neural networks," in *Proc. Int. Jt. Conf. Neural Netw. (IJCNN)*, Jul. 2020.
- [7] S. Sharmin, N. Rath, P. Panda, and K. Roy, "Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Aug. 2020.
- [8] R. El-Allami, A. Marchisio, M. Shafique, and I. Alouani, "Securing deep spiking neural networks against adversarial attacks through inherent structural parameters," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Feb. 2021, pp. 774–779.

TABLE VI: Comparison of universal adversarial attacks on the IBM DVS Gesture.

Attack	Hand Clap	RH Wave	LH Wave	RH Clockwise	RH Counter Clockwise	LH Clockwise	LH Counter Clockwise	Arm Roll	Air Drum	Air Guitar	Other	Average
[11]	90.3	99	89.8	87.3	79.7	49.7	51.5	63.6	79.1	92.3	64.7	77
This work	100	100	100	100	75.8	86.2	89.7	100	66.9	70.5	75.2	87.6

- [9] A. Bagheri, O. Simeone, and B. Rajendran, "Adversarial training for probabilistic spiking neural networks," in *Proc. IEEE Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Jun. 2018.
- [10] A. Marchisio, G. Pira, M. Martina, G. Masera, and M. Shafique, "DVS-Attacks: Adversarial attacks on dynamic vision sensors for spiking neural networks," in *Proc. Int. Jt. Conf. Neural Netw. (IJCNN)*, Jul. 2021.
- [11] J. Büchel, G. Lenz, Y. Hu, S. Sheik, and M. Sorbaro, "Adversarial attacks on spiking convolutional neural networks for event-based vision," *Front. Neurosci.*, vol. 16, Dec. 2022.
- [12] L. Liang *et al.*, "Exploring adversarial attack in spiking neural networks with spike-compatible gradient," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 5, pp. 2569–2583, May 2023.
- [13] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Front. Neurosci.*, vol. 12, May 2018.
- [14] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2018, pp. 1412–1421.
- [15] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Front. Neurosci.*, vol. 9, Nov. 2015, Article 437.
- [16] A. Amir *et al.*, "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017.
- [17] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 7, pp. 2744–2757, Jul. 2022.
- [18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learn. Representations (ICLR)*, May 2015.
- [19] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. Int. Conf. Learn. Representations (ICLR)*, Apr. 2018.
- [20] A. Modas, S.-M. Moosavi-Dezfooli, and P. Frossard, "SparseFool: A few pixels make a big difference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9079–9088.
- [21] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *arXiv:1712.09665*, 2018.
- [22] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, "HATS: Histograms of averaged time surfaces for robust event-based object classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 1731–1740.
- [23] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Feb. 2017.
- [24] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Nov. 2017.
- [25] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv:1308.3432*, 2013.
- [26] S. Raptis and H.-G. Stratigopoulos, "Minimum time maximum fault coverage testing of spiking neural networks," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar./Apr. 2025.
- [27] S. Raptis, P. Kling, I. Kaskampas, I. Alouani, and H.-G. Stratigopoulos, "Input-triggered hardware trojan attack on spiking neural networks," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST)*, May 2025.
- [28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2017.