Doctoral Dissertation

Doctoral Program in Electronic Engineering (36thcycle)

Security and Privacy in Artificial Intelligence

By

Farzad Nikfam

******

Supervisor(s):

Prof. Maurizio Martina, Supervisor

Prof. Muhammad Shafique, Co-Supervisor

Politecnico di Torino

2024

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Farzad Nikfam
2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Abstract

Artificial Intelligence (AI) is considered the great revolution of the modern era. After the industrial revolution that took place over a century ago, other technologies such as computers and the World Wide Web have drastically changed the world, but none of them has unnerved humanity like AI. In reality, artificial intelligence is nothing more than a technology like any other, with great potential and great risks. Like every new discovery and invention, it must be regulated, and we must learn to use it correctly to harness its full potential.

Contrary to its name, AI has nothing truly intelligent about it; it is simply an evolution of mathematical statistics. Thanks to the computing power of modern computers and state-of-the-art software, AI enables us to do things that, until recently, only humans could do. The significant change introduced by AI is hidden from the eyes of most people. AI has simply changed the paradigm of how we have always thought about solving problems. Until now, we thought that by having an equation ($f(x)$) and its solution, i.e., the output (y), we could obtain the input (x) by solving the equation mathematically. What AI does is change this paradigm, allowing us to find very complex equations, which we couldn't find with simple calculations, by examining not only the outputs but also the input variables. It's no longer a matter of finding the x of equations studied in school; with many x and y, we need to find the equation that connects them. This is the true revolution of AI.

However, like all major technologies, AI has its risks and dangers. What I will show in this thesis is an example of the most classic problems and risks associated with AI. Specifically, we will address privacy and security, providing examples in various fields to understand how sometimes a small manipulation can cause enormous damage, sometimes unintentionally, and other times with declared malicious intentions. In the following chapters, we will delve into the details of two cases, one related to security and the other to privacy, which I personally dealt with during my

Ph.D. I will present some possible solutions that I found together with my research group. In conclusion, I would like to emphasize that it will not be AI dominating us but our intelligence deciding whether to remain masters of ourselves or choose self-destruction.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms / Abbreviations

AI	Artificial Intelligence
BFV	Brakerski/Fan-Vercauteren
CNN	Convolutional Neural Network
DNN	Deep Neural Network
FAT	Free Adversarial Training
FHE	Fully Homomorphic Encryption
HE	Homomorphic Encryption
LIF	Leaky Integrate-and-Fire
LR	Learning Rate
LRF	Learning Rate Finder
ML	Machine Learning
NB	Noise Budget
PGD	Projected Gradient Descent
ReLu	Rectified Linear Unit
SNN	Spiking Neural Network

Chapter 1

Security and Privacy

As Machine Learning (ML) [1–3] continues to advance and permeate various aspects of our lives, it brings with it a host of security and privacy challenges [4]. One significant concern is the vulnerability of ML models to adversarial attacks. Adversarial attacks involve manipulating input data in subtle ways that are imperceptible to humans but can lead to misclassification by ML models. These attacks pose serious threats across various domains, including image recognition, natural language processing, and autonomous vehicles.

Moreover, the widespread collection and use of personal data for training ML models raise privacy issues. Organizations often gather vast amounts of sensitive information from individuals, such as personal preferences, health records, and financial data. If not handled properly, this data can be susceptible to unauthorized access, leading to breaches of privacy and potential misuse.

Furthermore, the deployment of ML models in critical systems, such as healthcare, finance, and transportation, introduces concerns about reliability and safety. Flaws or biases in the models can lead to incorrect decisions with far-reaching consequences, underscoring the need for robustness and accountability in ML systems.

Addressing these security and privacy challenges requires interdisciplinary efforts involving experts in ML, cybersecurity, ethics, and policy-making. Strategies such as robust model training, data anonymization, encryption techniques, and regulatory frameworks are essential to mitigate risks and ensure the responsible development and deployment of ML technologies.

In this Chapter 1, we will provide a general analysis of some common issues in the field of security and privacy, along with the existing techniques to address these problems. Subsequently, in Chapters 2 and 3, we will examine two practical examples of research aimed at improving some of these issues:

- Chapter 2 - Security Problem - Accelat: Fast Adversarial Training of DNNs.
- Chapter 3 - Privacy Problem - Spyking: Homomorphic Encryption on DNNs and SNNs.

1.1 Security in Artificial Intelligence

The security problem [5] in ML concerns the protection of models, data, and results of the ML process from external threats that may compromise the integrity, confidentiality, and reliability of the system [6].

Some of the most common cases include adversarial attacks [7], which we will delve into further in Chapter 2. For example, external malicious intrusions can deceive models with potentially disastrous consequences. Consider autonomous vehicles [8]; in the event of an external attack, they may fail to recognize road signs and fail to adjust the cruise speed, significantly increasing the risk of accidents. Similarly, in healthcare applications [9], corrupted models may misclassify medical data of some patients, leading to potentially fatal misdiagnoses.

Among the various types of attacks are data misclassification, injection of adversarial data, or removal of correct data. However, there are also types of attacks that exploit existing biases in the models themselves because they were trained on datasets already influenced by human bias. Consider the gender bias [10]; many models, for example, classify doctors as men and nurses as women. Similarly, models often struggle to identify or create data on the Black population because they were underrepresented in training data [11]. There have been cases in the past where biometric recognitions struggled to distinguish between the faces of Black individuals because they could not capture their main characteristics as they did on lighter faces. As one can imagine, these cases can have serious consequences in terms of security in unauthorized access to data or influencing model outcomes.

Finally, security also concerns all the infrastructures that we use every day, and in cases of external attacks, they may experience malfunction or total service interruption. For example, in the management of traffic at major airports, train stations, commercial ports, or energy industries.

To address these issues, it is necessary to implement robust security measures throughout all stages of the ML lifecycle, including data collection, model training, testing, and deployment. This may involve the use of encryption techniques, data validation, model diversification, and training personnel on best security practices.

1.1.1 Adversarial Attack

Adversarial attack is a technique [7] used to deliberately manipulate Artificial Intelligence (AI) models. The goal of an adversarial attack is to deceive the model into producing incorrect or undesired results by including malicious or imperceptibly modified inputs.

A common example of an adversarial attack occurs in the context of image classification [12]. Suppose we have a deep learning model trained to recognize objects in images, specifically distinguishing between images of cats and images of dogs. An attacker could create an image of a cat that appears normal to the human eye, but add small imperceptible perturbations to the model. These perturbations are designed to deceive the model and cause it to incorrectly classify the image as a dog instead of a cat.

Another example of an adversarial attack could occur in the context of text classification [13]. Suppose we have a ML model trained to classify movie reviews as positive or negative. An attacker could deliberately manipulate the words or structure of a review to cause it to be incorrectly classified as positive instead of negative, or vice versa.

In both examples, the attacker's goal is to exploit weaknesses or vulnerabilities in the ML model to achieve undesired results. Adversarial attacks raise concerns about the security and reliability of ML models, especially when used in critical contexts such as cybersecurity, medical diagnosis, or autonomous driving.

The most common method to generate an adversarial attack adv_x on data x with label y is by adding a small perturbation ε to the original image according to the formula presented in Equation (1.1), where Δ_x is the gradient function and Θ represents the weights of the loss function J.

$$adv_x = x + \varepsilon \cdot sign(\Delta_x J(\Theta, x, y)) \tag{1.1}$$

Adversarial attacks will be further explained and utilized in the research project of Chapter 2.

1.1.2 Model Inversion Attack

Model inversion attacks [14] are a class of privacy attacks designed to extract sensitive or private information about individuals using a trained ML model. The basic idea of a model inversion attack is to use the model itself, along with a set of public or background data, to *invert* or *recover* personal information about the individuals used to train the model. In practice, this means that an attacker uses the model to infer or guess the characteristics or personal data of an individual based on the model's predictions and other public information.

For example, suppose we have a ML model that classifies facial images as $smiling$ or $notsmiling$. An attacker could use this model to attempt to determine if a particular person is smiling based on an image of their face. The attacker might repeatedly feed the model with different images of the person's face, observing the model's predictions and using this information to infer whether the person is smiling or not.

In a broader context, model inversion attacks can be used to recover other sensitive information about individuals, such as sexual orientation, religion, political preferences, or other personal characteristics. This type of attack raises significant privacy concerns, as it can be used to violate people's privacy and jeopardize the security of their personal information.

1.1.3 Model Extraction Attack

Model extraction attacks [15], also known as model learning attacks, are a category of attacks where an attacker seeks to extract or replicate a trained ML model from another party. The main goal of a model extraction attack is to obtain an approximate copy or representation of the ML model used by another party without having direct access to the model itself. This type of attack can be executed by leveraging information obtained from the responses of the trained model to certain inputs.

Here's an example to better explain the concept: imagine you work for a company that has developed a highly accurate ML model for detecting malware in files. The model has been trained on a large dataset and has proven to be effective in detecting various forms of malware. A malicious competitor wants to develop a similar model but does not want to invest the time and resources required to collect and annotate a large dataset of malware for model training. Instead, the attacker could execute a model extraction attack to obtain an approximate copy of the malware detection model developed by the company. The attacker could send a large number of files to the company's model and carefully observe the model's responses for each file, along with details about the files themselves. Using this information, the attacker could gather a set of input-output pairs, i.e., the input files and the corresponding predicted output labels from the company's model. With a sufficiently large number of input-output pairs, the attacker could use ML techniques to train a new model that approximates the behavior of the company's model. Once trained, the attacker's new model could be used to detect malware with a similar accuracy to the original model developed by the company, without the need to collect its own training data. In this way, the attacker has effectively *extracted* the company's malware detection model, gaining a competitive advantage without having to invest resources in training a model from scratch. This is just one of many scenarios where model extraction attacks can be used to compromise the security and intellectual property of ML models.

1.1.4 Membership Inference Attack

Membership inference is a privacy attack technique [16] in which an attacker seeks to determine whether specific data belongs to the training set of a ML model or not. The main objective of a membership inference attack is to determine whether a particular data point, known as a *query*, was used to train a ML model without having direct access to the model itself or the training set.

Here's membership inference explained with a practical example: imagine you have a ML model trained to classify emails as spam or non-spam. The model has been trained on a large dataset of emails labeled as spam or non-spam. However, you do not have access to the actual training set used to create the model. Now, suppose you have a series of emails that you want to classify using the ML model. However, you are concerned that some of these emails may have been used during the training of the model itself. To determine whether a particular email was used during the training of the model, you could perform a membership inference attack. In this attack, you send your query email to the model and carefully observe the model's response. Subsequently, you slightly modify your query email and send it to the model again, observing the response once more. If the model's responses change significantly when you send slightly modified versions of your query email, it may indicate that your query email is similar to those used during the training of the model. In this way, you could infer that your query email is part of the model's training set. In this example, the attacker uses the membership inference attack to determine whether a particular email belongs to the training set of the ML model or not, leveraging the model's responses to the queries sent.

1.1.5 De-Anonymization

De-anonymization is a technique [17] by which anonymized data is correlated or linked to specific identities, making them effectively no longer anonymous. This process can be used to identify or trace individuals behind data that was previously considered devoid of personally identifiable information.

Suppose we have a dataset of anonymized health records containing information about patients with a particular disease. Initially, personally identifiable information such as names, surnames, or identification numbers has been removed or replaced with generic unique identifiers. However, a malicious researcher might attempt to de-anonymize the dataset by trying to correlate the information contained in it with other sources of public or private data. For example, the researcher could acquire another dataset containing demographic information from a specific geographic area, such as age, gender, and postal codes. By using this demographic information along with the anonymized data in the health dataset, the researcher could try to identify individual patients. If the health dataset contains detailed enough information and if the researcher has sufficient knowledge or tools, they might be able to link the anonymized information to specific identities in the real world. In this example, de-anonymization jeopardizes the privacy of patients, as their initially anonymized information is exposed and can be used to identify them. Such practice is often considered a privacy violation and can have serious consequences for the individuals involved.

Another well-known case of de-anonymization is related to the Netflix Prize mentioned in Section 1.2.

1.2 Privacy in Artificial Intelligence

Nowadays, we are increasingly concerned about our lack of privacy [18], and while on one hand, we make all our data available by posting the details of our daily lives on social media, on the other hand, we also worry about keeping some sensitive data confidential, but why?

The answer is quite simple, but not to be taken for granted. For example, if we let everyone know where we live and what our usual working hours are, we could allow malicious individuals to choose the right day to burglarize our home. This is a classic example, but why are we often asked to consent to the processing of personal data, and why are there so many laws about it?

Home burglary is a trivial example, but many other implications could arise from the dissemination of our data without us realizing it. For instance, a hospital could collect data on its cancer patients [19] to try to conduct statistical studies to implement new treatments. However, to conduct this study, it needs to have this data analyzed by an external company, and this is where problems could arise. The hospital cannot pass on that data without anonymizing it; otherwise, it could easily end up in the hands of third parties, such as companies that provide health insurance. Upon learning about the health conditions of some patients, they might refuse to provide insurance, or worse still, they might do so by requiring a much higher premium than normal. At this point, one might think that anonymizing the data is sufficient, but are we sure?

The most emblematic case of data deanonymization is that of the Netflix Prize of 2009. Netflix at the time decided to award \$1 million to those who managed to improve the algorithm recommending new films to users by over 10%. To do so, they released an anonymized dataset containing lists of films, reviews, and users. The films and users did not appear as names but as integers; they had been anonymized.

However, some researchers [20] managed to deanonymize a large part of the dataset, compromising the users' privacy. They used a simple statistical technique that allowed them to compare the Netflix dataset with IMDB review data. By comparing data from both sources, they were quickly able to find the names of users who had reviewed on both sites and consequently also discovered the names of the films. This example shows that anonymizing data is not sufficient to maintain privacy. Living in a world where the boundary between public and private is becoming increasingly blurred, we need to seek, find, and exploit increasingly sophisticated anonymization techniques.

Anonymity, and consequently privacy, derive from uncertainty and not just from data camouflage. Nowadays, to privatize data, it is no longer enough to simply hide it, but one must always introduce a certain amount of uncertainty so that it becomes practically impossible to understand which data are real and which are not. Unfortunately, however, it is necessary to always consider that there is a trade-off between the quality of usable data and their anonymization (see Figure 1.1). Sometimes data with too much noise are now useless, and those with too little noise are not anonymous enough. The purpose of the privacy-preserving techniques [21–23] that we will see in this Section 1.2 is precisely to overcome these issues by finding the right balance between privacy and utility.

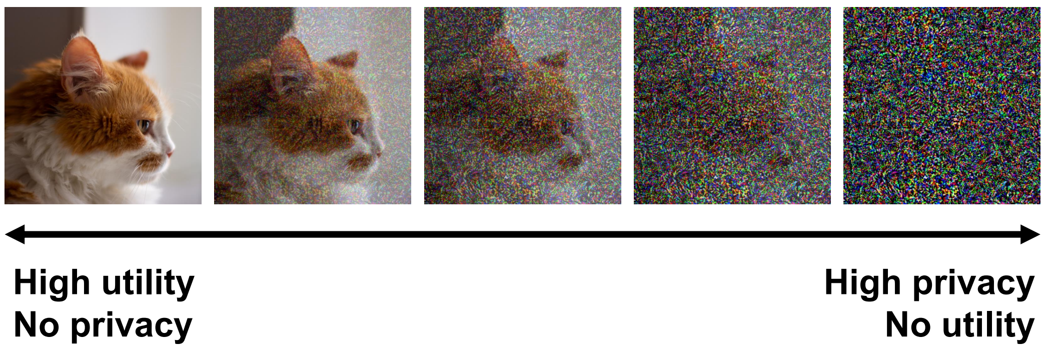


Fig. 1.1 Trade-off between privacy and utility.

1.2.1 Homomorphic Encryption

Homomorphic Encryption (HE) is a cryptographic technique [24] that allows operations to be performed on encrypted data without the need to decrypt it. In other words, it enables computations to be carried out on encrypted data, yielding encrypted results that, once decrypted, correspond to the results that would be obtained if the operations were performed on unencrypted data. This is particularly useful for data privacy and security, as it enables sensitive data to be processed without ever revealing it in unencrypted form.

HE can take various forms, allowing computations at various levels. For example, a partially HE of the multiplicative type allows for the multiplication of two encrypted numbers, resulting in an encrypted outcome that, once decrypted, corresponds to the product of the original numbers. This type of HE is especially useful in applications such as salary computation or medical data analysis, where complex operations need to be performed without compromising data privacy.

We will delve deeper into this type of cryptography in Chapter 3, where we will apply it to a real research case.

1.2.2 Garbled Circuit

The term *garbledcircuit* refers to a cryptographic technique [25] used to perform computations on encrypted data so that the results are accessible only to those who possess the correct decryption keys. This technique is widely used in the field of cybersecurity and privacy protection. In simple terms, a garbled circuit allows two parties to compute a result without revealing their input data to each other. It works by encrypting logical circuits, such as those used in mathematical calculations, so that the values of the data are hidden. Only authorized parties can interpret the data and obtain the correct result.

Suppose A and B want to compare their salaries but do not want to reveal the specific details to their respective employers. Using a garbled circuit, they can compare their salaries without disclosing the exact figures. A and B agree on a garbled circuit protocol and generate an encrypted logical circuit that performs the operation of comparing their salaries. A and B encode their salaries so that they are represented by encrypted data within the circuit. The garbled circuit performs the comparison so that only A and B can interpret the results. The result of the comparison is returned only to A and B, while their employers cannot access the original data. In this example, the garbled circuit allows A and B to perform a comparison operation on their salaries without revealing the exact figures to their employers. This technique is useful in scenarios where it is necessary to protect the privacy of data during encrypted computations.

1.2.3 Differential Privacy

Differential privacy [26] is one of the most commonly used techniques to attempt to anonymize a group of data. To achieve this goal, differential privacy adds random noise so that it is not possible to revert to the original data, but without statistically altering the dataset.

Differential privacy can formally be defined according to the formula presented in Equation (1.2), where the parameters are as follows:

- M - random algorithm.
- S - the predictable outputs of M.
- Pr - the probabilty.
- x - data from the dataset.
- y - data from the parallel dataset.
- ε- the maximum distance between the same data in the two databases.
- δ - the probability that information is leaked.

$$Pr[M(x) \in S] \leq exp(\varepsilon)Pr[M(y) \in S] + \delta \tag{1.2}$$

For example, with an anonymous questionnaire directed at a group of people, each person's profile could be reconstructed based on the answers given, narrowing down the circle of possible authors for each response. What differential privacy does, in this case, is to add random noise to each response so as not to be able to reliably reconstruct the original profile of the respondent. For instance, if a single question is asked with only two answers, *yes* and *no*, the respondent can answer whatever they want, but before recording their answer, they flip a coin. If it lands heads, *yes* is recorded, otherwise, the coin is flipped again. At this point, if heads come up, *no* is recorded, otherwise, if the coin is flipped again, the correct answer is recorded.

By doing so, it is obvious that the cases of authentic replies recorded are reduced to 25% of the total, but it also ensures that it becomes practically impossible to trace the authors of the replies. The truth is that the useful answers are not 25%, but almost 100%, contrary to what one might think. Since the coin toss is statistically random, we should obtain 50% false *yes* answers, 25% false *no* answers, and 25% true answers. Knowing this data, we can analyze our data in relation to these percentages and thus obtain correct studies as if the noise had not been introduced. The problem with this technique is that to anonymize the data, sometimes we tend to insert too much noise, risking significantly to decrease the amount of useful data. There is always a trade-off to consider between the goodness of the data and their anonymity.

Another critical point is the amount of data present. Being a random noise technique, it works better on a large amount of data, while it risks having significant errors on small datasets.

1.2.4 Federated Learning

With federated learning [27], the underlying method of operation in ML changes. As the word itself suggests $federated$, in this case, it's not about a single super-model built based on the data sent and accumulated in a single dataset with the help of high computing power. Instead, in this case, it's the algorithm that is sent to the *small* devices containing the data. In federated learning, unlike what happens in distributed learning, data is not requested from users, who can be private individuals, but also large companies, such as banks. Instead, it's the algorithm that is sent/installed on the local server to perform calculations on the local dataset. Once the calculations are completed, only the weights that characterize the model are sent to the central server to improve global learning.

From a technical standpoint, the objective function is given by Equation (1.3), where K represents the number of nodes among which the model is divided, x_i represents the weights of each individual model, and f is the local objective function of each individual node.

$$f(x_1,...,x_K) = \frac{1}{K}\sum_{i=1}^{K} f_i(x_i) \tag{1.3}$$

In federated learning, much less computing power is needed, and therefore more limited results are derived since it is carried out on local servers/computers. Furthermore, the amount of data available is not always homogeneous and is often intermittent due to the lack of connection with the central server. All these aspects might lead us to think that federated learning is useful only to maintain data privacy; however, it has several applications, and one of the best-known is that of the smartphone autocorrector that we all use every day.

The autocorrector indicates to us every term that it considers *wrong* according to its model, but it also suggests the most probable words that could appear after the one already written, and this function is personalized for each individual. The algorithm slowly learns to use the terms we use most often but does not share them with others. If we often use a name that it does not know, then over time that name will automatically appear on our smartphone, but not on that of others, maintaining privacy. At the same time, however, it learns grammar better and better, and this helps everyone because it is implemented in the central model.

1.2.5 Secure Multiparty Computation

This is a very efficient technique [28] for preserving privacy when multiple parties with similar data are involved. With secure multiparty computation, all the parties involved only partially know the data of their partners, but they cannot go back to the original data in any way.

To better understand how it works, we can give a practical example. Let's say we want to analyze the data of the major social networks users, take for example Facebook, Twitter, Instagram, TikTok, and YouTube. Let's assume that these companies have no relations with each other and there are no internal data exchanges, as happens in reality between Facebook and Instagram. Everyone's goal is to profile their users compared to the average users who take advantage of the major social networks (see Equation (1.4)).

$$f(x_1, x_2, ..., x_n) = avg(x_1, x_2, ..., x_n) \tag{1.4}$$

Everyone, therefore, wants to understand if compared to the average it has more male, female, young, elderly users, coming from certain areas of the world, etc. For simplicity, let’s take a single datum: that is, every company wants to understand if its users have more males or more females than the average. Now let’s assume that the percentage of males out of the total for each social network is the following: Facebook 70%, Twitter 60%, Instagram 85%, TikTok 35%, and YouTube 60%. With secure multiparty computation, none of them will have to tell the exact percentage to the others, but everyone will be able to know what the total average is. In practice, each company takes its data and divides it into smaller percentages for the total number of companies. In the case of Facebook where we have placed 70% of males, we divide the data into 5%, 10%, 15%, 15%, 25%. The sum of the percentages thus divided makes 70% again, but this does not know anyone outside of Facebook itself, at this point, all the companies do the same thing and distribute every single part of the total to each of the other companies. In doing so, each company has a fragment of the data of the other companies. Then, each company calculates the sum of the percentages received and makes it public. Now you can make an average of the data without any of the companies knowing exactly what the precise percentages of the others are because each has obtained only a part of the data of all the others.

1.2.6 Secret Sharing

The secret sharing scheme, or secret sharing, is a cryptographic technique [29] used to divide a secret into multiple parts, called *shares*, so that the secret can be reconstructed only when a sufficient number of these parts are combined together.

The secret sharing process involves three main phases:

- Secret generation - in this phase, the secret to be shared is generated. It can be any type of sensitive information, such as a cryptographic key, a password, or financial data.
- Secret division - the secret is divided into multiple independent parts, each of which contains a portion of the original information. These parts are distributed among different participants, called *sharers*, using a specific division algorithm.
- Secret reconstruction - to reconstruct the original secret, a sufficient number of parts must be collected. Generally, reconstruction can occur only when a

certain minimum number of participants collaborate together and combine their parts of the secret using a combination algorithm.

For example, imagine three friends want to share a safe containing valuable items, but they want to ensure that none of them can access the safe alone. They decide to use a secret sharing scheme.

- Secret generation - they decide to use a four-digit code as the secret to open the safe.
- Secret division - using a secret sharing scheme, the code is divided into three parts, each containing two digits of the code. Each friend then receives one part of the code, without knowing the other parts.
- Secret reconstruction - to open the safe, all three friends must come together and combine their parts of the code. Once each friend has correctly entered their part of the code, the safe opens.

In this way, no single friend is able to access the safe alone, but the secret can be reconstructed only when all three collaborate together.

1.2.7 Private Set Intersection

Private set intersection is a cryptographic technique [30] used to determine the intersection between two sets of private data held by two different parties without revealing any information beyond the intersection itself. In other words, it allows two parties to compare their sets of data without disclosing the specific details of the elements in their respective sets.

An example of private set intersection could involve two organizations wanting to compare their customer lists to identify any overlaps without revealing sensitive information about the users. Suppose Organization A has a customer list containing unique IDs associated with each customer, and Organization B has a similar list. Both organizations want to identify the customers that are present in both lists without revealing the names of the customers or other personal data. To perform the private set intersection, the two organizations adopt a cryptographic protocol. Firstly, Organization A encrypts its customer list so that only Organization B can interpret it.

Similarly, Organization B encrypts its own customer list so that only Organization A can interpret it. Subsequently, the two organizations exchange the encrypted data with each other. Using the private set intersection protocol, each organization can compare the other organization's encrypted data with its own encrypted data without ever revealing the specific details of individual customers. At the end of the process, both parties obtain only the list of customer IDs that are present in both sets without knowing any other details.

In this way, private set intersection allows two parties to compare their data without compromising the privacy of the individuals represented in the data themselves.

1.2.8 Zero-Knowledge Proof

Zero-knowledge proofs are a type of cryptographic protocol [31] that allows an entity to prove knowledge of specific information without actually revealing that information. In other words, the entity can demonstrate possession of certain knowledge without having to disclose what that knowledge is. This concept is extremely powerful for ensuring the privacy and security of information during communications and transactions.

Zero-knowledge proofs are used in various applications, such as authentication without revealing a password or proving knowledge of a private key without disclosing it. For example, suppose A wants to prove to B that they know the password to access a website, but without revealing the password itself.

- B sends a sequence of random *challenges* to A.
- A uses their secret password to respond to B's challenges.
- B verifies A's responses and confirms that they are all correct, demonstrating that A possesses knowledge of the password.

Despite B being confident that A knows the password, they have not actually learned anything about the password itself because A's responses contain no information about the password. This example illustrates how zero-knowledge proofs allow for proving knowledge of a password without revealing it. It is important to note that zero-knowledge proofs are designed to ensure that no useful information can be deduced from the provided responses, thus protecting the user's privacy.

Zero-knowledge proofs are applied in numerous scenarios, including financial transactions, authentication systems, identity verification procedures, and much more, helping to ensure a high level of security and privacy in digital communications and transactions.

1.2.9 Privacy Preserving Record Linkage

Privacy preserving record linkage is a technique [32] used to securely and privately link information from different sources without revealing sensitive or identifying data. The goal is to identify records that refer to the same entity across separate datasets without compromising the privacy of the data itself.

The are several phases:

- Pre-processing - before linkage, the data is preprocessed to ensure that there are no direct identifying information and to apply anonymization or encryption techniques.
- Tokenization - a cryptographic representation or *token* is created for each piece of data, so that the linkage can be performed based on these tokens without revealing sensitive information.
- Linkage - the data tokens are compared between datasets to identify matches. This linking occurs without the need to reveal the original data.
- Post-processing - after linkage, additional privacy protection techniques may be applied to remove any redundancies and ensure the security of the results.

Suppose we have two datasets, one containing people's names and their postal codes, and the other containing people's phone numbers and their respective postal codes.

- Pre-processing - before linkage, the datasets are anonymized by replacing people's names with unique identifiers and encrypting phone numbers.
- Tokenization - for each name and postal code in the first dataset and for each phone number and postal code in the second dataset, cryptographic tokens are generated.

- Linkage - the tokens from the two datasets are compared to find matches between people who share the same postal code.
- Post-processing - after linkage, additional privacy protection techniques, such as removing duplicates or adding noise to hide any identifiable patterns, may be applied.

In this way, privacy preserving record linkage allows for identifying matches between records in the two datasets without directly revealing personal information such as names and phone numbers. This ensures the privacy of the individuals involved and protects sensitive data during the linkage process.

Chapter 2

AccelAT

Machine Learning (ML) [33] has experienced rapid growth and widespread adoption in recent years, primarily due to advancements in hardware efficiency, particularly GPUs. However, training sophisticated and complex ML models often requires significant computational resources and time. Consequently, accelerating ML processes with fast training techniques allows for optimizing calculations and improving GPU management, even in large data centers. On the other hand, over the last decade, it has become evident that ML models are susceptible to external attacks not identifiable by humans. Hence, there is a growing need for ML models to possess robustness [34–36] against such attacks [37], especially when used in safety-critical applications [38].

In this work [39], we aimed to create models that are fast and robust to adversarial attacks [40–42], laying the groundwork for the future models [43].

In the flowchart presented in Figure 2.1, you can see a schematic of how the research work was conducted, divided into an initial phase of analysis and evaluation of existing models, followed by the study and development of new fast adversarial training techniques.

2.1 Adversarial Attacks

An adversarial attack [40, 44, 45] is a technique used to intentionally manipulate the input data into a ML model in order to induce errors in its classification or operation.

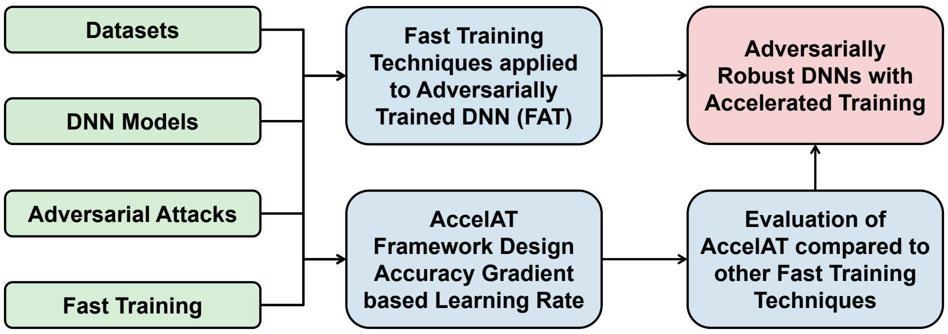

Datasets
DNN Models
Adversarial Attacks
Fast Training
Fast Training Techniques applied to Adversarially Trained DNN (FAT)
AccelAT Framework Design Accuracy Gradient based Learning Rate
Adversarially Robust DNNs with Accelerated Training
Evaluation of AccelAT compared to other Fast Training Techniques

Fig. 2.1 A summary flowchart of the AccelAT research project.

Adversarial attacks are designed to exploit vulnerabilities in ML models, which may be susceptible to small, imperceptible perturbations that can significantly influence the model's output. Adversarial attacks can be employed to deceive ML models, compromise their security, or cause malfunctions. For example, in cases where facial, voice, or fingerprint recognition is used to unlock certain services, an external attack can have serious consequences [46]. Complex Deep Neural Network (DNN) models are highly vulnerable to external attacks, with their accuracy potentially dropping from nearly 100% to about 0% in worst-case scenarios [47]. Consequently, there is a desire for DNN models to be robust against such external attacks. To counter these attacks, it is necessary to develop and implement robust models capable of maintaining high accuracy in the presence of such malicious variations.

2.1.1 Adversarial Examples

For an adversarial attack [40] to work, it's necessary to create modified data that is imperceptible to a human but can deceive DNN models and result in fatal outcomes. Randomly generated attacks wouldn't be effective because the data would be illegible even to a human, halting the entire process. However, targeted attacks can bypass human scrutiny and target the DNN model directly. In the case of image data, as shown in Figure 2.2, attacks focus on modifying images by adding small adversarial perturbations, resulting in an image that appears nearly identical to the original but is interpreted differently by the model.

In Figure 2.2, we can observe how an image initially classified as an *eagle* with over 90% accuracy is misclassified as a *chicken* with just the addition of some noise [48]. The noise image shown has been amplified 1000 times to make it visible. Thus,

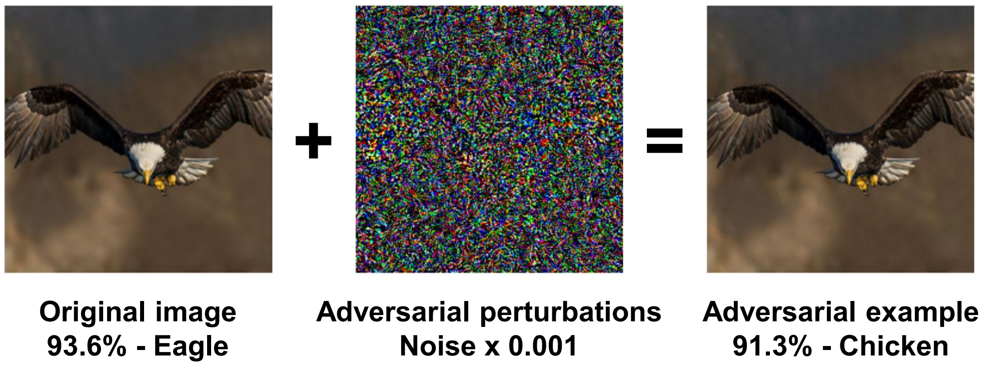


Fig. 2.2 An example of adversarial attack, where an eagle is misclassified as a chicken after adding specific noise to the original image.

while the initial and final images may appear identical to a human, every single pixel has been modified according to the noise pattern shown in the center of Figure 2.2.

There also exists the reverse scenario where images are attacked to yield completely wrong results for a human observer but are correctly classified by an Artificial Intelligence (AI) model. However, this latter case isn't critical as it wouldn't pass human scrutiny. Henceforth, when referring to external attacks, we mean those resulting in misclassification by DNN models.

2.1.2 Attack Methods

Given that the classification of data by DNN models does not occur as it does in the human brain, adversarial attacks have been devised considering how models tend to proceed to distinguish one class from another [49, 45]. In the specific case of image classification, for example, there are various techniques to deceive the model, including: variation of recurring patterns in images, modification of a single pixel [50] and variation of the decision boundary between two or more classes [51, 52].

The most common case is based on computing the separation line between two classes. In Figure 2.3a, we can see how in a simple case with two classes, there is an imaginary line that separates the features of one class from those of another class. To deceive the model, it is possible to slightly perturb the data by shifting the boundary features slightly beyond, as shown in Figure 2.3b, so that the data is not too altered but enough to be misclassified with the nearest similar class. Another scenario could involve shifting the entire demarcation line as depicted in Figure 2.3c, causing all boundary features to fall into the contiguous class, resulting in misclassification of the data. These are just some simplified examples of how an adversarial attack can perturb data and almost nullify the accuracy of a DNN model.

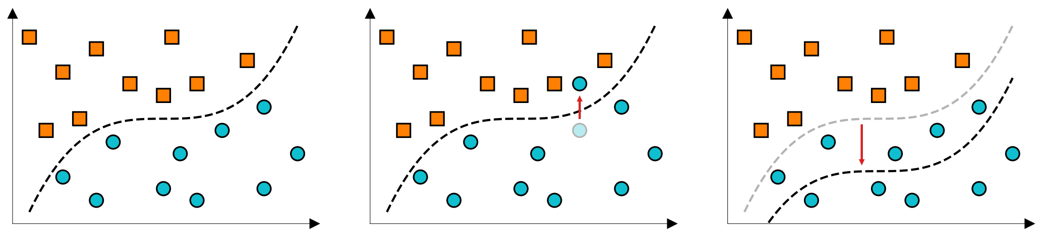

(a) Separation line. (b) Feature moving. (c) Line moving.

Fig. 2.3 Two ways of fooling a classifier using the separation line between two different classes.

2.1.3 White-Box Attacks

There are mainly two types of attacks, white box [53] and black box [54] (see Figure 2.4). The main difference lies in the fact that in white box attacks, the internal structure of the model is known, enabling a more targeted and powerful attack to be designed. On the other hand, in black box attacks, only the input and output are known, making them more complex and requiring more time to execute. In this work, only targeted white box attacks were conducted, knowing all the internal parameters of the attacked model.

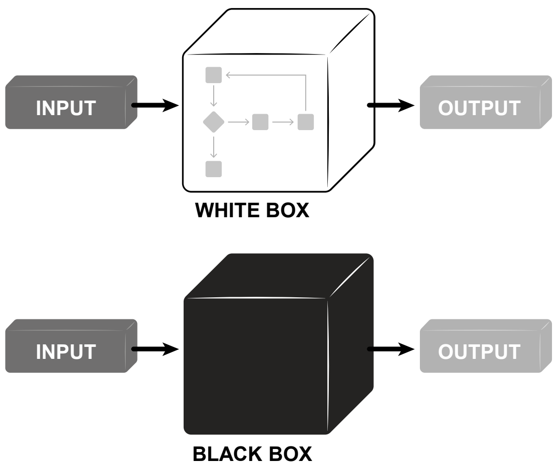


Fig. 2.4 Difference between a white box attack and a black box attack.

2.1.4 Adversarial Training

Adversarial training [55] involves training a model to be robust [35, 36] against external attacks. Various techniques can be exploited to obtain robust models [37], such as control with a second model, fine-tuning hyperparameters [56], and data augmentation [57].

The control method leverages two DNN models. One of the models, the secondary one, checks the other, namely the main one, to verify whether the analyzed data is adversarial or not. This technique uses an $external guard$ logic, but its actual effectiveness is still under study.

Hyperparameter tuning, such as weight decay, Learning Rate (LR), or batch size adjustment, is useful for countering external attacks, but it rarely leads to acceptable results.

The most widely used method is certainly data augmentation with the addition of adversarial samples [58]. Therefore, during training, the model utilizes both clean images, i.e., correct ones, and those with added noise, i.e., already attacked or generated with other models based on desired attacks. This latter technique leads to significant increases in accuracy against external attacks, but as a drawback, the performance towards clean images decreases. Additionally, this robustness comes at the expense of computation times, which inevitably become longer, proportional to the adversarial samples added to the training and the attack models against which robustness is desired.

2.1.5 Foolbox Library

Among the various existing libraries that allow implementing various types of attacks, we chose to use Foolbox [59]. This library stands out for its comprehensive documentation, features, and support for the TensorFlow [60] library, which was used in this work. With Foolbox, it is possible to execute a wide range of customizable attacks, which easily enable adversarial training.

2.2 Fast Training

The fast training is a technique [61] used in ML model training aimed at accelerating the learning process by reducing the time required for model training, without excessively compromising the performance or quality of the model itself. This can be particularly important in scenarios where training models on large datasets or in real-time is necessary, where learning speed is crucial. For example, popular websites like Google, YouTube, and Facebook need to manage continuous streams of incoming data [46], and being able to train data quickly is a necessity. On the other hand, using large amounts of data leads to significant energy consumption, which is unsustainable for small-scale applications with limited resources.

These improvements can be achieved through various strategies, including optimization of learning algorithms, the use of specialized hardware such as GPUs or TPUs to perform parallel computations, reducing the number of training data used, or optimizing learning parameters.

In this work, we aimed to expedite the training of robust models through hyperparameter [56] fine-tuning. Some of these parameters include epochs, batch size, weight decay, momentum, LR, etc. Among these, the most crucial is the LR, and to set it optimally, it's essential to first find the maximum usable LR.

2.2.1 Learning Rate Finder

The Learning Rate Finder (LRF) is a method used to determine the highest usable LR during training. To understand it better, one can imagine that most training models attempt to optimize results by seeking the deepest minimum within the search function. For this search, the crucial parameter is the LR, which determines how quickly the model will find a local minimum, ideally as close as possible to the absolute minimum. A too-small LR results in excessively long training times or, worse, sometimes leads to unacceptable convergence because the model gets stuck within a very small local minimum, resulting in very low accuracy. Conversely, a too-high LR risks preventing the model from finding the minimum as it continually overshoots it due to its high LR. Therefore, finding the optimal LR helps identify the upper limit for this hyperparameter, which can then be gradually reduced as the model converges and achieves adequate accuracy. This technique is particularly useful during the training phase of DNN models, where the LR is a critical hyperparameter that affects the speed and stability of the learning process.

The LRF automates the process of finding the optimal LR, avoiding the trial-and-error part that would otherwise need to be done manually. The most intuitive implementation of the LR finder involves using an exponential LR during a trial training session while simultaneously evaluating the model's loss [62]. For effective results, it is reasonable to vary the LR by at least ten orders of magnitude, and if the other parameters have been properly normalized, then the LR will typically range from 0.001 to 10. Therefore, it's advisable to cover this range of values during training. At the end of training, the optimal LR can be easily identified by examining the model's loss graph.

For example, looking at Figure 2.5, where the LR is exponentially increased and plotted on a logarithmic scale, it can be observed that for low LR values, the loss is relatively stable. From a certain point onwards, the loss begins to decrease significantly until it reaches a minimum, after which it diverges definitively. Consequently, the usable LR values are all those preceding the point of minimum loss. It is generally recommended to use a maximum LR value at least one order of magnitude less than that used during the loss minimum, ensuring that the model does not diverge. In conclusion, as evident from Figure 2.5, the optimal LR should be chosen from the point of loss descent to approximately one order of magnitude before the absolute minimum.

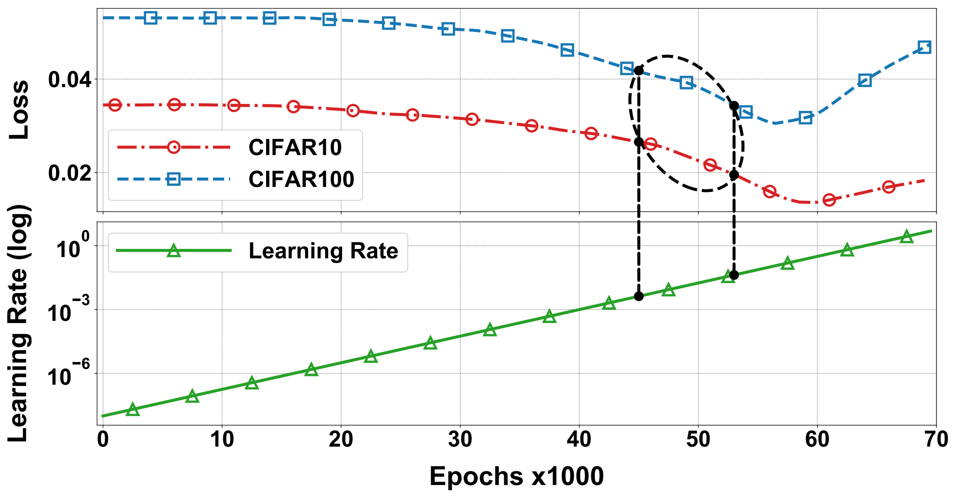


Fig. 2.5 LR finder executed on the ResNet50 model for the CIFAR10 and CIFAR100 datasets. The LR is plotted in logarithmic scale and the best LR is just before the loss minimum.

2.2.2 Fast Training Techniques

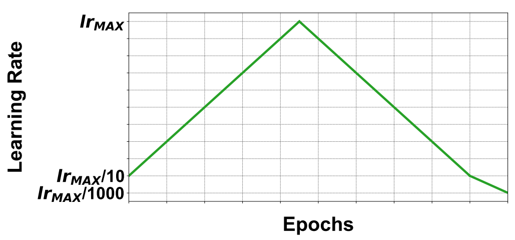
The most commonly used fast training techniques [63] involve varying the LR and other hyperparameters. Among these, the most advanced ones found in the literature include:

- One cycle policy
- Cyclical policy
- Warm restarts

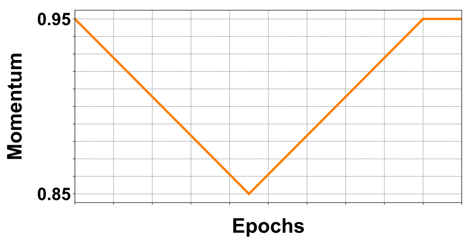
Each of these proposes a different technique for modifying the shape of the LR during training, and later on, we will see how each of them performs in a real-world scenario. Since these are all regularization techniques, to avoid interference, it is generally necessary to reduce other types of regularization for optimal results.

One Cycle Policy

The one cycle policy [64], as the name suggests, applies a single cycle throughout the entire training to both the LR and momentum. In Figures 2.6a and 2.6b, the shapes of LR and momentum during the one cycle policy are depicted, and it can be immediately observed that they have inverse shapes. This is because when LR increases, it is preferable to decrease the regularization brought by the momentum, and vice versa.



(a) LR shape starting from $\mathrm{LR}_{\mathrm{MAX}}/10$, reaching $\mathrm{LR}_{\mathrm{MAX}}$, and ending with $\mathrm{LR}_{\mathrm{MAX}}/1000$.



(b) Momentum shape, inverse of that of the LR, with a final constant value of 0.95.

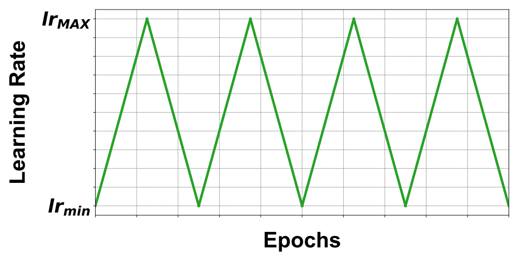
Fig. 2.6 One cycle policy.

To set up the one cycle policy, one starts with the LRF. After finding the optimal LR, LR_{MAX}, the initial LR value is set to 1/10 of the $\mathrm{LR}_{\mathrm{MAX}}$ (see Figure 2.6a). For the remaining 90% of the time, the LR shape becomes triangular, meaning it linearly increases from the initial value to the maximum $\mathrm{LR}_{\mathrm{MAX}}$ at 45% of the total time, and then linearly decreases back to the initial LR in the other 45% of the time. In the last part of the training, the LR transitions from the initial value, which was 1/10 of the maximum, to approximately $\mathrm{LR}_{\mathrm{MAX}}/1000$. In practice, in the last 10% of epochs, the LR value decreases rapidly by a factor of 100. This final part aims to reach the deep of the local minimum found and must be properly set, as if it lasts too long, it would lead to overfitting, whereas if it is too short, the accuracy would remain low.

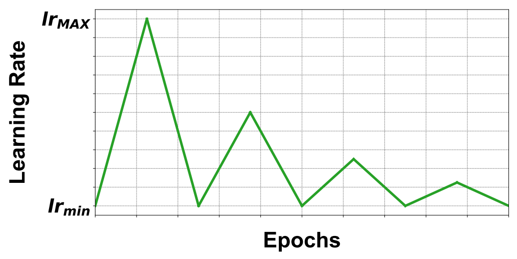
For the momentum, the same rule as LR is applied but in the opposite direction (see Figure 2.6b). Typically, a maximum value of 0.95 and a minimum value of 0.85 are used, resulting in an intermediate value of 0.90, which is commonly used for momentum. The only difference from LR is that in the final part, in the last 10% of epochs, the momentum remains constant at the value of 0.95 without further variation.

Cyclical Policy

The cyclical policy [64] (see Figures 2.7a and 2.7b) is based on a logic similar to that of the one cycle policy but is cyclical, meaning the shape of the LR oscillates multiple times between a maximum and a minimum value. This strategy can be useful when the training process of the DNN model is characterized by numerous local minimum points. The use of a cyclical LR allows the training to explore deeper minima, consequently improving the overall accuracy of the model. Additionally, the cyclical variation of the LR can help avoid issues such as overfitting or stagnation in suboptimal local minima.



(a) A triangular fixed shape.



(b) A triangular shape with fixed lower boundary.

Fig. 2.7 Cyclical policy.

The length of each cycle is defined as a multiple of one training epoch. During each cycle, the LR gradually increases from a minimum value to a maximum value, and then decreases back to the minimum value. This cycle is repeated multiple times during the training of the model. For optimal results, it is advisable to choose cycle lengths ranging from 4 to 20 times an epoch and perform at least 3-5 cycles during training to observe significant improvement compared to a constant LR. Excessive increase in the number of cycles may compromise the usefulness of the cycle itself, as the training would not have time to adapt to the variations in the LR.

The choice of maximum and minimum values of the LR is important for the success of the training. It is recommended to use the maximum LR found through techniques like the LRF (see Figure 2.8), approximately 1/10 before the minimum loss value, and set the minimum LR to a reasonable value within the loss descent region, i.e., after the initial plateau.

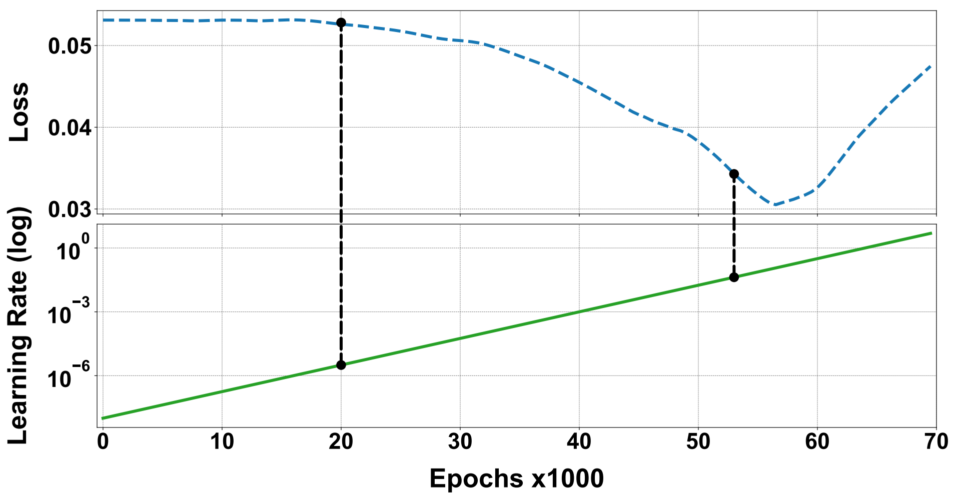
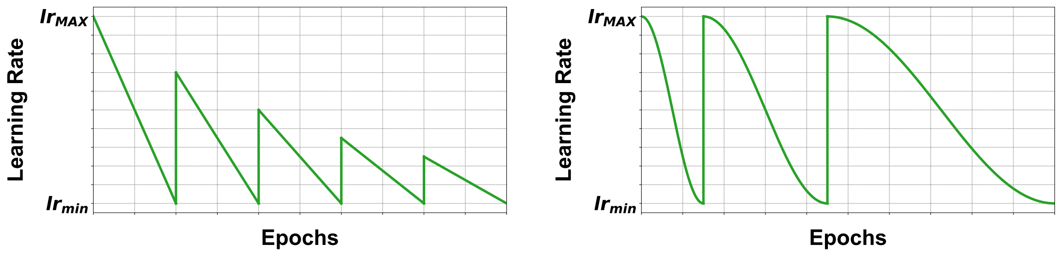


Fig. 2.8 LR boundary on the loss plot for the cyclical policy.

A variant of this technique involves creating cycles of equal length but with a decreasing maximum value, as shown in Figure 2.7b, to find deeper minima in local minima.

Warm Restarts

The warm restart [65, 66] (see Figures 2.9a and 2.9b) involves cyclically resetting the LR update process during training, but with sudden restarts from minimum to maximum, allowing the model to explore different optimization spaces more effectively.



(a) Linear decreasing shape. (b) Accordion-like sinusoidal shape.

Fig. 2.9 Warm restarts policy.

In practice, the LR value is regularly updated using a cyclic function, but instead of maintaining the LR's cyclic shape constantly, the warm restart technique involves periodically *restarting* the LR's cyclic function, bringing it back to its initial value or a predefined value. Once again, using an LR with warm restart prevents the model from getting stuck in suboptimal minima and enhances the model's ability to generalize to new data, reducing the risk of overfitting.

Other fast training methods

By altering the configurations of the hyperparameters or combining the techniques discussed earlier, it becomes feasible to devise novel training strategies for the LR that could potentially be more efficient than the original methods.

2.3 Datasets

For this research, two datasets were utilized: CIFAR10 [67] and CIFAR100 [68], which are similar but with a significant difference: the number of classes. Indeed, in this way, the effectiveness of the models in being trained on datasets with increasing learning difficulty was tested, where misclassifying classes is very simple, thus enabling the real testing of the model's robustness. The main characteristics of these two datasets are presented in Table 2.1.

Table 2.1 Summary of the main characteristics of the 2 datasets used.

Datasets	**CIFAR10**	**CIFAR100**
Total images	60000	60000
Train-set	50000	50000
Test-set	10000	10000
N° Classes	10	100
Dimensions	32x32	32x32
Colours	3 - RGB	3 - RGB
Classes Type	Objects	Objects

2.3.1 CIFAR10

CIFAR10 [67] (see Figure 3.6) consists of 60,000 color images sized 32x32 pixels, divided into 10 different classes, each containing 6,000 images. The 10 classes represent common objects such as airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is split into 50,000 training images and 10,000 test images. It is one of the most commonly used datasets for testing low-resolution image classification algorithms.

2.3.2 CIFAR100

CIFAR100 [68] is similar to CIFAR10 but contains 100 classes instead of 10. Each class contains 600 images. The 100 classes are divided into 20 superclasses, each containing 5 classes. For example, one of the superclasses could be *pets*, which includes classes like cats, dogs, and birds. Other superclasses include $wildanimals$, $fruitsandvegetables$, *vehicles*, etc. CIFAR-100 is designed to test classification algorithms that require greater variety and complexity in classes.

2.4 TensorFlow

TensorFlow [60] is an open-source library developed by Google for ML and AI. It is one of the most popular libraries for developing ML models, particularly for neural networks [69] and deep learning algorithms [70].

Its architecture is based on a computational graph, where nodes represent mathematical operations and edges represent tensors (or multidimensional arrays) flowing between operations. This approach makes TensorFlow highly efficient in terms of parallel and distributed execution across various hardware platforms, including CPUs, GPUs, and TPUs.

TensorFlow offers a wide range of functionalities, including model construction, training, evaluation, inference, and optimization. Additionally, it supports the TensorBoard tool, which allows for the visualization of data and many other parameters useful for optimizing workflow.

Furthermore, TensorFlow is highly flexible and supports a wide range of programming languages, including Python [71], C++, Java, and others, enabling developers to easily integrate TensorFlow into their existing projects.

2.5 Free Adversarial Training

In the first part of this research activity, we studied and tested the feasibility of fast adversarial training on an existing model, and we chose the Free Adversarial Training (FAT) [72].

In this model, there are two main parameters:

- m - also known as $free-m$, a parameter of FAT itself, indicating how many times a perturbation is applied for each minibatch. Setting m to 1 results in standard training.
- ε - the adversarial perturbation, which is the noise added to the data. Values that are too small are ineffective, while values that are too large would distort the data so much that it becomes recognizable even to a human.

First, the FAT model was analyzed to evaluate its baseline performance. Then, training was performed on the FAT ResNet50 [73, 74] model using the CIFAR10 and CIFAR100 datasets. The anticipated attack in the original paper is the Projected Gradient Descent (PGD) with a constant ε of 8.0. The model was trained on both natural and adversarial images for 80,000 epochs, and the final accuracy and loss [62] achieved for natural images is shown in Table 2.2.

Table 2.2 Final accuracy and loss on the original FAT.

Datasets	CIFAR10	CIFAR100
Accuracy	84.34%	59.89%
Loss	0.00562	0.01459

As shown in Figure 2.10, the training on CIFAR100 performs less effectively both in terms of accuracy and loss. This is due to the higher complexity of CIFAR100, which has 100 classes instead of 10. In both cases, however, the robust models manage to achieve fairly acceptable results on natural images, surpassing 50% accuracy.

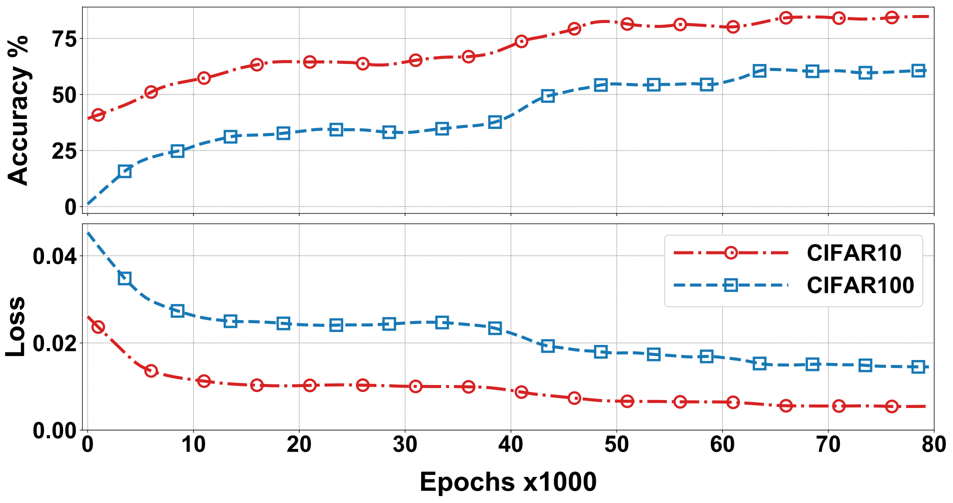


Fig. 2.10 Original FAT accuracy and loss on natural images.

2.5.1 Hyperparameters Setup

In Figure 2.11, you can see the shape of the original LR of FAT. As you can notice, it's not constant but has already been optimized as it follows a step function pattern outlined in Table 2.3.

To find the best LR, we used the LRF, resulting in the following maximum LR values for each dataset:

- CIFAR10 - 0.15
- CIFAR100 - 0.12

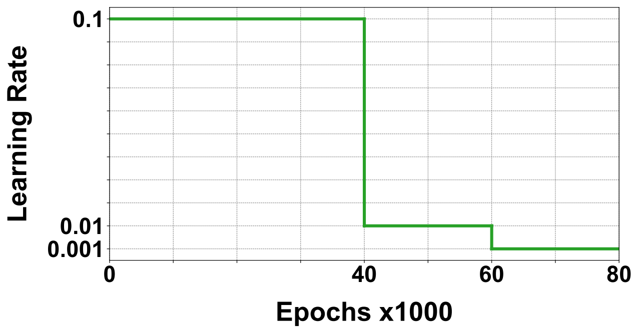


Fig. 2.11 FAT original 3-steps LR shape.

Table 2.3 FAT original 3-steps LR values during the training.

Epochs	LR
0 - 40.000	0.1
40.000 - 60.000	0.01
60.000 - 80.000	0.001

These values are higher than those of the original FAT, and will allow us to achieve better results as we'll see later on.

Regarding momentum, we chose to keep the constant value of 0.90 for all fast training techniques and to keep it variable, following the pattern shown in Figure 2.6b, between 0.85 and 0.95 only for the one cycle policy. For regularization and due to computational constraints of our computer, we set a value of 0.0002 for weight decay and 128 for batch size. The remaining hyperparameters were left unchanged from the original FAT to ensure the most similar conditions, aiming to speed up the model while maintaining robustness.

2.5.2 Optimization Results

In Figures 2.12 and 2.13, various LR optimization techniques applied to FAT with the CIFAR10 and CIFAR100 datasets can be seen.

All tests were conducted with models robust to PGD attacks and tested on the natural images of the datasets. To perform the tests, a Tesla K40c GPU was used, which takes about 5 hours to run 10,000 epochs. The original FAT lasted 80,000 epochs, equivalent to 40 hours of training. With the implemented fast training

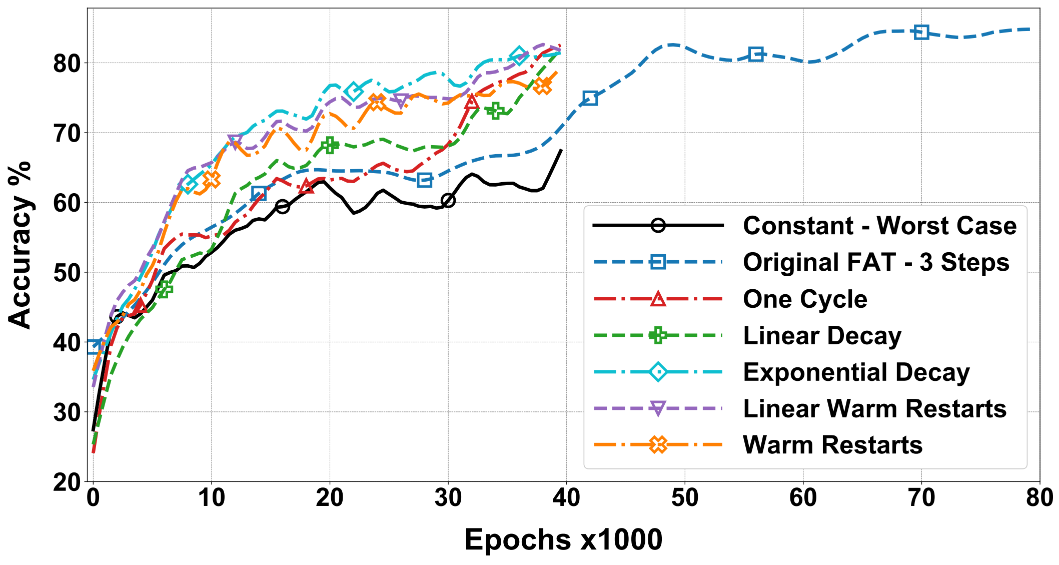


Fig. 2.12 FAT adversarial training of ResNet50 on the CIFAR10 and validation on natural images dataset with different LR techniques.

techniques, it can be seen how the same final accuracy of the original FAT is achieved in half the epochs. For a more comprehensive comparison, we also conducted a test with constant LR, which obviously performed less well than the 3-steps LR of the original FAT. Among the various techniques, the one cycle policy probably performed best in terms of accuracy and will therefore be used in the subsequent part of the research. From these results, it is clear how fine-tuning hyperparameters can lead to super-convergence in both standard training on natural images and adversarial training [43] without affecting the accuracy and robustness of the original models [75].

2.6 AccelAT Framework

The AccelAT framework was implemented to reduce model setup time. In fact, to use existing fast training techniques, one must spend a considerable amount of time setting all the parameters and conducting various tests to obtain the best configuration. This initial setup phase is not considered in the training time but is a considerable part of the total time to create a robust model. With the AccelAT framework, the model itself will decide when to adjust the LR for more efficient training, achieving equivalent or better results than existing fast training techniques.

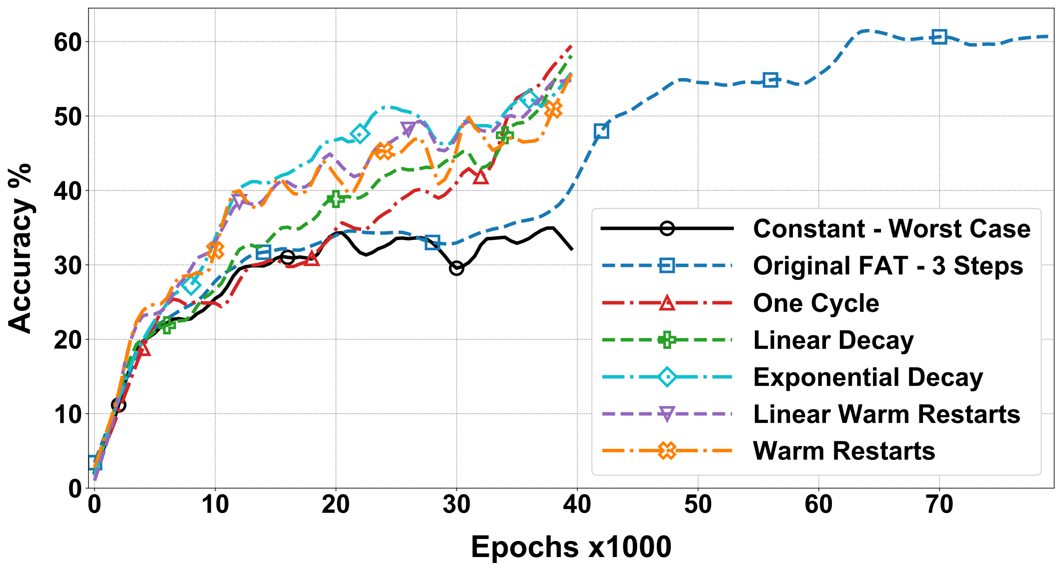


Fig. 2.13 FAT adversarial training of ResNet50 on the CIFAR100 and validation on natural images dataset with different LR techniques.

Looking at Algorithm 1, you can see how AccelAT works. With the LRF, we find the maximum usable LR, LR_{MAX}, and set the initial LR to that value. During training, if the accuracy does not improve for a certain period of time, the LR decreases to search for deeper minima, i.e., we calculate the gradient of accuracy to decide the value of the next LR. This process is repeated whenever accuracy stagnates in a plateau zone for a certain number of epochs, until the desired accuracy is reached (see Figure 2.14). Here's a step-by-step explanation of the framework:

- Search for LR_{MAX} with the LRF.
- Set the LR to the previously found LR_{MAX}.
- Training begins with the LR_{MAX} set.
- Calculate accuracy, and if it continues to increase, proceed without further modifications.
- If accuracy stagnates on a plateau, i.e., does not increase by a predetermined Δ_{acc} for a certain number of epochs n, then the LR is reduced by a percentage with a fixed coefficient p.
- Once the LR reaches a predetermined minimum value, LR_{min}, decided in advance, the remaining training continues with this LR value.

Algorithm 1 *AccelAT*

Require: Maximum LR - LR_{MAX}, minimum LR - LR_{min}, accuracy delta - Δ_{acc}, percentage reduction - p, number of cycles of interest - n, accuracy - acc, previous average accuracy - acc_{pre}

```
1: $LR \leftarrow LR_{MAX}$
2: for $e$ in $\overline{epochs}$ do
3:     if $(\overline{acc(e, e-n)} - acc_{pre}) < \Delta_{acc}$ then
4:         $LR \leftarrow LR \cdot p$
5:     end if
6:     if $LR < LR_{min}$ then
7:         $LR \leftarrow LR_{min}$
8:     end if
9:     $acc_{pre} \leftarrow \overline{acc(e, e-n)}$
10: end for
```

- If accuracy reaches an optimal value, then training terminates.

For instance, let's consider a value n of ten epochs to evaluate accuracy and a Δ_{acc} of 1%. Suppose we've determined a maximum LR of 0.01 and want to reduce it by approximately p = 10% each time we encounter a plateau area. We then initiate training for 100 epochs. We monitor the accuracy increase rate at each epoch. For example, in the first 40 epochs, if the accuracy increase is greater than or equal to 1% compared to the last ten epochs, then the LR remains fixed at 0.01. At the 41st epoch, if the accuracy hasn't increased by at least 1% in the last ten epochs, we reduce the LR by 10%, multiplying it by 0.9, thus obtaining a value of 0.009. We then resume training, and accuracy begins to rise again. Around the 70th epoch, if another plateau area is encountered, we again reduce the LR by 10%. Training continues until the end of the 100 epochs. At the end, the LR is lower than the maximum LR_{MAX}, allowing us to increase accuracy by delving deeper into the found local minimum. With our AccelAT framework, we obtain an adaptive LR based on the specific type of training we're conducting, i.e., based on the accuracy gradient, enabling us to overcome the limitations of a fixed LR, thereby ensuring more efficient training.

Utilizing the AccelAT framework does not yield a defined shape for the LR, but rather varies from training to training based on the model, dataset, and various parameters. Since the LR is reduced by a percentage, it can be said to assume a form similar to a scale, albeit logarithmic, as it decreases less and less as the value becomes smaller. Calculating the gradient each time during training slightly slows

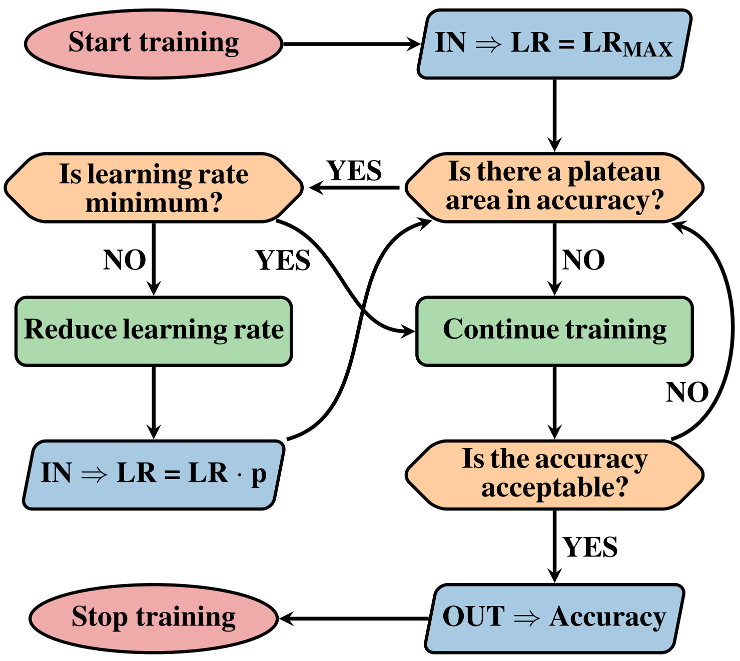


Fig. 2.14 AccelAT workflow.

down the process, but for cases with complex datasets, it allows for optimal LR to achieve the best results. Conducting preliminary analyses to choose the values of p, n, and Δ_{acc} enables obtaining the best efficiency of the AccelAT framework.

2.7 AccelAT Setup

To test the AccelAT framework, we attempted to recreate an environment similar to that used for FAT. We used pre-trained ResNet50 [73, 74] and MobileNet [76] models on ImageNet [77–79] and attacked them with LinfPGD and DeepFool [7] from the Foolbox library to make them robust, while CIFAR10 and CIFAR100 were used as datasets for training.

We tested all combinations of network/dataset/attack with each of the three different types of LR policy: constant, one cycle, and AccelAT. Thus, a total of 24 trainings were performed to verify the effectiveness of AccelAT, with each training taking approximately 2 hours with the available hardware. Using the LRF, we found a value of LR_{MAX} ranging from 0.001 to 0.0001 for all simulations, and the other hyperparameters were set to the following optimal values:

- Momentum - 0.9
- Batch size - 128
- Weight decay - 0.0002

The number of epochs was set to 50, using the logic of early stopping. In this way, we avoided overfitting [80] while still maintaining an adequate accuracy value. Finally, we set a value of 0.01 for the ε, the perturbation budget, and the algorithm iterated the process until fooling the DNN.

Figure 2.15 shows a summary diagram of the experimental setup for the AccelAT project, where the software, hardware, and parameters used are displayed.

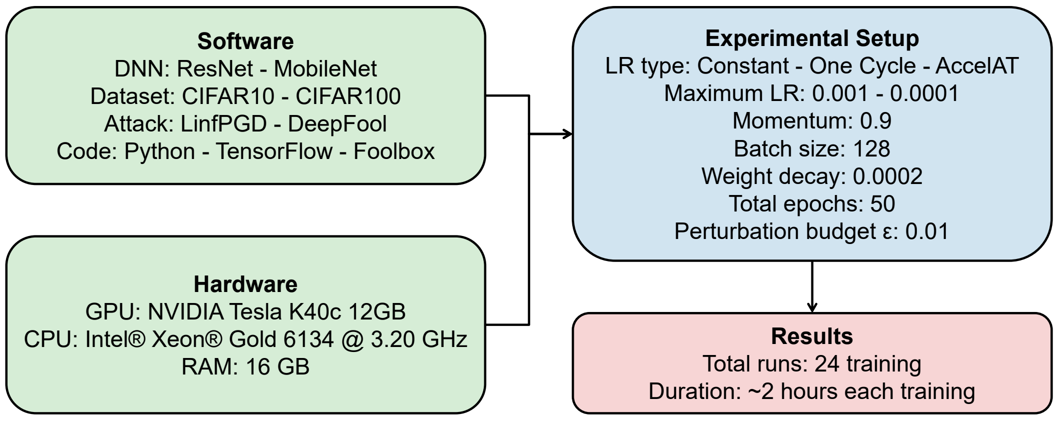

Software
DNN: ResNet - MobileNet
Dataset: CIFAR10 - CIFAR100
Attack: LinfPGD - DeepFool
Code: Python - TensorFlow - Foolbox
Hardware
GPU: NVIDIA Tesla K40c 12GB
CPU: Intel® Xeon® Gold 6134 @ 3.20 GHz
RAM: 16 GB
Experimental Setup
LR type: Constant - One Cycle - AccelAT
Maximum LR: 0.001 - 0.0001
Momentum: 0.9
Batch size: 128
Weight decay: 0.0002
Total epochs: 50
Perturbation budget ε: 0.01
Results
Total runs: 24 training
Duration: ~2 hours each training

Fig. 2.15 AccelAT experimental setup.

2.7.1 Models

The ResNet50 and MobileNet models are two convolutional models with different purposes. Indeed, the former is a state-of-the-art Deep Neural Network (DNN) used for complex applications, while the latter is a lighter network suitable for deployment on portable devices, as the name suggests.

ResNet

ResNet [73, 74] is a family of DNNs developed by Microsoft Research. It introduces the concept of *skipconnection* or *shortcutconnection*, which involves adding direct connections that skip one or more layers instead of passing through them. These connections enable the direct flow of gradients during the backpropagation process, facilitating the training of very DNNs. The key idea is to address the problem of network degradation, where performance deteriorates when additional layers are added to the network. Thanks to skip connections, ResNet can train neural networks with hundreds of layers, achieving better performance compared to shallower architectures.

MobileNet

MobileNet [76] is another family of convolutional neural networks developed by Google, primarily designed for computer vision applications on mobile and embedded devices, where computational resources and energy are limited. The architecture of MobileNet is based on the idea of separating standard convolutions into *depthwise* convolutions and *pointwise* convolutions. Depthwise convolutions apply a separate kernel to each input channel, significantly reducing the number of operations. Pointwise convolutions are 1x1 convolutions that combine the results of depthwise convolutions to create more complex representations. This design allows MobileNet to achieve a good trade-off between accuracy and computational efficiency, making it ideal for devices with limited resources.

2.7.2 Attacks

The LinfPGD and DeepFool attacks were implemented using the Foolbox library, and in both cases, they were optimized to deceive the model without the perturbed data being identifiable by a human subject.

LinfPGD

LinfPGD [81] is an algorithm used in attacking ML algorithms, particularly in the contexts of adversarial attacks. This algorithm employs projected gradient descent to generate ℓ_∞ perturbations, meaning perturbations bounded by the ℓ_∞ norm, which represents the maximum deviation allowed for each pixel in the image. The goal of LinfPGD is to modify the input in order to deceive the ML model, causing it to make an error in classifying the image.

DeepFool

DeepFool [82] is an algorithm used to generate adversarial attacks on neural networks. The approach of DeepFool is based on the idea of finding the direction in which a small perturbation of the input can move the input instance across the decision boundary of the model, i.e., the hyperplane separating the different output classes. The algorithm iteratively computes the minimum perturbation required to move the input instance beyond the decision boundary, thus producing an image that looks very similar to the original but induces a misclassification by the model. DeepFool is designed to be efficient and produces minimal perturbations that are imperceptible to humans but can easily deceive ML models.

2.7.3 Resources

The hardware used to conduct all the tests consisted of a NVIDIA Tesla K40c 12 GB GPU, an Intel® Xeon® Gold 6134 @ 3.20 GHz CPU, and 16 GB of RAM.

The AccelAT code was entirely written in Python with the help of libraries such as TensorFlow and Foolbox and can be found at the following GitHub address: https://github.com/farzadnikfam/AccelAT.

2.8 Results

In Figures 2.16 to 2.18, 9 out of the 24 conducted tests are shown, featuring different combinations of network/dataset/attack. The simulations were carried out by training the models on adversarial images and then evaluating them on the natural training-set, natural test-set, adversarial training-set, and adversarial test-set. Since the obtained results were proportionally comparable, only those tested step by step on the adversarial training set were reported in Figures 2.16 to 2.18.

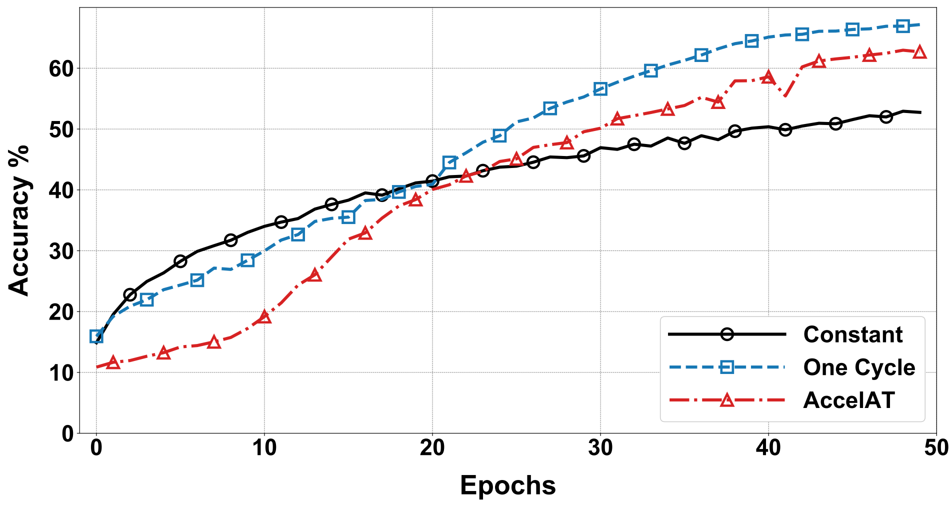


Fig. 2.16 ResNet model trained on the CIFAR10 dataset, attacked with DeepFool and tested on the adversarial training-set.

2.9 Discussion

As evident from the graphs in Figures 2.16 to 2.18, our AccelAT framework yields results similar to those of the one cycle policy, which, as seen in Section 2.5.2, had results similar to other fast training techniques.

In certain situations, AccelAT performs even better than one cycle, such as in the early epochs in Figures 2.17 and 2.18. In fact, with MobileNet being a less complex model than ResNet, a high initial value of LR, as in AccelAT, allows for quickly reaching deeper minima before delving deeper with lower LR values. On the other hand, with complex models like ResNet (see Figure 2.16), AccelAT exhibits more difficulty in the initial part of training, then effectively surpasses the constant LR. In this case, starting with lower LR values, as done by the one cycle method, to evaluate the presence of multiple local minima allows the model to learn more rapidly.

AccelAT remains a promising method that should be further developed in the future, as it still has some limitations. It performs well on simple models but still requires slightly longer computational calculations compared to other methods, as it needs to compute the accuracy gradient each time. Additionally, the initial setup to find LR_{MAX}, LR_{min}, and especially the decay factor p and the Δ_{acc}, is not straightforward. Despite these limitations, AccelAT can still serve as a solid

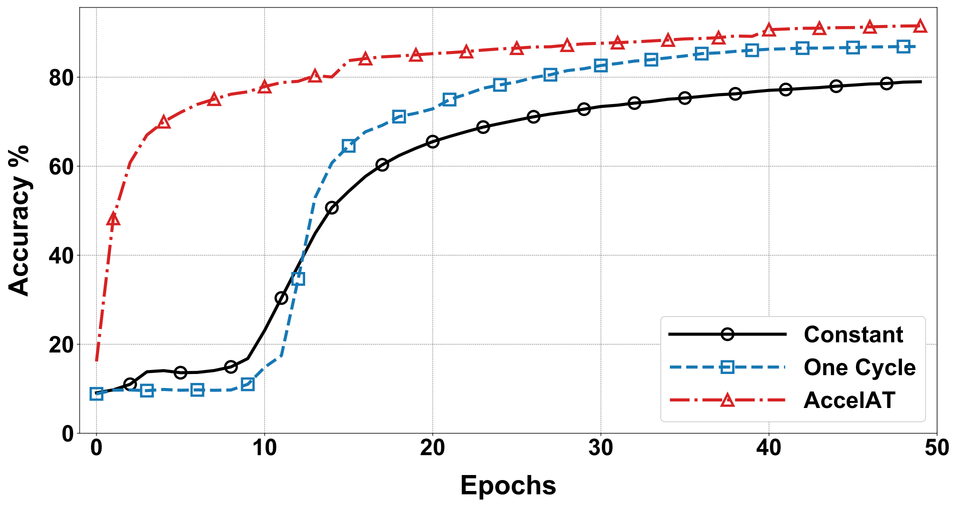


Fig. 2.17 MobileNet model trained on the CIFAR10 dataset, attacked with LinfPGD and tested on the adversarial training-set.

foundation for future research in the field of fast training, especially in fast adversarial training.

2.10 Conclusion

Nowadays, robustness is crucial for DNNs; however, achieving fast and robust training still poses a challenge. This research has demonstrated that advanced fast training techniques can also be applied to adversarial training, yielding significant improvements with similar robustness. Additionally, the AccelAT framework has been introduced to adjust the LR during training based on the accuracy gradient. Experimental results have shown that AccelAT performs on par with other LR-based fast training techniques, surpassing not only training with a constant LR, which usually reaches a sub-optimal local minimum, but also optimized techniques like the one cycle policy under certain conditions. AccelAT is also effective for training with large datasets, where a variable LR during execution is essential to better adapt the DNN model and enable more effective learning. However, there are still some challenges, especially regarding the initial setup and training with complex models like ResNet, where the presence of many local minima can create difficulties. Therefore, future work will aim to further develop AccelAT and explore its combination with the optimization of other hyperparameters such as momentum.

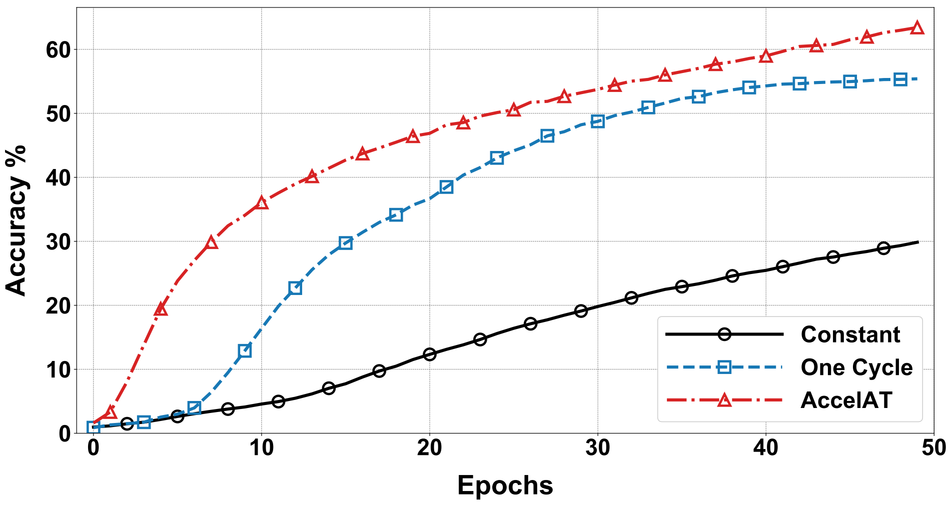


Fig. 2.18 MobileNet model trained on the CIFAR100 dataset, attacked with DeepFool and tested on the adversarial training-set.

While some hyperparameters, such as batch size [83], are determined by hardware conditions, optimizing others can lead to significant advantages during training. In conclusion, this work has demonstrated the feasibility of robust and fast models that are likely to be widely adopted in the near future.

Chapter 3

SpyKing

This work [84] aims to present a comparison between Deep Neural Networks (DNNs) and Spiking Neural Networks (SNNs) in the field of Homomorphic Encryption (HE). As seen before, various techniques exist to maintain the privacy of the processed data, and one of the oldest, evolving over time with mathematics, is encryption. In this case, we specifically used Fully Homomorphic Encryption (FHE) to encrypt the data and DNN and SNN models of LeNet5 to compare the performance of the results with unencrypted data, as well as between standard [85] and spiking [86] models. The findings of this work indicate that privacy preservation [21–23] through encryption is computationally expensive, but the results are satisfactory. Especially under certain conditions, SNNs [87] can outperform DNNs. This example of maintaining the privacy of sensitive data represents an essential part of a niche sector that is gradually expanding, considering the increasing need for protected data in various application domains in the future [88].

Figure 3.1 shows a summary diagram of the research work carried out for SpyKing, from the inputs used to the results obtained.

3.1 Spiking Neural Network

A SNN [86, 89], or pulse neural network, is a type of neural architecture inspired by the functioning of biological neurons in the brain. Unlike traditional neural networks, such as DNNs, SNNs use a communication model based on pulses or spikes, representing signals sent by neurons (see Figure 3.2).

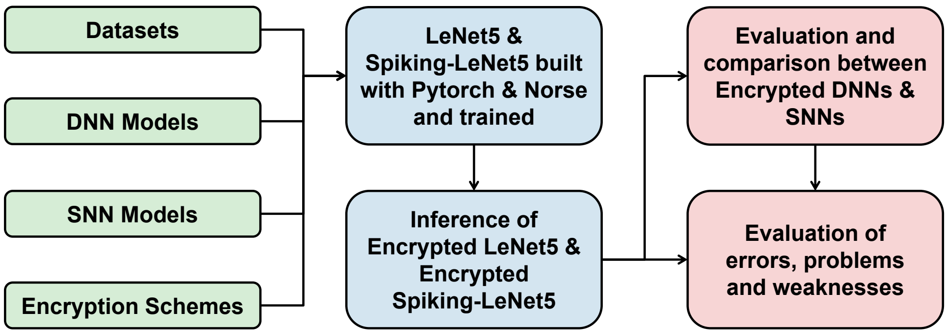

Datasets
DNN Models
SNN Models
Encryption Schemes
LeNet5 & Spiking-LeNet5 built with Pytorch & Norse and trained
Inference of Encrypted LeNet5 & Encrypted Spiking-LeNet5
Evaluation and comparison between Encrypted DNNs & SNNs
Evaluation of errors, problems and weaknesses

Fig. 3.1 A summary flowchart of the SpyKing research project.

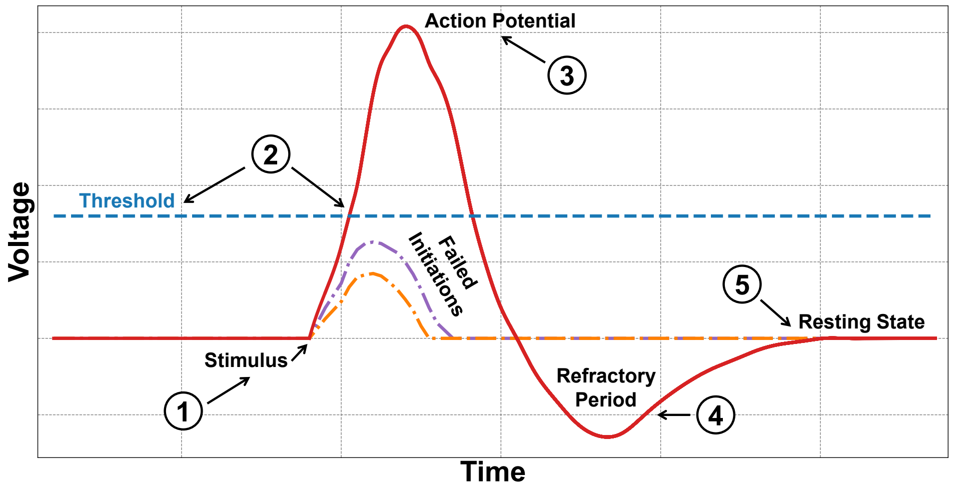


Fig. 3.2 An example of a spiking neuron that activates only after receiving the necessary charge to surpass the threshold, undergoing a refractory period before returning to a resting state.

In traditional models, artificial neurons, after receiving input, apply a transformation using an activation function and produce a continuous output. In spiking neurons [90], communication occurs through discrete pulses or *spikes* [91]. Each neuron accumulates input signals (see pointer ① - Figure 3.2) over time and generates a spike when a certain threshold is exceeded (see pointer ② - Figure 3.2). Synapses, the connections between neurons, are determined by weights that can change during the learning process, increasing or decreasing the probability of a neuron firing. The activation of each neuron is based on both spatial and temporal factors. Each neuron depends on its position and connections with nearby neurons, and its activation is influenced by the time of charge before firing, which typically cannot be less than a certain threshold. When a neuron releases a spike (see pointer ③ - Figure 3.2) after its potential reaches the threshold, its charge is reset, and the neuron enters a passive waiting phase, the refractory period (see pointer ④ - Figure 3.2) before the resting state (see pointer ⑤ - Figure 3.2).

This construction allows SNNs to closely mimic the real and biological functioning of the human brain. Considering the latency times between spikes due to charge times, SNNs also enable more energy-efficient models [92].

3.1.1 Leaky Integrate-and-Fire

In the context of SNNs, Leaky Integrate-and-Fire (LIF) is a specific type of spiking neuron model. To better understand how it works, here's an explanation of the acronym LIF:

- Integration - the LIF neuron accumulates input over time. Each time it receives a spike, its *charge* increases. This accumulation of charge represents how the neuron *integrates* information over time.
- Firing - when the neuron's charge reaches a certain threshold, the neuron *fires* a spike. This simulates the idea of activation in the context of neural networks.
- Leak - the *Leak* indicates that, over time, the neuron's charge tends to dissipate or lose energy. This process of charge loss over time is implemented to simulate the dynamic and adaptive nature of biological neurons.

So, the LIF model is essentially a way to describe how a spiking neuron accumulates and releases energy over time, reflecting some features of biological neurons. Its simplicity makes it computationally efficient, and the addition of the leak component makes it more adaptable and realistic compared to some more basic spiking neuron models.

There are other SNN neuron models, such as Hodgkin-Huxley [93], which are based on very complex differential calculations, making it challenging to construct large computational models due to lower efficiency. Considering the trade-off between efficiency and reliability, the LIF neuron model was chosen for the creation of SpyKing.

3.1.2 Norse Library

Norse [94] is a Python [71] library that leverages the advantages of bio-inspired neural components. Norse extends the PyTorch [95] library for implementing DNN with primitives for biologically inspired neural components.

With Norse, it is possible to start with basic PyTorch DNN models and create their spiking versions. As we will see later in this work, the Lenet5 model, implemented in PyTorch, was used, and with the use of Norse, the spiking version, Spiking-Lenet5 [96–100], was created.

LIF Parameters

The LIF parameters within Norse are specific configurations that define the behavior of LIF neurons in SNNs. These parameters include:

- τ_{syn}^{-1} - represents the inverse of the synaptic time constant, determining how quickly the synaptic input decays over time.
- τ_{mem}^{-1} - represents the inverse of the membrane time constant, influencing the rate of decay of the neuron's membrane potential without input.
- v_{leak} - specifies the leak potential of the neuron, indicating the resting potential of the membrane when there is no synaptic input or other stimuli.
- v_{th} - defines the threshold potential of the neuron. An action potential is generated when the membrane potential reaches or exceeds this threshold.
- v_{reset} - represents the reset potential of the neuron. After firing an action potential, the membrane potential is reset to this value.

These parameters play a critical role in determining the dynamics of the LIF neuron in the SNN. They govern how the neuron integrates and responds to incoming synaptic input, as well as when it generates an action potential. The specific values of these parameters can be adjusted to achieve the desired behavior, providing control over the firing rate and responsiveness of the neuron within the network.

Encoders

SNNs require an encoder as they process temporal data represented as spikes. Since most ML datasets lack a temporal representation, adding an encoding phase is essential to provide the necessary temporal component. The encoder transforms input data into sequences of spikes, subsequently processed by the SNN as tensors with binary values. The Constant Current LIF encoder, found in the Norse library, is an encoding method that transforms constant input into constant voltage spikes. During a specified time interval, known as seq_{length}, spikes are simulated based on the input current. This approach allows Norse to operate on sparse input data as a sequence of binary tensors, optimizing the SNN's processing efficiency. If the potential reaches the required threshold during seq_{length}, the spike is released.

Simply put, seq_{length} represents the number of iterations a SNN needs to biologically simulate the human brain. Consequently, the seq_{length} value serves as a temporal multiplier. For example, if a DNN takes time x to be trained or evaluate data, the corresponding SNN model will take a time equivalent to x multiplied by seq_{length}. This poses a temporal efficiency challenge intrinsic to SNNs, and this temporal factor that elongates computation times cannot be eliminated. The only solution to address this issue is to choose a seq_{length} value that is balanced, accurately simulating SNNs without excessively extending computation times.

3.2 Homomorphic Encryption

HE [24] is an advanced form of cryptography that enables operations on encrypted data without the need for prior decryption. This technique is particularly useful when preserving data privacy [101] during processing in environments where security is crucial, such as in cloud computing. Examining Figure 3.3 provides a clearer understanding of how HE works. Initial data is encrypted with a public key [102, 103] that anyone can obtain. Once encrypted, the data is sent to the server where it undergoes manipulation and specific computations. Finally, the results, still encrypted, are sent back to the client, who is the only entity capable of decrypting them using a secret key known only to them. In this manner, the entire data processing is kept secret, and only the client knows the original data and the final results.

The security of HE relies on the strength of the encryption algorithm and the secrecy of the keys. Unfortunately, there are limitations because computations on encrypted data are much more time, memory, and energy consuming, and therefore are only executed when necessary.

The term *homomorphic* indicates that operations performed on encrypted data correspond to the same operations executed on unencrypted data. Homomorphism can take various forms, including partially HE [104], somewhat HE [105] and fully HE (FHE) [106–110]. Each of these allows different levels of computation on encrypted data.

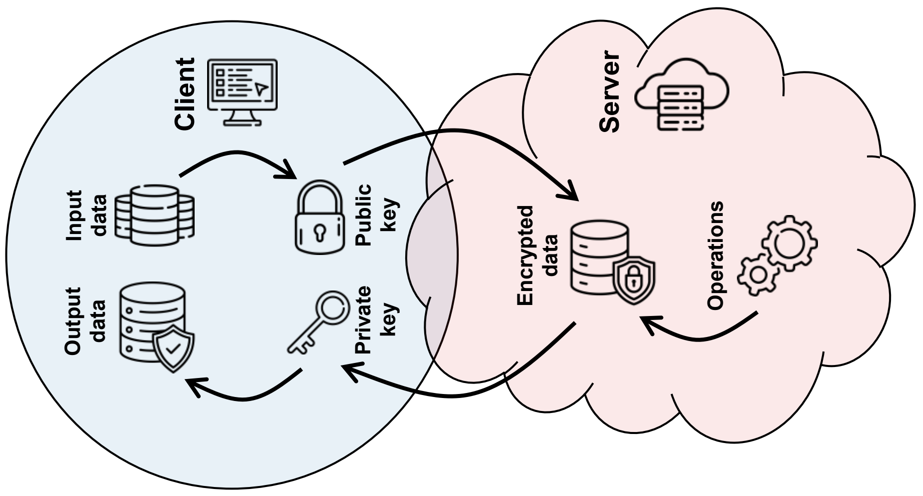


Fig. 3.3 A HE scheme with a clear separation between client and server, where the data and results in plaintext are visible only to the client.

3.2.1 Fully Homomorphic Encryption

FHE [106–110] is the most comprehensive form of HE, as it enables both addition and multiplication operations on encrypted data. One of the widely used schemes in this field is the Brakerski/Fan-Vercauteren (BFV) scheme, which we utilized in our framework. To better understand its functioning, equations from Equation (3.1) to Equation (3.7) illustrate a simplified example of how achieving the same result is possible even after a transformation. In this example, the structure of the functions has been designed to only preserve addition, but in FHE, the same logic applies to multiplications.

Let's consider the Equation (3.1) and apply a homomorphic transformation (encryption) as depicted in Equation (3.2). To verify if the transformation occurred homomorphically, we choose two random values for x and y, as represented in Equation (3.3). Adding our values to Equation (3.1), we obtain Equation (3.4), from which, by performing the calculations, we arrive at Equation (3.5). At this point, we introduce the transformation from Equation (3.2), as mentioned earlier, resulting in Equation (3.6). By performing the last simple calculation, we can observe in Equation (3.7) that the result is equal on both sides, despite the transformation in between. Hence, we can conclude that this transformation was homomorphic concerning additions.

FHE applies the same logic to encryption with more complex calculations, making both additions and multiplications homomorphic. Unfortunately, in the case of non-linear calculations, FHE is not supported. Data must be decrypted before proceeding with the computation; otherwise, there is a risk of obtaining completely incorrect and unreadable results.

$$f(2x+3y) = f(2x) + f(3y) \tag{3.1}$$

$$f(z) = 6z \tag{3.2}$$

$$\begin{cases} x = -2 \\ y = +4 \end{cases} \tag{3.3}$$

$$f(2\cdot(-2)+3\cdot(+4)) = f(2\cdot(-2)) + f(3\cdot(+4)) \tag{3.4}$$

$$f(+8) = f(-4) + f(+12) \tag{3.5}$$

$$[6\cdot(+8)] = [6\cdot(-4)] + [6\cdot(+12)] \tag{3.6}$$

$$+48 = +48 \tag{3.7}$$

3.2.2 Pyfhel Library

Pyfhel [111] is a Python [71] library that allows encryption using various schemes and a wide range of data while maintaining limited computational capabilities based on the chosen data type. It supports the BFV scheme and implementation on neural networks. Unfortunately, it was not designed exclusively for the field of ML. Despite being usable for neural networks, it has not been optimized for this purpose and only leverages the CPU, not utilizing the hardware acceleration possible with the GPU. Considering that encryption is already inefficient and computationally intensive, the inability to use the GPU on large datasets, such as those in Artificial Intelligence (AI), inevitably leads to very long computing processes.

HE Parameters

The implementation of the BFV scheme in Pyfhel relies on three key elements:

- m - represents the degree of the polynomial modulus, impacting computational capabilities and the security level of the encryption system.
- t - denotes the plaintext modulus, determining the size and precision of the ciphertext values for the plaintext.
- q - represents the ciphertext modulus, influencing the size of the ciphertext values and affecting the security and computational performance of the encryption scheme.

Balancing security and computational efficiency in FHE operations becomes possible by selecting appropriate values for these parameters. Pyfhel provides an easy-to-use interface for working with the BFV scheme, enabling encryption, computation, and decryption of data with concise and comprehensible code.

Another crucial element to consider is the Noise Budget (NB), which denotes the maximum amount of disturbance or error that can be introduced during the encryption and computation process without compromising the accuracy of the results. In operations performed on encrypted data, activities such as addition and multiplication can accumulate disturbance, putting at risk the accuracy of the results when decrypted. The NB sets a limit on how much disturbance can be tolerated before the decrypted results become unreliable. It is imperative to carefully manage and continuously monitor the NB throughout the entire computation process to ensure the security and integrity of cryptographic operations.

3.3 Datasets

The MNIST [112], FashionMNIST [113], and CIFAR10 [67] datasets are popular datasets used in the ML community for training and evaluating algorithms in computer vision. Table 3.1 shows the main characteristics of each dataset.

Table 3.1 Summary of the main characteristics of the 3 datasets used.

Datasets	**MNIST**	**FashionMNIST**	**CIFAR10**
Total images	70000	70000	60000
Train-set	60000	60000	50000
Test-set	10000	10000	10000
N° Classes	10	10	10
Dimensions	28x28	28x28	32x32
Colours	1 - Grayscale	1 - Grayscale	3 - RGB
Classes Type	Number 0-9	Clothes	Objects

3.3.1 MNIST

MNIST [112] is one of the most widely used datasets in ML. It consists of grayscale images of handwritten digits from 0 to 9. It represents a standard among ML datasets and is often used for basic testing. Accuracy on this dataset can easily reach high values close to 100%. In Figure 3.4, there are examples extracted from the dataset representing all 10 classes. The images appear pixelated as they are in a very small format, namely 28x28 pixels.

3.3.2 FashionMNIST

FashionMNIST [113] is a dataset containing images of clothing items. It was created as a more complex alternative to the MNIST dataset, as it maintains the same structure but instead of handwritten digits, it features grayscale images of clothing

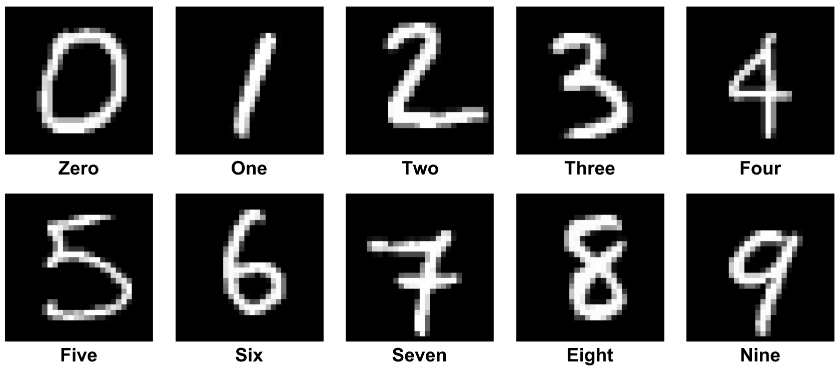


Fig. 3.4 An example for each class of the MNIST dataset.

items. Similarly, the dataset contains 70,000 images, divided into 60,000 for the training set and 10,000 for the test set, with a size of 28x28 pixels as seen in the examples in Figure 3.5.

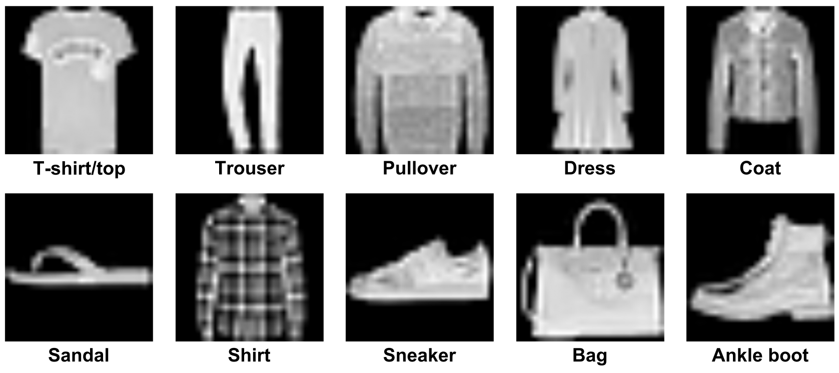


Fig. 3.5 An example for each class of the FashionMNIST dataset.

3.3.3 CIFAR10

CIFAR10 [67] is an RGB color image dataset with dimensions of 32x32 pixels, which are slightly larger than those in the MNIST group, and consists of 10 classes of common objects and animals (see Figure 3.6). Among the datasets we used, this is the most complex, and indeed, the accuracy of various models on this dataset generally falls well below 90%. In terms of total size, it is slightly smaller than MNIST, with 50,000 images for the training set, 10,000 for the test set, totaling 60,000 data points.

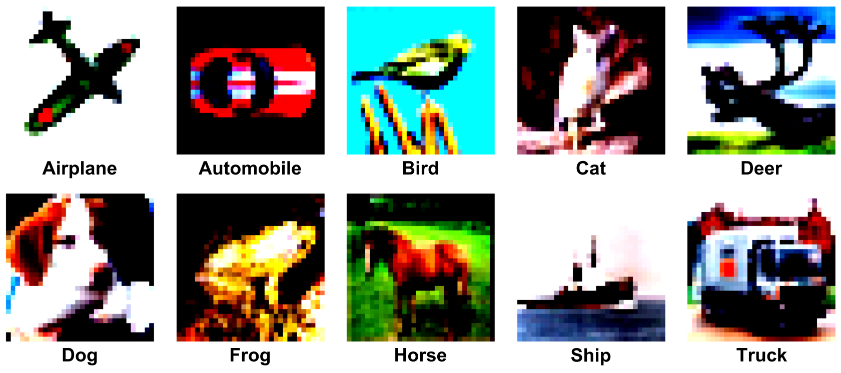


Fig. 3.6 An example for each class of the CIFAR10 dataset.

3.4 PyTorch

PyTorch [95] is an open-source library for ML developed by Facebook. It is designed to provide a flexible and scalable platform for developing AI models and is fully compatible with the Python [71] programming language.

One of PyTorch's key features is its support for automatic gradient computation, which significantly simplifies the implementation of algorithms by allowing users to modify the network structure during program execution.

PyTorch offers various tools such as data loading and preprocessing, neural network creation, GPU training support, and integration with third-party libraries, such as Norse, which allows the creation of SNNs.

The syntax of PyTorch is clear and intuitive, making it a popular choice among ML developers. PyTorch is widely used in both academic and industrial settings for various applications, including image classification, natural language processing, computer vision, and more. Given its widespread adoption, PyTorch is continuously growing and evolving.

3.5 LeNet5 Model

LeNet5 is a Convolutional Neural Network (CNN) model developed by Yann LeCun and his team at Bell Labs in the 1990s [114]. It was one of the first CNN models to be widely used for image classification and played a crucial role in the early advances of deep learning. Since then, LeNet5 has served as a foundational model for the development of more advanced CNN architectures and has found applications in various domains, including character recognition, object detection, and facial recognition.

LeNet5 is composed of convolutional, pooling, and fully connected layers. The convolutional layers extract features from the input images using convolutional filters. The pooling layers reduce the dimensionality of the extracted features while preserving their essential information. Finally, the fully connected layers classify the features and produce the output predictions. During training, the LeNet5 model utilizes error backpropagation to update the weights of the convolutional filters and fully connected layers in order to minimize the loss function [62] and improve the network's performance.

In Figure 3.7, there is a 3D reconstruction of LeNet5 for the classification of the FashionMNIST and MNIST datasets. Each color represents the various layers of the model and their respective matrix dimensions, from the input image to the final output classification. In Figures 3.8 and 3.9, you can see the 2D models with an explanation of the various steps for the MNIST dataset family and for CIFAR10.

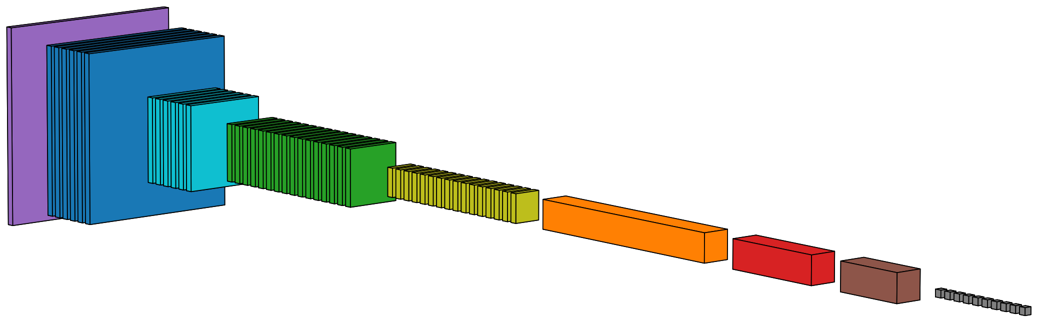
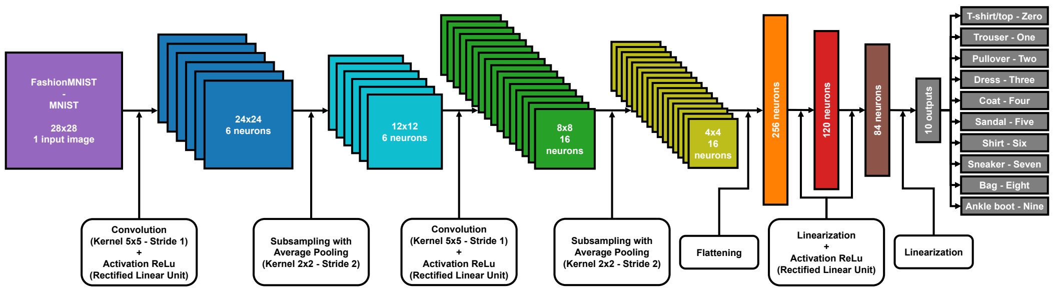


Fig. 3.7 LeNet5 model for the FashionMNIST and MNIST datasets. Each color represents a layer and the squares are the matrices dimensions during the training. For a better explanation see Figures 3.8 and 3.9.

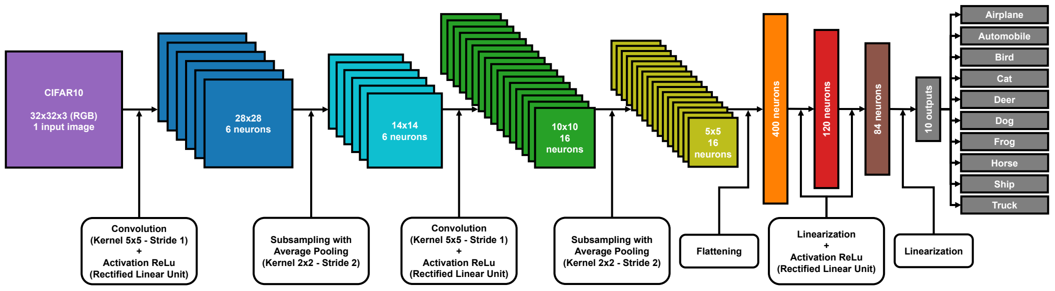
3.5.1 Spiking-LeNet5 Model

The Spiking-LeNet5 model [96–100] was built based on the standard LeNet5 model. We then integrated the Norse python library with the PyTorch library to obtain the spiking version. The LeNet5, which processed each dataset differently, was modified by replacing the Rectified Linear Unit (ReLu) activation commands with the LIF



FashionMNIST
-
MNIST
28x28
1 input image
24x24
6 neurons
12x12
6 neurons
8x8
16
neurons
4x4
16
neurons
256 neurons
120 neurons
84 neurons
10 outputs
T-shirt/top - Zero
Trouser - One
Pullover - Two
Dress - Three
Coat - Four
Sandal - Five
Shirt - Six
Sneaker - Seven
Bag - Eight
Ankle boot - Nine
Convolution (Kernel 5x5 - Stride 1) + Activation ReLu (Rectified Linear Unit)
Subsampling with Average Pooling (Kernel 2x2 - Stride 2)
Convolution (Kernel 5x5 - Stride 1) + Activation ReLu (Rectified Linear Unit)
Subsampling with Average Pooling (Kernel 2x2 - Stride 2)
Flattening
Linearization + Activation ReLu (Rectified Linear Unit)
Linearization

Fig. 3.8 LeNet5 model with each layer and matrix size for FashionMNIST and MNIST training.



CIFAR10
32x32x3 (RGB)
1 input image
28x28
6 neurons
14x14
6 neurons
10x10
16
neurons
5x5
16
neurons
400 neurons
120 neurons
84 neurons
10 outputs
Airplane
Automobile
Bird
Cat
Deer
Dog
Frog
Horse
Ship
Truck
Convolution (Kernel 5x5 - Stride 1) + Activation ReLu (Rectified Linear Unit)
Subsampling with Average Pooling (Kernel 2x2 - Stride 2)
Convolution (Kernel 5x5 - Stride 1) + Activation ReLu (Rectified Linear Unit)
Subsampling with Average Pooling (Kernel 2x2 - Stride 2)
Flattening
Linearization + Activation ReLu (Rectified Linear Unit)
Linearization

Fig. 3.9 LeNet5 model with each layer and matrix size for CIFAR10 training.

activation from the Norse library, and the entire model was then placed in a timed sequence controlled by seq_{length} to allow for neuron firing.

In Figure 3.10, you can see how an image from the dataset appears during the spiking temporal sequence with seq_{length} set to 30, in this case it is the Ankle Boot, label 9 in the FashionMNIST dataset (see Figure 3.5). You can observe how the image only appears in certain parts because only some neurons fire at a time. In Figure 3.11, there is a comparison between the original image and the sum of the previous timed images. The final result is not identical, but it can be noted that during the temporal sequence, more or less all neurons fire, allowing the image to still be recognized.

3.6 Training Phase

For the training phase, we set the parameters optimally to increase accuracy. The PyTorch library was chosen for defining the model, as the Norse library relies on PyTorch, allowing us to create both the LeNet5 and the Spiking-LeNet5 models

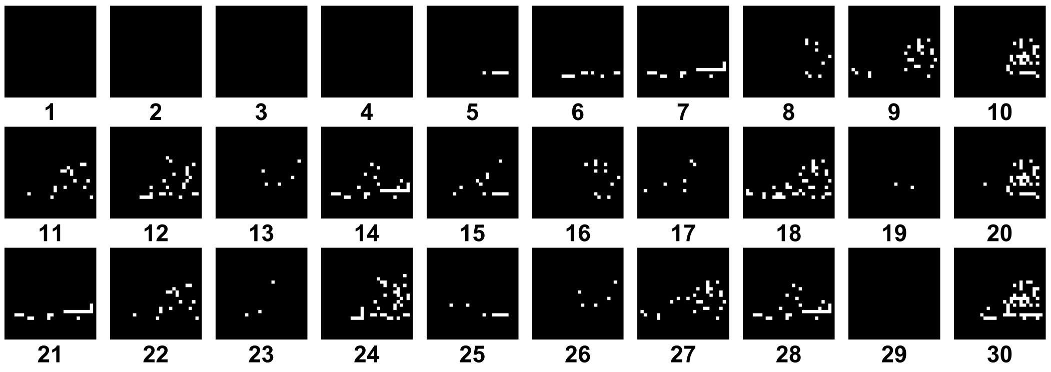


Fig. 3.10 In the Spiking-LeNet5 the neurons fire randomly during the seq_{length} and the result is each time a portion of the total image, in this case it is the Ankle Boot, label 9 in the FashionMNIST dataset (see Figure 3.5).

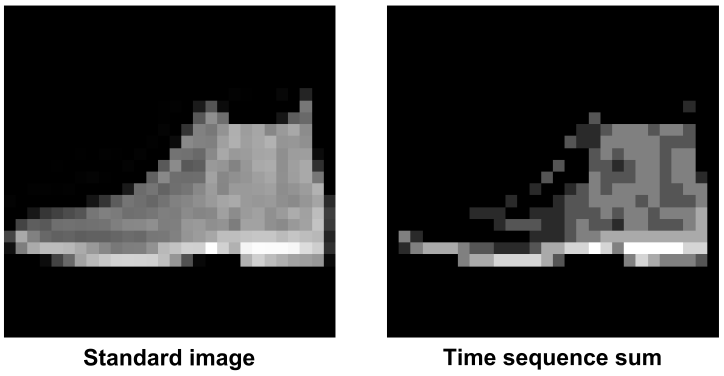


Fig. 3.11 On the left we have the native Ankle Boot (Label 9 in the FashionMNIST dataset (see Figure 3.5)) image, while on the right there is the sum of the temporal sequence seq_{length} of Figure 3.10.

based on PyTorch. The selected parameters can be seen in Table 3.2, and Figure 3.12 provides a summary diagram of the experimental setup for the SpyKing project. The learning rate was chosen using the learning rate finder technique, while the number of epochs was selected using early stopping to prevent overfitting.

In Figure 3.13, we can observe the accuracy and loss [62] for each epoch during the training on the FashionMNIST dataset, comparing LeNet5 and Spiking-LeNet5 [118]. Additionally, the dashed lines illustrate how, for each model, validation has slightly lower performance compared to training. Furthermore, we can notice that the spiking model has slightly lower final accuracy compared to the non-spiking model, which is due to the intrinsic complexity of the spiking version. Also, the computation time of the spiking model differs from that of LeNet5; on average, the

Table 3.2 Training phase parameters.

Parameters	**LeNet5**	**Spiking-LeNet5**
Learning Rate	0.001	0.001
Epochs	20	20
Batch Size	256	256
Optimizer	Adam [115]	Adam [115]
Loss	Cross Entropy [116]	Negative Log-Likelihood [117]
seq_{length}	-	30
τ_{syn}^{-1}	-	200
τ_{mem}^{-1}	-	100
v_{leak}	-	0
v_{th}	-	0.5
v_{reset}	-	0
Encoder	-	Constant Current LIF

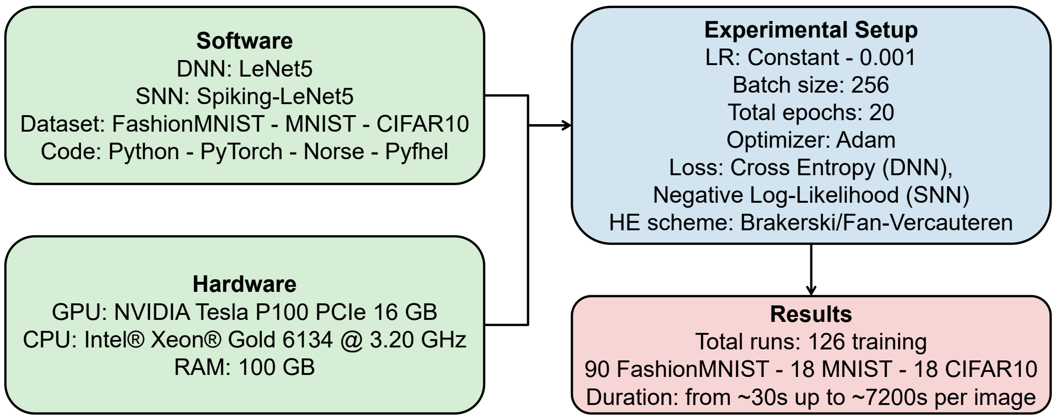

Software
DNN: LeNet5
SNN: Spiking-LeNet5
Dataset: FashionMNIST - MNIST - CIFAR10
Code: Python - PyTorch - Norse - Pyfhel
Hardware
GPU: NVIDIA Tesla P100 PCIe 16 GB
CPU: Intel® Xeon® Gold 6134 @ 3.20 GHz
RAM: 100 GB
Experimental Setup
LR: Constant - 0.001
Batch size: 256
Total epochs: 20
Optimizer: Adam
Loss: Cross Entropy (DNN),
Negative Log-Likelihood (SNN)
HE scheme: Brakerski/Fan-Vercauteren
Results
Total runs: 126 training
90 FashionMNIST - 18 MNIST - 18 CIFAR10
Duration: from ~30s up to ~7200s per image

Fig. 3.12 SpyKing experimental setup.

spiking model takes the same time as LeNet5 multiplied by the value of seq_{length}. The respective training graphs for the MNIST and CIFAR10 datasets are visible in Figures A.1 and A.2.

As can be observed, the final accuracy achieved by the standard LeNet5 model varies across the datasets: it's around ≈99% for MNIST, ≈80-90% for FashionMNIST, and ≈60-70% for CIFAR10. This disparity among the datasets arises from practical reasons; MNIST, being the simplest dataset, exhibits the highest accuracy. FashionMNIST is similar to MNIST but with slightly more complex classes to distinguish. Lastly, CIFAR10 is a dataset with 3 RGB channels and consequently much more complex than the previous two, resulting in lower model accuracy on this

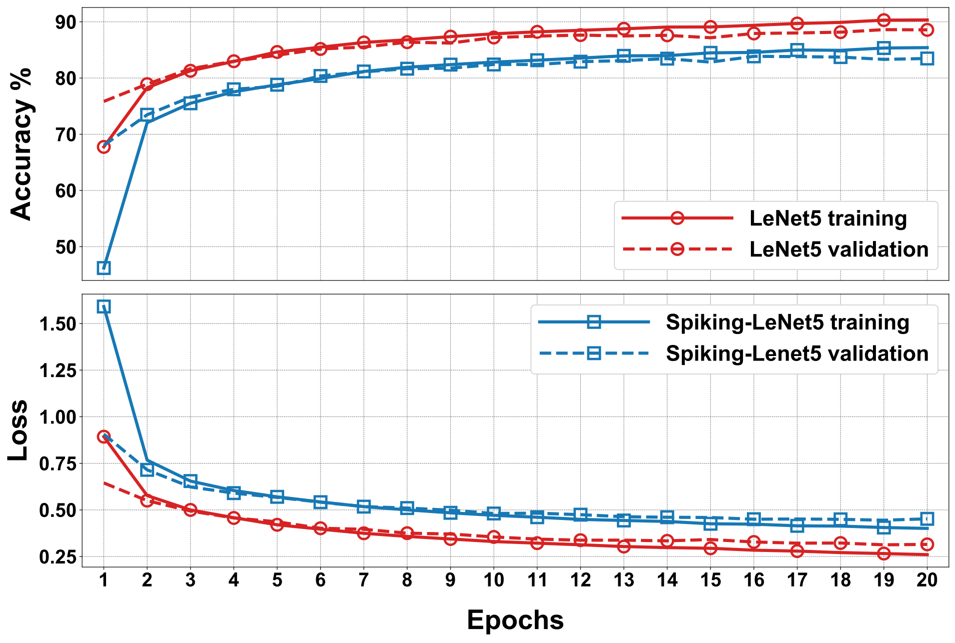


Fig. 3.13 Accuracy and loss during training and validation of LeNet5 and Spiking-LeNet5 for the FashionMNIST dataset. The figure shows accuracy and loss values across different training epochs.

dataset as well. Given the differences between the datasets and the repeated trials for all, to avoid overwhelming subsequent paragraphs, the following discussion will focus more on the FashionMNIST dataset, which has intermediate complexity, while the results of the other two datasets can be found in the Appendix A.

3.6.1 Parameters Selection

After training, in order to proceed with encryption, it is necessary to define the parameters of the BFV scheme: m, t, and q. The parameter m must be a power of 2 greater than 1024 and is directly proportional to the NB. Values of m that are too high would lead to overly complex computational calculations, while low values would be too insecure. Values of m equal to 2048 or higher do not significantly alter the results but exponentially increase computation times. Therefore, we performed these calculations only on the FashionMNIST dataset, and the results are visible in Figure 3.14.

The value of t can also vary, but too low values lead to incorrect encryption, while too high values degrade the results, making them unreadable. For the FashionMNIST dataset, we evaluated a variation of t on 15 values between 10 and 500000, noting that after the value of 5000 there are no significant differences. Consequently, for the other two datasets, we evaluated the results between 10 and 5000.

The last parameter is q, but it is related to m in determining the value of NB and is automatically calculated by the Pyfhel library to obtain adequate encryption.

The NB also allows for a certain tolerance in operations before the results degrade too much, and therefore sometimes it needs to be *recharged* by decrypting and encrypting again. However, this did not affect our results because, as we will see later, due to nonlinear calculations in the models, we were forced to decrypt and encrypt multiple times. Consequently, the value of NB was replenished each time, allowing us to perform subsequent encrypted calculations without issues.

3.6.2 Encryption

In Table 3.3, there are comparisons for the computation times for each dataset. As can be seen, with the hardware available to us and with a value of m set to 1024, it takes approximately 1 second to encrypt an image from the FashionMNIST and MNIST datasets for the LeNet5 model, and about 30 seconds for the Spiking-LeNet5 model. After that, it takes another 30 seconds for evaluating the image on the encrypted LeNet5 model and about 15 minutes on the encrypted Spiking-LeNet5 model. The value of 15 minutes is obtained by multiplying the 30 seconds taken by LeNet5 by the value of seq_{length}, which in our case is 30. It can also be noted that increasing the value of m results in an exponential increase in computation time, while the variation in the parameter t has no significant effect.

In Table 3.4, there is an estimation of the execution time, based on the same hardware, for other types of models, considering only the FashionMNIST dataset. As can be seen, the time is proportional to the number of parameters handled by the model itself, and even with models slightly more complex than LeNet5, much longer computation times are obtained.

Table 3.3 Encryption and execution time for each image with respect to the variation of the model and the m parameter from 1024 to 4096.

Datasets	**Time (seconds)**	**LeNet5**			**Spiking-LeNet5**		
		1024	2048	4096	1024	2048	4096
Fashion MNIST	Encryption	1	2	8	30	60	240
	Plaintext execution	0.03	0.03	0.03	1	1	1
	Encrypted execution	30	60	240	900	1800	7200
MNIST	Encryption	1	2	8	30	60	240
	Plaintext execution	0.03	0.03	0.03	1	1	1
	Encrypted execution	30	60	240	900	1800	7200
CIFAR10	Encryption	2	4	16	60	120	480
	Plaintext execution	0.07	0.07	0.07	2	2	2
	Encrypted execution	60	120	480	1800	3600	14400

3.6.3 Resources

The hardware resources available for conducting the experiments consisted of a NVIDIA Tesla P100 PCIe 16 GB GPU, an Intel® Xeon® Gold 6134 @ 3.20 GHz CPU, and 100 GB of RAM.

The code (available at this GitHub address: https://github.com/farzadnikfam/SpyKing) was entirely written in Python with the help of various libraries, including PyTorch, Norse, and Pyfhel.

3.7 Results

In Figure 3.14, all the numerical data in percentage of the results obtained on the FashionMNIST dataset are presented in the form of a matrix. The simulations were conducted on 15 variations of t ranging from 10 to 500,000 and with 3 variations of m: 1024, 2048, and 4096. The calculations were performed for both LeNet5 and Spiking-LeNet5 models and were divided based on accuracy between plaintext and encrypted models. Since, as can be seen, the results with m set to 4096 are identical to those with m set to 2048, for both the standard and spiking models, the case with m set to 4096 will not be considered from now on.

Table 3.4 Prediction time for each image of the FashionMNIST dataset reported in seconds for each model with $m = 1024$. The long processing time of encrypted data are due to the complexity of the encrypted computations and it also depends on the complexity of each model (N° of parameters).

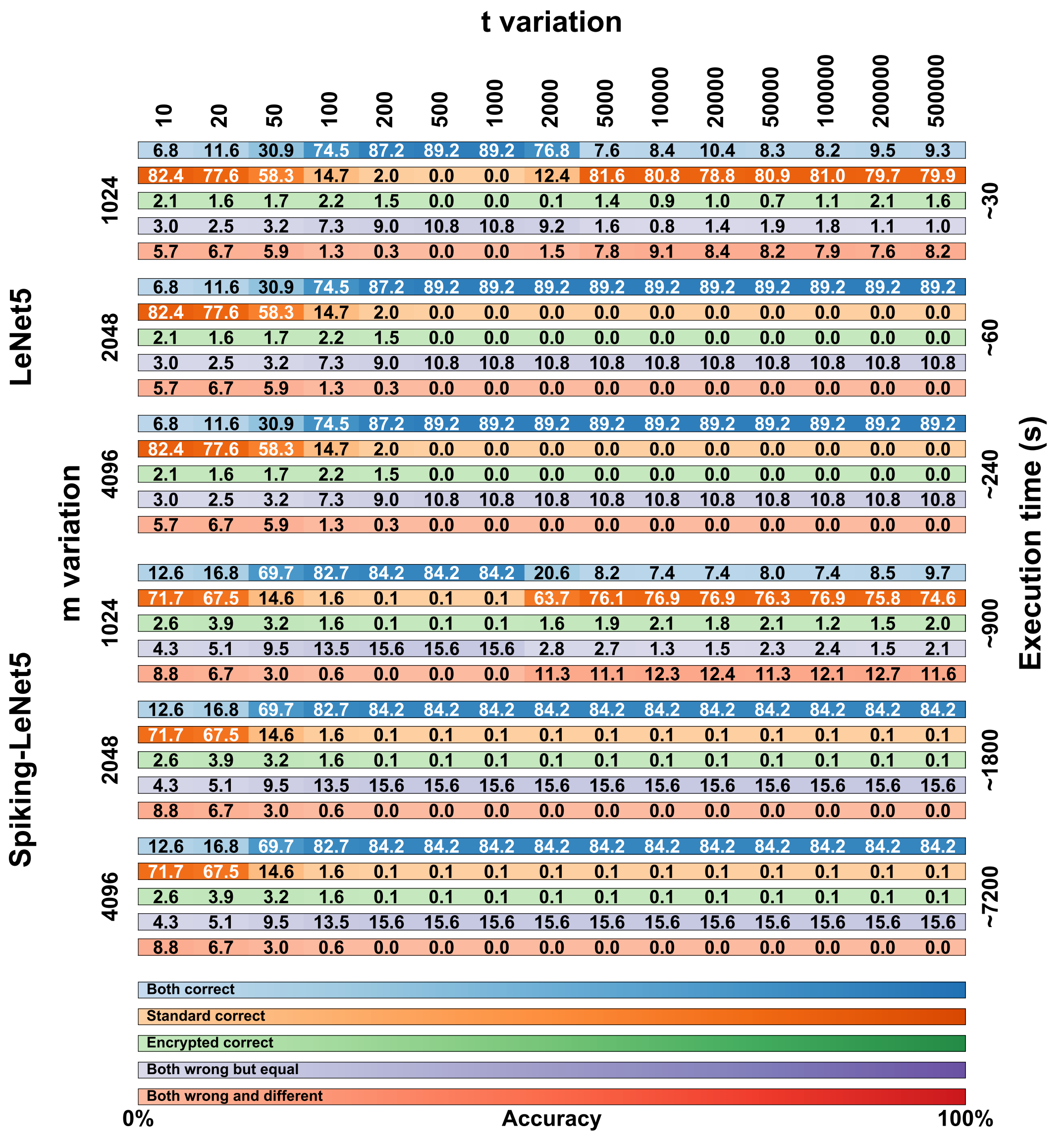
	Time (seconds)	**LeNet5** [114]	**AlexNet** [119]	**VGG16** [120]	**ResNet50** [73, 74]
Complexity		60k	60M	138M	23M
Standard	Encryption	1	60	140	20
	Plaintext execution	0.03	30	70	10
	Encrypted execution	30	30k	70k	10k
Spiking	Encryption	30	1.8k	4.2k	600
	Plaintext execution	1	1k	2.1k	300
	Encrypted execution	900	900k	2.1M	300k

In Figures 3.15 to 3.18, the visual representation of the same matrices can be seen with bar graphs to better understand the results. The results in matrix form for the MNIST and CIFAR10 datasets are in Figures A.3 to A.6, and the respective bar graphs have been grouped with those of FashionMNIST in Figure A.7 for better comparison.

To better understand how to read the matrices and bar graphs, here is an explanation of the colors:

- **Blue - both correct** - represents the percentage of images classified correctly by both the plaintext and encrypted models.
- **Orange - standard correct** - indicates the percentage of data classified correctly by the plaintext model but not by the encrypted one. It can be noticed that by adding the percentages of blue and orange colors, the same accuracy value is always obtained, whether changing m or changing t. This data represents the accuracy value of validation during training, which in the case of FashionMNIST corresponds to 89.2% for LeNet5 and 84.3% for Spiking-LeNet5.
- **Green - encrypted correct** - this percentage is the inverse counterpart of the previous one, meaning the images were classified correctly by the encrypted model but not by the plaintext model. The percentages are generally low and

almost insignificant, as this occurs because the encrypted model classifies differently from the plaintext model, which is incorrect, but by pure coincidence chooses the correct label. Therefore, this small percentage has no statistical value but is merely coincidental, as the encrypted model should classify like the plaintext model, even if the latter is wrong.



t variation

m variation			10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000	200000	500000	Execution time (s)
LeNet5	1024	🟦	6.8	11.6	30.9	74.5	87.2	89.2	89.2	76.8	7.6	8.4	10.4	8.3	8.2	9.5	9.3	~30
		🟧	82.4	77.6	58.3	14.7	2.0	0.0	0.0	12.4	81.6	80.8	78.8	80.9	81.0	79.7	79.9	
		🟩	2.1	1.6	1.7	2.2	1.5	0.0	0.0	0.1	1.4	0.9	1.0	0.7	1.1	2.1	1.6	
		🟪	3.0	2.5	3.2	7.3	9.0	10.8	10.8	9.2	1.6	0.8	1.4	1.9	1.8	1.1	1.0	
		🟥	5.7	6.7	5.9	1.3	0.3	0.0	0.0	1.5	7.8	9.1	8.4	8.2	7.9	7.6	8.2	
	2048	🟦	6.8	11.6	30.9	74.5	87.2	89.2	89.2	89.2	89.2	89.2	89.2	89.2	89.2	89.2	89.2	~60
		🟧	82.4	77.6	58.3	14.7	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
		🟩	2.1	1.6	1.7	2.2	1.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
		🟪	3.0	2.5	3.2	7.3	9.0	10.8	10.8	10.8	10.8	10.8	10.8	10.8	10.8	10.8	10.8	
		🟥	5.7	6.7	5.9	1.3	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
	4096	🟦	6.8	11.6	30.9	74.5	87.2	89.2	89.2	89.2	89.2	89.2	89.2	89.2	89.2	89.2	89.2	~240
		🟧	82.4	77.6	58.3	14.7	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
		🟩	2.1	1.6	1.7	2.2	1.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
		🟪	3.0	2.5	3.2	7.3	9.0	10.8	10.8	10.8	10.8	10.8	10.8	10.8	10.8	10.8	10.8	
		🟥	5.7	6.7	5.9	1.3	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Spiking-LeNet5	1024	🟦	12.6	16.8	69.7	82.7	84.2	84.2	84.2	20.6	8.2	7.4	7.4	8.0	7.4	8.5	9.7	~900
		🟧	71.7	67.5	14.6	1.6	0.1	0.1	0.1	63.7	76.1	76.9	76.9	76.3	76.9	75.8	74.6	
		🟩	2.6	3.9	3.2	1.6	0.1	0.1	0.1	1.6	1.9	2.1	1.8	2.1	1.2	1.5	2.0	
		🟪	4.3	5.1	9.5	13.5	15.6	15.6	15.6	2.8	2.7	1.3	1.5	2.3	2.4	1.5	2.1	
		🟥	8.8	6.7	3.0	0.6	0.0	0.0	0.0	11.3	11.1	12.3	12.4	11.3	12.1	12.7	11.6	
	2048	🟦	12.6	16.8	69.7	82.7	84.2	84.2	84.2	84.2	84.2	84.2	84.2	84.2	84.2	84.2	84.2	~1800
		🟧	71.7	67.5	14.6	1.6	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	
		🟩	2.6	3.9	3.2	1.6	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	
		🟪	4.3	5.1	9.5	13.5	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	
		🟥	8.8	6.7	3.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
	4096	🟦	12.6	16.8	69.7	82.7	84.2	84.2	84.2	84.2	84.2	84.2	84.2	84.2	84.2	84.2	84.2	~7200
		🟧	71.7	67.5	14.6	1.6	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	
		🟩	2.6	3.9	3.2	1.6	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	
		🟪	4.3	5.1	9.5	13.5	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	15.6	
		🟥	8.8	6.7	3.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

🟦 Both correct
🟧 Standard correct
🟩 Encrypted correct
🟪 Both wrong but equal
🟥 Both wrong and different

0% Accuracy 100%

Fig. 3.14 Comparison matrix for *t* and *m* variation for the FashionMNIST dataset on encrypted LeNet5 and Spiking-LeNet5 models.

- **Purple - both wrong but equal** - represents the case where the encrypted model and the plaintext model coincide but have not classified the correct label. This data is important because it shows how the encrypted model has functioned correctly by mimicking the plaintext model, even if the initial classification was incorrect.

- **Red - both wrong and different** - this last situation shows the case where both the encrypted and plaintext models have made mistakes and are different from each other. So, the label has not been correctly classified by either of the two models, and furthermore, the encrypted one has not copied the plaintext one. This percentage represents the worst-case scenario where nothing has worked as it should.

3.8 Discussion

To better discuss the results obtained in the previous section, we can refer to Figure 3.19, where the most relevant data has been presented in the form of graphs. Specifically, we compared, varying t, the accuracy of the LeNet5 and Spiking-LeNet5 models in both plaintext and encrypted versions, with the parameter m set to 1024 and 2048. To simplify, we essentially graphically represented the accuracy previously marked in **Blue - both correct**, i.e., when the encrypted model achieved the same results as the plaintext model and both coincided with the correct labels.

As we can see, both the standard and spiking versions reach approximately maximum accuracy, that is, the validation accuracy during training, with t values ranging from 200 to 1000. From this value onwards, the models with m set to 1024 show results that degrade quickly, while models with m set to or higher than 2048 maintain maximum accuracy. Apart from this difference between m set to 1024 and higher values, there are no other differences in the initial part, but especially high values of t are not so relevant because they indicate a high level of encryption that can increase computational costs or degrade data. The most important part is the part of the graph representing the lower t values, those below 200, where we can see how the spiking model performs significantly better than the standard model. Of course, these results are limited by the fact that the final accuracy of the validation of the spiking model is lower than that of the LeNet5 even in the plaintext version, which is why we created the graph in Figure 3.20.

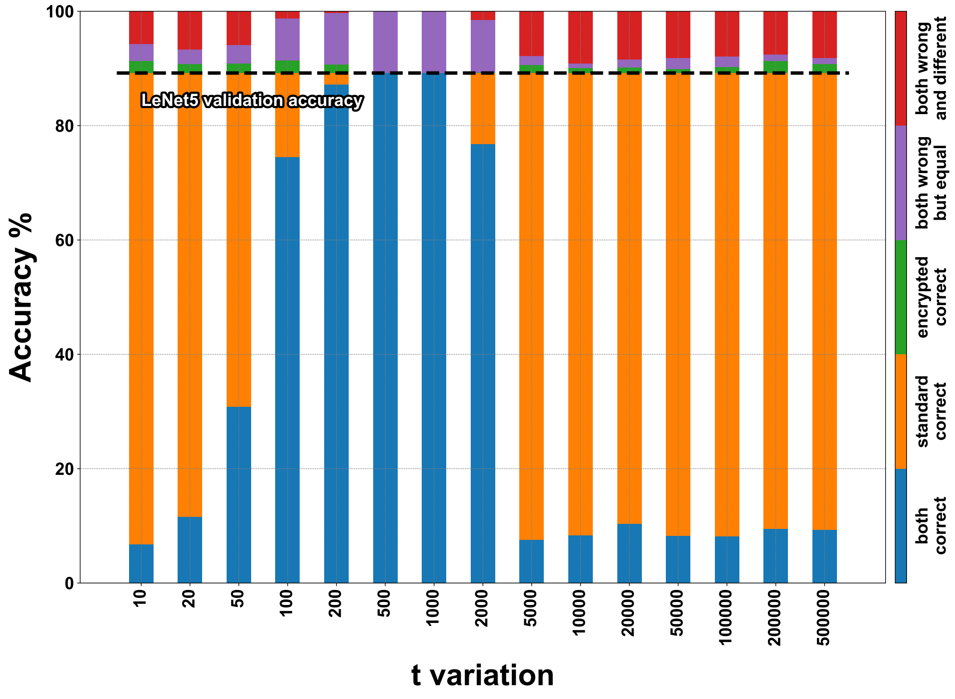


Fig. 3.15 FashionMNIST accuracy on encrypted LeNet5 for t variation with m set to 1024.

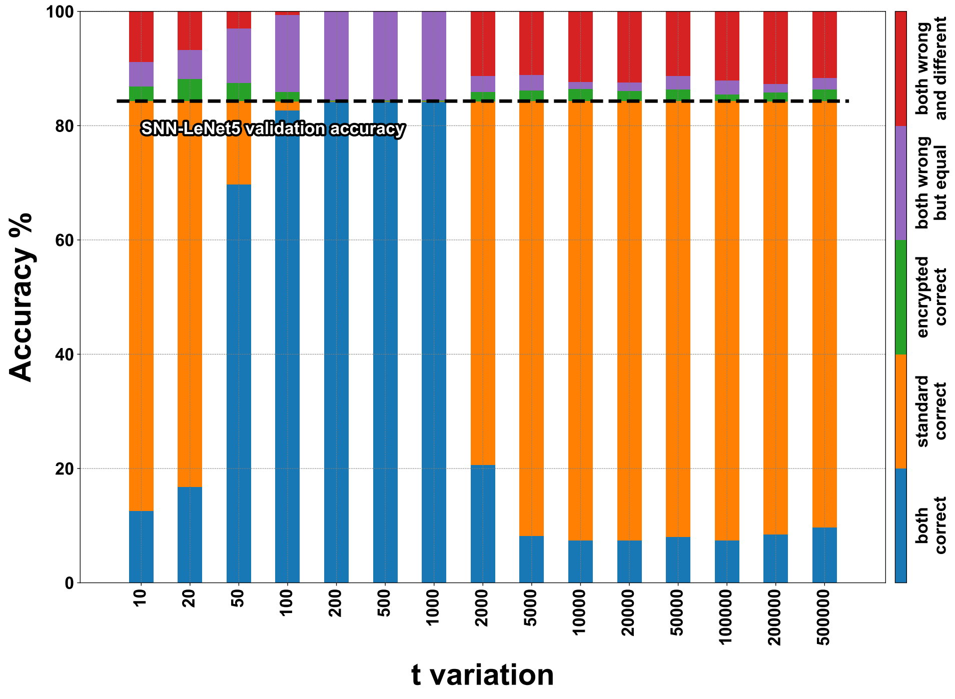


Fig. 3.16 FashionMNIST accuracy on encrypted Spiking-LeNet5 for t variation with m set to 1024.

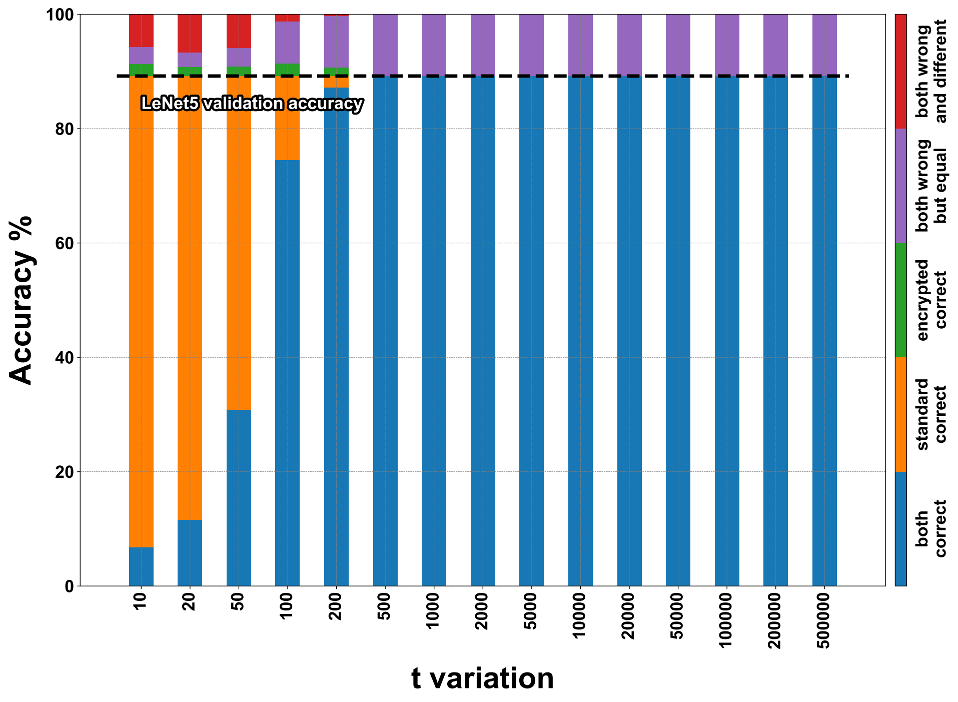


Fig. 3.17 FashionMNIST accuracy on encrypted LeNet5 for t variation with m set to 2048.

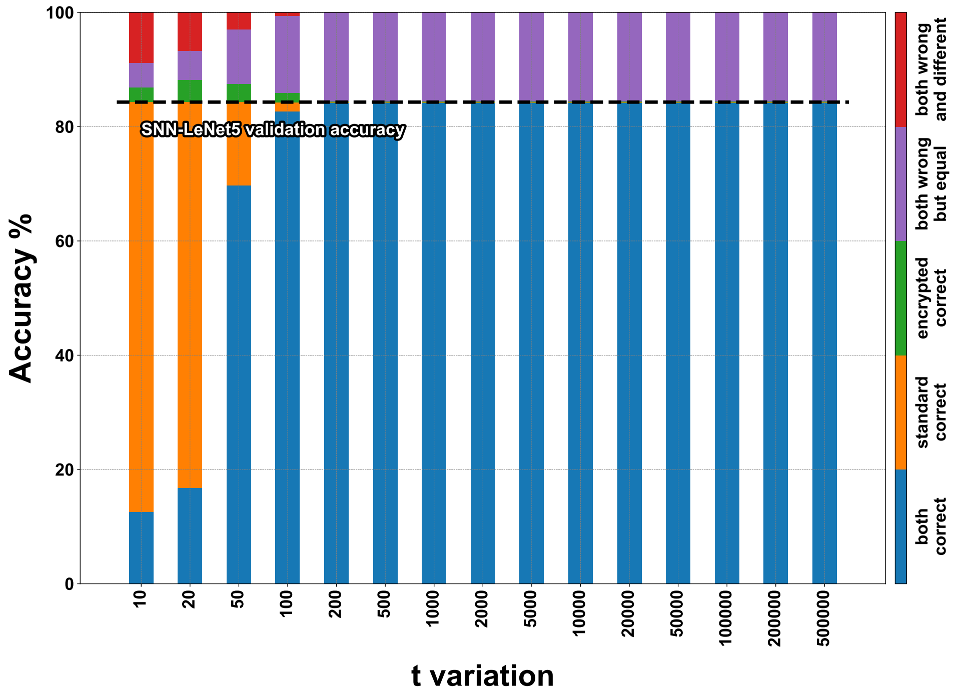


Fig. 3.18 FashionMNIST accuracy on encrypted Spiking-LeNet5 for t variation with m set to 2048.

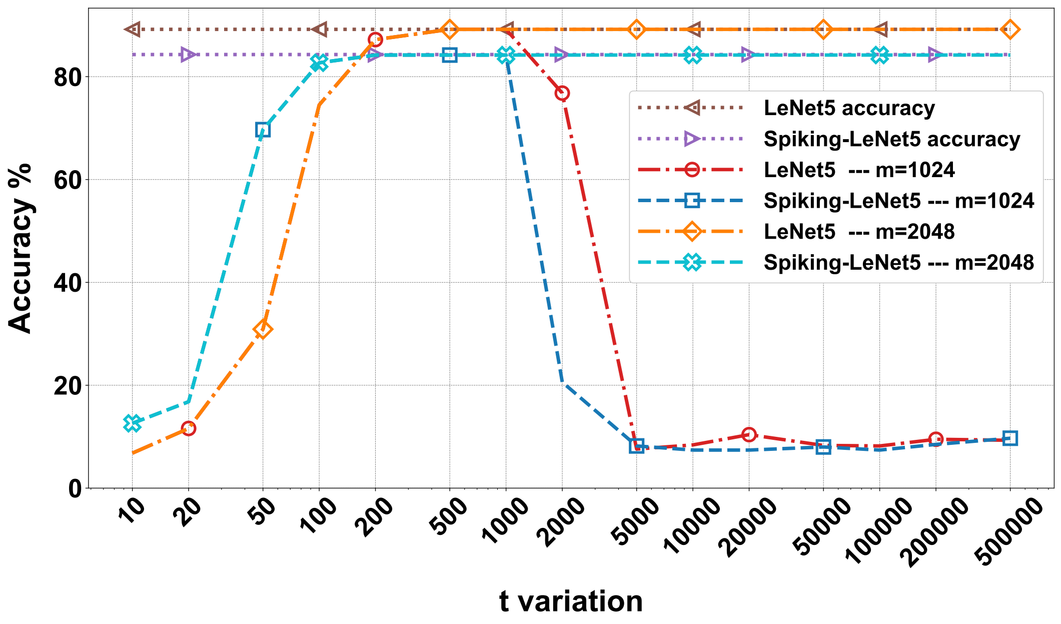


Fig. 3.19 Comparison of FashionMNIST accuracy between plaintext and encrypted versions of LeNet5 and Spiking-LeNet5 for t variations when both plaintext and encrypted versions classified correctly.

In Figure 3.20, we no longer compare only the **Blue - both correct** percentages, but we add these to those of **Green - encrypted correct**. In practice, we added all the cases where the encrypted model correctly provided the same result as the plaintext model, whether the latter was correct or not. In fact, the goal of this research was not only to demonstrate the feasibility of encrypted models but also their reliability, and considering that with certain combinations of t and m, values close to 100% correctness between the encrypted and plaintext models can be achieved, I would say that the result has been achieved. Specifically, we can see that in Figure 3.20, both the standard and spiking models in the encrypted versions reach 100% accuracy in emulating the plaintext versions, maintaining approximately the same shape as Figure 3.19. This means that even in this case, for low t values, the Spiking-LeNet5 model performs better than the LeNet5.

In conclusion, SNNs react better to encryption than DNNs, making them more secure for data encryption. However, they still have some criticalities: first of all, they have an intrinsic latency time, the seq_{length} parameter, that significantly lengthens computation times, and secondly, they generally have lower validation accuracy. The

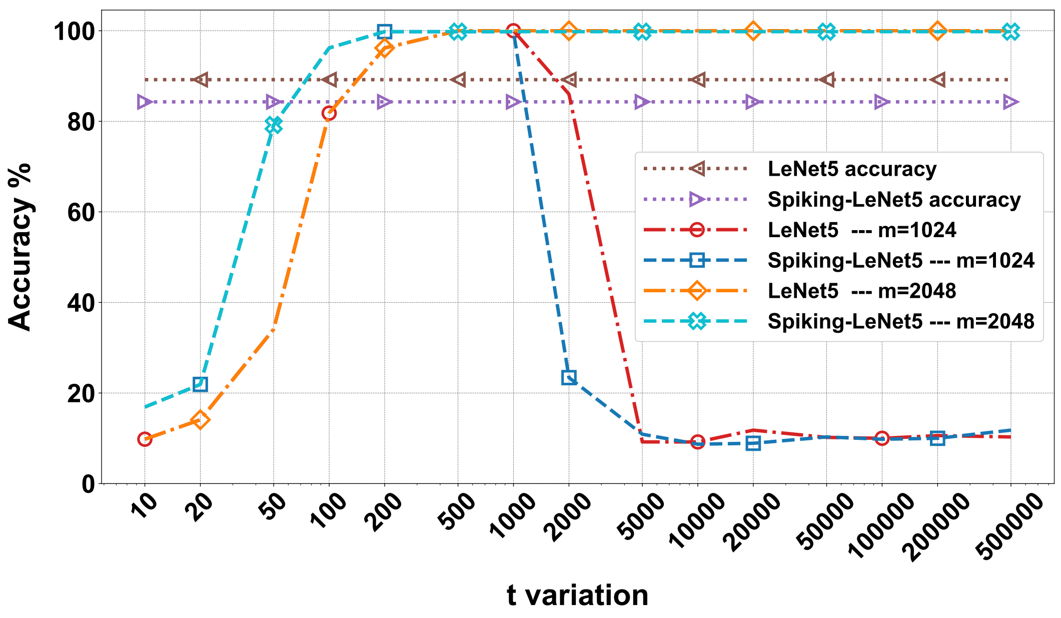


Fig. 3.20 Comparison of FashionMNIST accuracy between plaintext and encrypted versions of LeNet5 and Spiking-LeNet5 for *t* variations when both plaintext and encrypted versions coincide in both correct and incorrect classification.

same results can also be viewed for the MNIST and CIFAR10 datasets in Figures A.8 to A.11.

3.8.1 Models Encryption

One of the main problems of FHE is that it can only work with linear calculations of addition and multiplication. The LeNet5 model, as we have implemented it in Figure 3.8, also includes non-linear calculations: ReLu activations. Currently, this part of calculations cannot be achieved with the encrypted model, so every data pass through the activation layer must be decrypted first and then re-encrypted. Obviously, these steps lead to a model that is not fully encrypted and to data vulnerability during the activation phase. In fact, this research aspect falls within future projects to improve the model.

In Figure 3.21, we can see how the encrypted model actually behaves. The steps are the same for both the standard and the spiking model and apply to all datasets. The color codes used are the same as those used in Figure 3.8 for better understanding. As can be seen, the data must be decrypted and re-encrypted 4 times

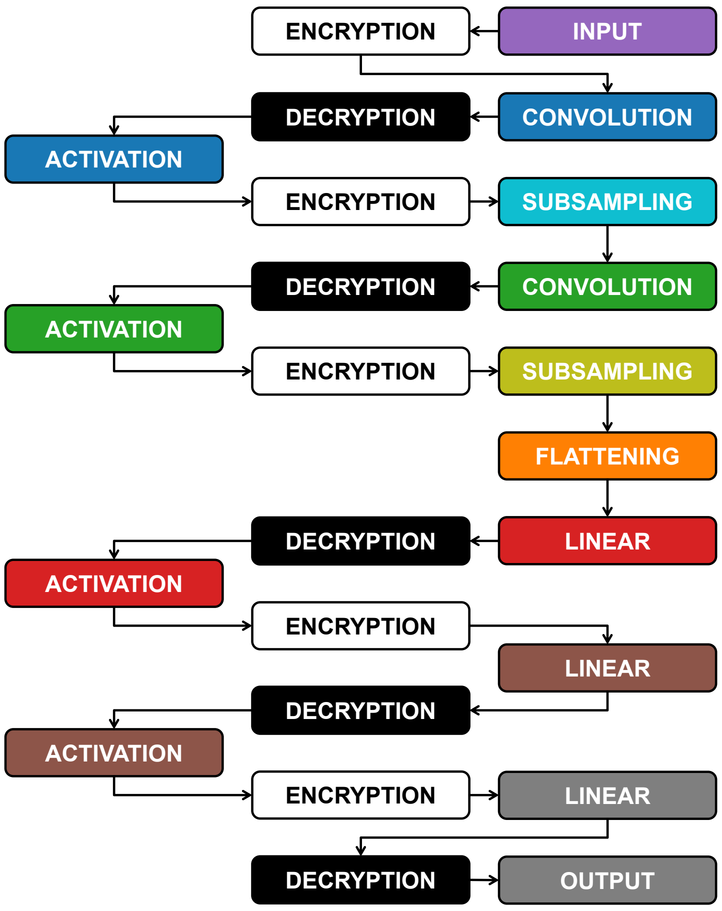


Fig. 3.21 Inside the LeNet5 we need to decrypt and encrypt again four times because the activation function ReLu is not a linear calculation.

during the entire process, in addition to the initial encryption and final decryption. Considering that the computation times of an encrypted model are up to 1000 times slower than plaintext, both in the standard and spiking cases, one can imagine how impactful these encryption steps due to activations are.

Noise Budget Values

However, these continuous encryptions also have a positive aspect. As mentioned in the NB section Section 3.2.2, the NB degrades every time linear calculations are performed, and if it reaches zero, the data would become unreadable. The continuous encryption during the process allows the NB to be recharged each time, enabling encrypted calculations without repercussions on the final accuracy.

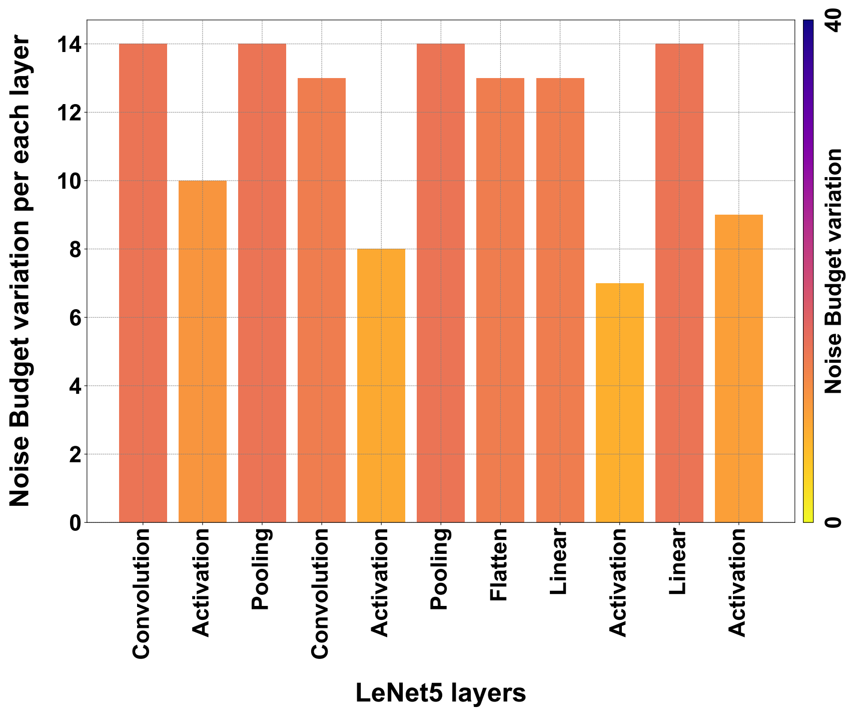


Fig. 3.22 NB qualitative variation during the process across the layers.

In Figure 3.22, we can qualitatively see the amount of NB during the various layers, and it can be observed how it reloads after each activation due to the new encryption. In Figure 3.23, we can see qualitatively how the NB value is not independent of t, but rather, for high values of t, i.e., high encryption, the initial NB value is lower, and therefore fewer calculations can be absorbed, while with low values of t, the NB is higher with greater manipulation possibilities. Figures 3.22 and 3.23 were extrapolated from the overall graph shown in Figure A.12, where all numerical data are reported, and it can be noted that NB does not depend solely on t but also on m. In fact, higher values of m lead to higher NB values, allowing more calculations, but at the same time, drastically increasing computation time.

3.8.2 Confusion Matrices

In Figure 3.24, the confusion matrices of both the standard and spiking models for the FashionMNIST dataset are depicted. It can be observed that in both cases, the matrix is fairly orderly between predicted classes and correct labels. The only class that creates slight confusion for the models is class number 6, representing the *Shirt*.

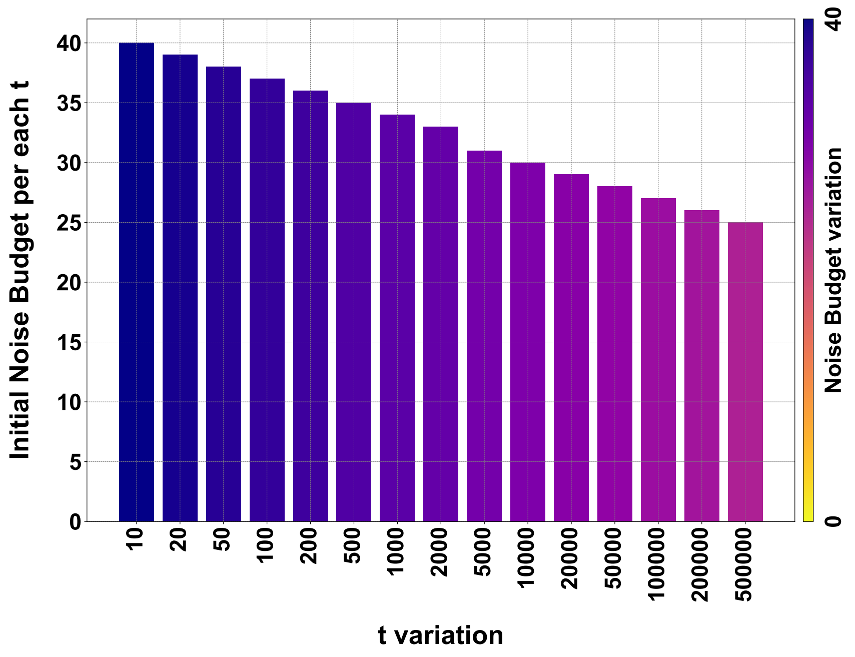
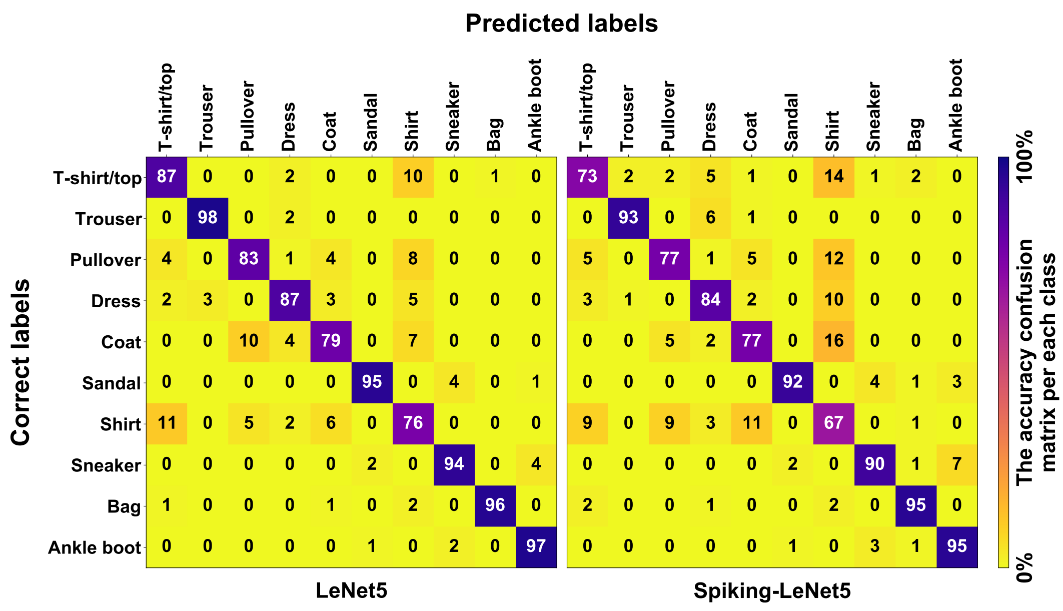


Fig. 3.23 NB qualitative variation for each t variation.

Now, looking at Figure 3.25, we can see all the confusion matrices of the LeNet5 and Spiking-LeNet5 models in the encrypted case, with all variations of t and m. It is easy to notice that for m values equal to 1024, the results degrade quickly with values of t that are too high, mostly resulting in random results equivalent to overly encrypted and no longer readable data. On the other hand, with m values equal to 2048 or higher, the results remain constant and unchanged, but obviously the excessive complexity of encryption makes calculations slower and more difficult. Instead, for low t values, the results are confusing but less random and tend to accumulate on certain classes, especially on the *Shirt* class, as in the plaintext case. Moreover, it can be noticed that they perform better in the spiking version since the matrices stabilize for lower values of t. Obviously, the classes on which the results accumulate depend on the shape and object represented by the class itself. In the case of FashionMNIST, it can be easily inferred that the *Shirt* class is the most confusing for the model, as it can be easily assimilated to other classes.

In Figures A.13 to A.16, the plaintext and encrypted confusion matrices for the MNIST and CIFAR10 datasets are displayed. In this case, it is evident how for MNIST, the most confusing class is class 8, which is visually more complex than all the other numbers and therefore more easily misleads the models, being



LeNet5

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	87	0	0	2	0	0	10	0	1	0
Trouser	0	98	0	2	0	0	0	0	0	0
Pullover	4	0	83	1	4	0	8	0	0	0
Dress	2	3	0	87	3	0	5	0	0	0
Coat	0	0	10	4	79	0	7	0	0	0
Sandal	0	0	0	0	0	95	0	4	0	1
Shirt	11	0	5	2	6	0	76	0	0	0
Sneaker	0	0	0	0	0	2	0	94	0	4
Bag	1	0	0	0	1	0	2	0	96	0
Ankle boot	0	0	0	0	0	1	0	2	0	97

Spiking-LeNet5

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	73	2	2	5	1	0	14	1	2	0
Trouser	0	93	0	6	1	0	0	0	0	0
Pullover	5	0	77	1	5	0	12	0	0	0
Dress	3	1	0	84	2	0	10	0	0	0
Coat	0	0	5	2	77	0	16	0	0	0
Sandal	0	0	0	0	0	92	0	4	1	3
Shirt	9	0	9	3	11	0	67	0	1	0
Sneaker	0	0	0	0	0	2	0	90	1	7
Bag	2	0	0	1	0	0	2	0	95	0
Ankle boot	0	0	0	0	0	1	0	3	1	95

Fig. 3.24 Plaintext confusion matrix for FashionMNIST. The *Shirt* class is the one that misleads the model the most.

able to resemble any other number. It should also be noted that all these confusion matrices reflect the graphs shown in Figures A.3 to A.11 and 3.14 to 3.20, showing the correspondence of various accuracies and the different behavior for different values of m and t.

3.8.3 Layer Errors

In the matrices of Figures A.17, A.18, 3.26 and 3.27, the normalized layer-by-layer errors are represented. Normalization was performed with values ranging from 0 to 1 for each individual matrix. Naturally, for low and high values of t, errors are much higher, even in the order of tens of times, compared to central t values, but normalizing each matrix separately served to show the differences between individual layers and especially between LeNet5 and Spiking-LeNet5. This way, it is better appreciated which model performs better and which layers accumulate more errors. Total normalization across all t values would not have allowed to notice the differences, given the huge difference between the central t values and the most extreme ones.

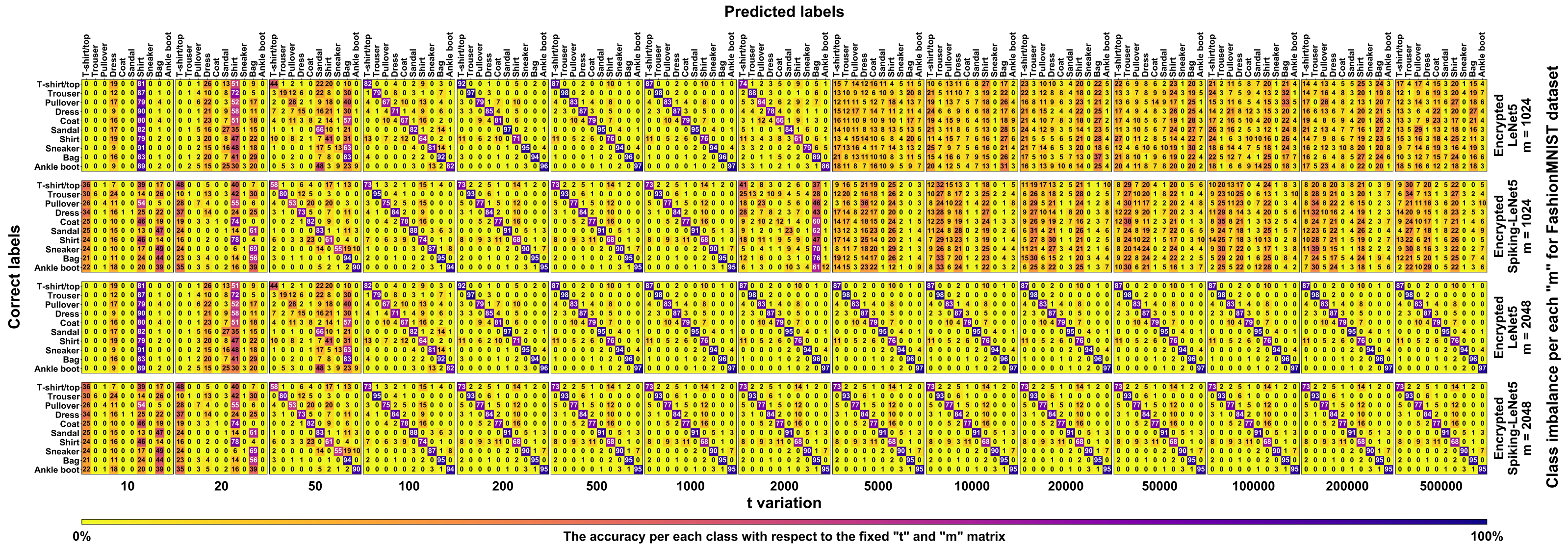


Fig. 3.25 Encrypted confusion matrix for FashionMNIST with t and m variation. It can be noticed that for low values of t, the results tend to concentrate on labels that resemble each other the most. Spiking-LeNet5 is less random than LeNet5 for low values of t.

Observing the matrices, we can notice that errors mainly accumulate in the final layers, especially in the linearization layers, precisely because there are more calculations for reducing large matrices to the final linear array of 10 classes. It can also be noted that there are not many errors in the activation layers because during activations, the models are decrypted and there is a lower accumulation of errors. Furthermore, it is noted that there are no significant differences between the various classes and more or less all have the same error values in the various layers, with a greater accumulation in the final linearizations.

To better understand Figures A.17, A.18, 3.26 and 3.27, here is an explanation of the strips:

- in the first strip (the **Red** one) there are the errors produced by the encrypted LeNet5.
- in the second strip (the **Blue** one) there are the errors produced by the encrypted Spiking-LeNet5.
- in the third strip, the difference between the errors of the standard model and the spiking model was calculated, the normalization in this case was performed after the calculation of the difference. The **Red** parts show that there was a greater error in LeNet5, vice versa the **Blue** parts show that Spiking-LeNet5 made more mistakes.

Looking at the third strip of Figures A.17, A.18, 3.26 and 3.27, it can be noticed that it is mainly **Red**, which means that generally Spiking-LeNet5 performed better.

The sporadic squares much denser than those of other classes or layers generally show those classes that mislead the models the most under certain conditions. For FashionMNIST (see Figure 3.26), for example, a dense square can be seen in the *Shirt* class with m equal to 1024 and t equal to 1000.

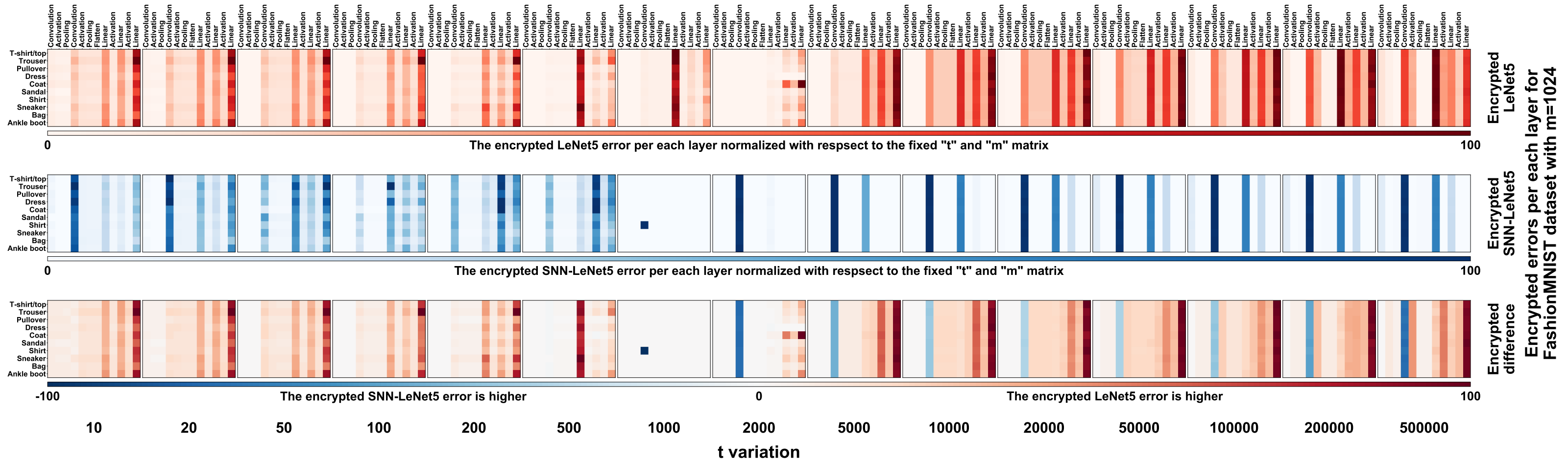


Fig. 3.26 Errors layer-by-layer with FashionMNIST and $m = 1024$. The top **Red** strip represents the errors in the layers of the LeNet5, the **Blue** strip in the middle represents the errors in the layers of the Spiking-LeNet5. The last strip at the bottom represents the difference between the errors in the layers of LeNet-5 and Spiking-LeNet5, where the **Red** parts indicate that LeNet5 has made more mistakes, while the **Blue** parts indicate that Spiking-LeNet5 has mainly made mistakes. It can be noticed that the third strip is predominantly **Red**, indicating that Spiking-LeNet5 generally performs better.

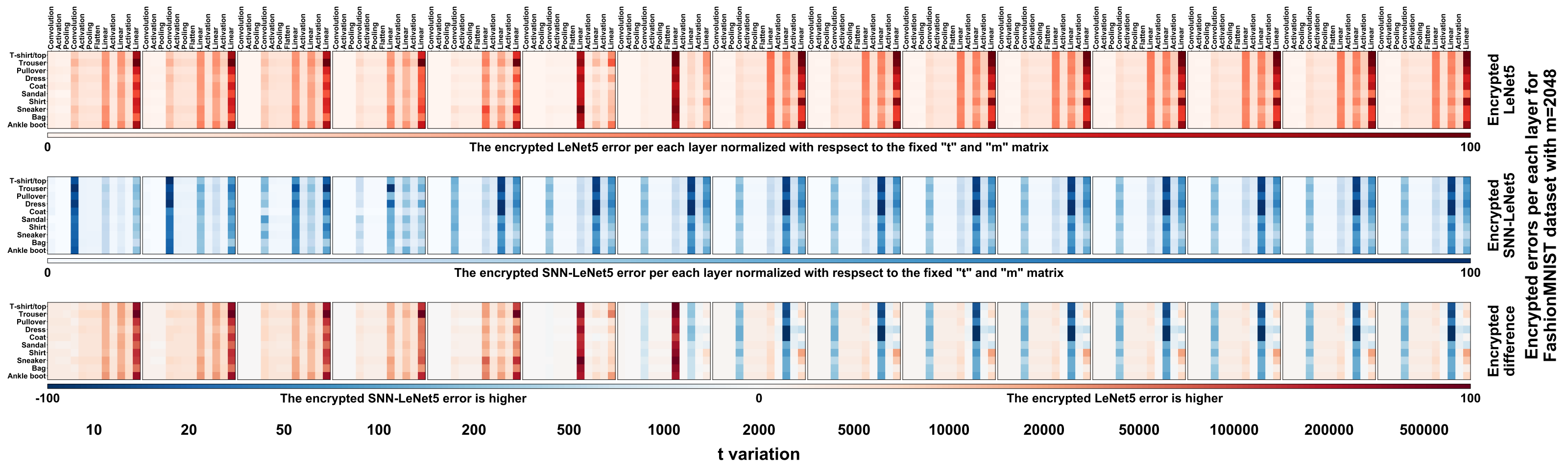


Fig. 3.27 Errors layer-by-layer with FashionMNIST and $m = 2048$. The top **Red** strip represents the errors in the layers of the LeNet5, the **Blue** strip in the middle represents the errors in the layers of the Spiking-LeNet5. The last strip at the bottom represents the difference between the errors in the layers of LeNet-5 and Spiking-LeNet5, where the **Red** parts indicate that LeNet5 has made more mistakes, while the **Blue** parts indicate that Spiking-LeNet5 has mainly made mistakes. It can be noticed that the third strip is predominantly **Red**, indicating that Spiking-LeNet5 generally performs better.

3.9 Conclusion

In this work, we aimed to provide a comparison between classical models like DNNs and the less commonly used and less known SNNs, additionally leveraging FHE to assess their effectiveness and practicality in realistic scenarios. The final outcome demonstrated how, under certain conditions, SNNs are indeed more efficient than DNNs, and how FHE can enable the manipulation of sensitive data without the risk of intrusions. However, this work needs to be further developed to address some of its most glaring limitations:

- the inability to use encrypted data in the nonlinear phases of a model [121].
- the slowness attributed to encryption, particularly pronounced in this specific case since the Pyfhel library operates solely on CPU.
- the latency of SNNs, which precludes the application of these studies to real-time cases.

Therefore, the next steps in this field involve developing accelerated encryption models using GPUs and conducting in-depth research to overcome the issue of non-linear computations. Furthermore, SNNs are still in their infancy, and undoubtedly, there will be more opportunities for their utilization in the future, especially when latency becomes less of a factor due to advancements in computing power.

References

[1] John Paul Mueller and Luca Massaron. *Machine Learning for dummies*. For Dummies, 2016.

[2] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning*. Packt Publishing, 2017.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Nicolas Papernot, Patrick D. McDaniel, Arunesh Sinha, and Michael P. Wellman. Towards the science of security and privacy in machine learning. *CoRR*, abs/1611.03814, 2016.

[5] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.

[6] Shail Dave, Alberto Marchisio, Muhammad Abdullah Hanif, Amira Guesmi, Aviral Shrivastava, Ihsen Alouani, and Muhammad Shafique. Special session: Towards an agile design methodology for efficient, reliable, and secure ML systems. In *40th IEEE VLSI Test Symposium, VTS 2022, San Diego, CA, USA, April 25-27, 2022*, pages 1–14. IEEE, 2022.

[7] Joana Cabral Costa, Tiago Roxo, Hugo Proença, and Pedro R. M. Inácio. How deep learning sees the world: A survey on adversarial attacks & defenses. *CoRR*, abs/2305.10862, 2023.

[8] Kabid Hassan Shibly, Md Delwar Hossain, Hiroyuki Inoue, Yuzo Taenaka, and Youki Kadobayashi. Towards autonomous driving model resistant to adversarial attack. *Appl. Artif. Intell.*, 37(1), 2023.

[9] Samuel G. Finlayson, Isaac S. Kohane, and Andrew L. Beam. Adversarial attacks against medical deep learning systems. *CoRR*, abs/1804.05296, 2018.

[10] Nicole Gross. What chatgpt tells us about gender: A cautionary tale about performativity and gender biases in ai. *Social Sciences*, 12(8), 2023.

[11] Lorenzo Belenguer. AI bias: exploring discriminatory algorithmic decision-making models and the application of possible machine-centric solutions adapted from the pharmaceutical industry. *AI Ethics*, 2(4):771–787, 2022.

[12] Jaydip Sen, Abhiraj Sen, and Ananda Chatterjee. Adversarial attacks on image classification models: Analysis and defense. *CoRR*, abs/2312.16880, 2023.

[13] Zhouhang Xie, Jonathan Brophy, Adam Noack, Wencong You, Kalyani Asthana, Carter Perkins, Sabrina Reis, Sameer Singh, and Daniel Lowd. Identifying adversarial attacks on text classifiers. *CoRR*, abs/2201.08555, 2022.

[14] Lukas Struppek, Dominik Hintersdorf, Antonio De Almeida Correia, Antonia Adler, and Kristian Kersting. Plug & play attacks: Towards robust and flexible model inversion attacks. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 20522–20545. PMLR, 2022.

[15] Yi Xie, Jie Zhang, Shiqian Zhao, Tianwei Zhang, and Xiaofeng Chen. SAME: sample reconstruction against model extraction attacks. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 19974–19982. AAAI Press, 2024.

[16] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 3–18. IEEE Computer Society, 2017.

[17] Homanga Bharadhwaj, Dylan Turpin, Animesh Garg, and Ashton Anderson. De-anonymization of authors through arxiv submissions during double-blind review. *CoRR*, abs/2007.00177, 2020.

[18] James Curzon, Tracy Ann Kosa, Rajen Akalu, and Khalil El-Khatib. Privacy and artificial intelligence. *IEEE Trans. Artif. Intell.*, 2(2):96–108, 2021.

[19] Stefan Fenz, Johannes Heurix, and Thomas Neubauer. Recognition and pseudonymization of personal data in paper-based health records. In Witold Abramowicz, Dalia Kriksciuniene, and Virgilijus Sakalauskas, editors, *Business Information Systems - 15th International Conference, BIS 2012, Vilnius, Lithuania, May 21-23, 2012. Proceedings*, volume 117 of *Lecture Notes in Business Information Processing*, pages 153–164. Springer, 2012.

[20] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, abs/cs/0610105, 2006.

[21] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In Sviatoslav Voloshynovskiy, Jana Dittmann, and Jessica J. Fridrich, editors, *Proceedings of the 8th workshop on Multimedia & Security, MM&Sec 2006, Geneva, Switzerland, September 26-27, 2006*, pages 146–151. ACM, 2006.

[22] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptol. ePrint Arch.*, page 35, 2017.

[23] Simone Disabato, Alessandro Falcetta, Alessio Mongelluzzo, and Manuel Roveri. A privacy-preserving distributed architecture for deep-learning-as-a-service. In *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*, pages 1–8. IEEE, 2020.

[24] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.*, 51(4):79:1–79:35, 2018.

[25] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796. ACM, 2012.

[26] Zhanglong Ji, Zachary Chase Lipton, and Charles Elkan. Differential privacy and machine learning: a survey and review. *CoRR*, abs/1412.7584, 2014.

[27] Subrato Bharati, M. Rubaiyat Hossain Mondal, Prajoy Podder, and V. B. Surya Prasath. Federated learning: Applications, challenges and future directions. *Int. J. Hybrid Intell. Syst.*, 18(1-2):19–35, 2022.

[28] Yehuda Lindell. Secure multiparty computation (MPC). *IACR Cryptol. ePrint Arch.*, page 300, 2020.

[29] Amos Beimel. Secret-sharing schemes: A survey. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *Coding and Cryptology - Third International Workshop, IWCC 2011, Qingdao, China, May 30-June 3, 2011. Proceedings*, volume 6639 of *Lecture Notes in Computer Science*, pages 11–46. Springer, 2011.

[30] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.

[31] Austin Mohr. A survey of zero-knowledge proofs with applications to cryptography. 01 2007.

[32] Dinusha Vatsalan, Dimitrios Karapiperis, and Vassilios S. Verykios. Privacy-preserving record linkage. *CoRR*, abs/2212.05682, 2022.

[33] Basu Dev Shivahare, Shashikant Suman, Sai Sri Nandan Challapalli, Prakarsh Kaushik, Amar Deep Gupta, and Vimal Bibhu. Survey paper: Comparative study of machine learning techniques and its recent applications. In *2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM)*, volume 2, pages 449–454, 2022.

[34] Nathan Drenkow, Numair Sani, Ilya Shpitser, and Mathias Unberath. Robustness in deep learning for computer vision: Mind the gap? *CoRR*, abs/2112.00639, 2021.

[35] Yiwen Guo, Chao Zhang, Changshui Zhang, and Yurong Chen. Sparse dnns with improved adversarial robustness. *CoRR*, abs/1810.09619, 2018.

[36] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian J. Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *CoRR*, abs/1902.06705, 2019.

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[38] Wenjie Ruan, Xinping Yi, and Xiaowei Huang. Adversarial robustness of deep learning: Theory, algorithms, and applications. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 4866–4869. ACM, 2021.

[39] Farzad Nikfam, Alberto Marchisio, Maurizio Martina, and Muhammad Shafique. Accelat: A framework for accelerating the adversarial training of deep neural networks through accuracy gradient. *IEEE Access*, 10:108997–109007, 2022.

[40] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[41] Siwakorn Srisakaokul, Zexuan Zhong, Yuhao Zhang, Wei Yang, and Tao Xie. MULDEF: multi-model-based defense against adversarial examples for neural networks. *CoRR*, abs/1809.00065, 2018.

[42] Derek Wang, Chaoran Li, Sheng Wen, Yang Xiang, Wanlei Zhou, and Surya Nepal. Defensive collaborative multi-task training - defending against adversarial attack towards deep neural networks. *CoRR*, abs/1803.05123, 2018.

[43] Maksym Andriushchenko and Nicolas Flammarion. Understanding and improving fast adversarial training. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[44] Nikolaos Pitropakis, Emmanouil Panaousis, Thanassis Giannetsos, Eleftherios Anastasiadis, and George Loukas. A taxonomy and survey of attacks against machine learning. *Comput. Sci. Rev.*, 34, 2019.

[45] Jingfeng Zhang, Xilie Xu, Bo Han, Gang Niu, Lizhen Cui, Masashi Sugiyama, and Mohan S. Kankanhalli. Attacks which do not kill training make adversarial learning stronger. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 11278–11287. PMLR, 2020.

[46] Osvaldo Simeone. A very brief introduction to machine learning with applications to communication systems. *IEEE Trans. Cogn. Commun. Netw.*, 4(4):648–664, 2018.

[47] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7472–7482. PMLR, 2019.

[48] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[49] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Networks Learn. Syst.*, 30(9):2805–2824, 2019.

[50] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.*, 23(5):828–841, 2019.

[51] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[52] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 86–94. IEEE Computer Society, 2017.

[53] V Porkodi, M Sivaram, Amin Mohammed, and V Manikandan. Survey on white-box attacks and solutions. *Asian Journal of Computer Science and Technology*, 7:28–32, 11 2018.

[54] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 506–519. ACM, 2017.

[55] Tianyu Pang, Xiao Yang, Yinpeng Dong, Hang Su, and Jun Zhu. Bag of tricks for adversarial training. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[56] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018.

[57] Sylvestre-Alvise Rebuffi, Sven Gowal, Dan Andrei Calian, Florian Stimberg, Olivia Wiles, and Timothy A. Mann. Data augmentation can improve robustness. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 29935–29948, 2021.

[58] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 5019–5031, 2018.

[59] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox v0.8.0: A python toolbox to benchmark the robustness of machine learning models. *CoRR*, abs/1707.04131, 2017.

[60] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh

Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[61] Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Comput. Surv.*, 55(12):259:1–259:37, 2023.

[62] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *CoRR*, abs/1702.05659, 2017.

[63] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Guido Masera, Maurizio Martina, and Muhammad Shafique. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access*, 8:225134–225180, 2020.

[64] Leslie N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017*, pages 464–472. IEEE Computer Society, 2017.

[65] Purnendu Mishra and Kishor Sarawadekar. Polynomial learning rate policy with warm restart for deep neural network. In *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), Kochi, India, October 17-20, 2019*, pages 2087–2092. IEEE, 2019.

[66] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[67] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2015.

[68] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research). 2015.

[69] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[70] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *CoRR*, abs/1708.08296, 2017.

[71] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Inf.*, 11(4):193, 2020.

[72] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John P. Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3353–3364, 2019.

[73] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press, 2016.

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[75] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017.

[76] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[77] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

[78] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[79] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[80] Leslie Rice, Eric Wong, and J. Zico Kolter. Overfitting in adversarially robust deep learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 8093–8104. PMLR, 2020.

[81] William Eduardo Villegas-Ch., Angel Jaramillo-Alcázar, and Sergio Luján-Mora. Evaluating the robustness of deep learning models against adversarial attacks: An analysis with fgsm, PGD and CW. *Big Data Cogn. Comput.*, 8(1):8, 2024.

[82] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2574–2582. IEEE Computer Society, 2016.

[83] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *CoRR*, abs/1712.02029, 2017.

[84] Farzad Nikfam, Raffaele Casaburi, Alberto Marchisio, Maurizio Martina, and Muhammad Shafique. A homomorphic encryption framework for privacy-preserving spiking neural networks. *Information*, 14(10):537, 2023.

[85] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[86] Julius von Kügelgen. On artificial spiking neural networks: Principles, limitations and potential, 06 2017.

[87] Youngeun Kim, Yeshwanth Venkatesha, and Priyadarshini Panda. Privatesnn: Privacy-preserving spiking neural networks. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 1192–1200. AAAI Press, 2022.

[88] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016.

[89] Filip Ponulak and Andrzej Kasiński. Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis*, 71:409–33, 01 2011.

[90] Eugene M. Izhikevich. Simple model of spiking neurons. *IEEE Trans. Neural Networks*, 14(6):1569–1572, 2003.

[91] Kaushik Roy, Akhilesh R. Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575:607 – 617, 2019.

[92] Hélène Paugam-Moisy and Sander M. Bohté. Computing with spiking neuron networks. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors, *Handbook of Natural Computing*, pages 335–376. Springer, 2012.

[93] Amirali Amirsoleimani, Majid Ahmadi, Arash Ahmadi, and Mounir Boukadoum. Brain-inspired pattern classification with memristive neural network using the hodgkin-huxley neuron. In *2016 IEEE International Conference on Electronics, Circuits and Systems, ICECS 2016, Monte Carlo, Monaco, December 11-14, 2016*, pages 81–84. IEEE, 2016.

[94] Christian-Gernot Pehle and Jens Egholm Pedersen. Norse - A deep learning library for spiking neural networks, 2021. Documentation: https://norse.ai/docs/.

[95] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

[96] Bing Han and Kaushik Roy. Deep spiking neural network: Energy efficiency through time based coding. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part X*, volume 12355 of *Lecture Notes in Computer Science*, pages 388–404. Springer, 2020.

[97] Junhaeng Lee, Tobi Delbrück, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *CoRR*, abs/1608.08782, 2016.

[98] Chankyu Lee, Syed Shakib Sarwar, and Kaushik Roy. Enabling spike-based backpropagation in state-of-the-art deep neural network architectures. *CoRR*, abs/1903.06379, 2019.

[99] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural Comput.*, 30(6), 2018.

[100] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019.

[101] Alessandro Falcetta and Manuel Roveri. Privacy-preserving deep learning with homomorphic encryption: An introduction. *IEEE Comput. Intell. Mag.*, 17(3):14–25, 2022.

[102] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 617–635. Springer, 2009.

[103] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[104] Jihyeon Ryu, Keunok Kim, and Dongho Won. A study on partially homomorphic encryption. In Sukhan Lee, Hyunseung Choo, and Roslan Ismail, editors, *17th International Conference on Ubiquitous Information Management and Communication, IMCOM 2023, Seoul, Korea, Republic of, January 3-5, 2023*, pages 1–4. IEEE, 2023.

[105] Guillaume Bonnoron, Caroline Fontaine, Guy Gogniat, Vincent Herbert, Vianney Lapôtre, Vincent Migliore, and Adeline Roux-Langlois. Somewhat/fully homomorphic encryption: Implementation progresses and challenges. In Said El Hajji, Abderrahmane Nitaj, and El Mamoun Souidi, editors, *Codes, Cryptology and Information Security - Second International Conference, C2SI 2017, Rabat, Morocco, April 10-12, 2017, Proceedings - In Honor of Claude Carlet*, volume 10194 of *Lecture Notes in Computer Science*, pages 68–82. Springer, 2017.

[106] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[107] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, USA, 2009.

[108] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.

[109] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[110] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *IACR Cryptol. ePrint Arch.*, page 277, 2011.

[111] Alberto Ibarrondo and Alexander Viand. Pyfhel: Python for homomorphic encryption libraries. In *WAHC '21: Proceedings of the 9th on Workshop on*

Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021, pages 11–16. WAHC@ACM, 2021.

[112] Li Deng. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process. Mag.*, 29(6):141–142, 2012.

[113] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[114] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[115] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[116] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions: Theoretical analysis and applications. In *International Conference on Machine Learning (ICML)*, volume 202, pages 23803–23828. PMLR, 2023.

[117] Donglai Zhu, Hengshuai Yao, Bei Jiang, and Peng Yu. Negative log likelihood ratio loss for deep neural network classification. *CoRR*, abs/1804.10690, 2018.

[118] Boudjelal Meftah, Olivier Lézoray, Soni Chaturvedi, Aleefia A. Khurshid, and Abdelkader Benyettou. Image processing with spiking neuron networks. In Xin-She Yang, editor, *Artificial Intelligence, Evolutionary Computing and Metaheuristics - In the Footsteps of Alan Turing*, volume 427 of *Studies in Computational Intelligence*, pages 525–544. Springer, 2013.

[119] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

[120] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: VGG and residual architectures. *CoRR*, abs/1802.02627, 2018.

[121] Alberto Marchisio, Giorgio Nanfa, Faiq Khalid, Muhammad Abdullah Hanif, Maurizio Martina, and Muhammad Shafique. Is spiking secure? A comparative study on the security vulnerabilities of spiking and deep neural networks. In *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*, pages 1–8. IEEE, 2020.

Appendix A

SpyKing Extras

In this Appendix A, all the results related to the SpyKing research (see Chapter 3) have been provided, which were not included in the main chapter to avoid overburdening the argumentation. Since Chapter 3 mainly contained results for the FashionMNIST dataset, this Appendix A presents the figures and graphs related to the MNIST and CIFAR10 datasets. The following images are simply an ordered list corresponding to Chapter 3, along with their descriptions. For a complete explanation, refer to the text in Chapter 3, which also includes references to the images in this Appendix A.

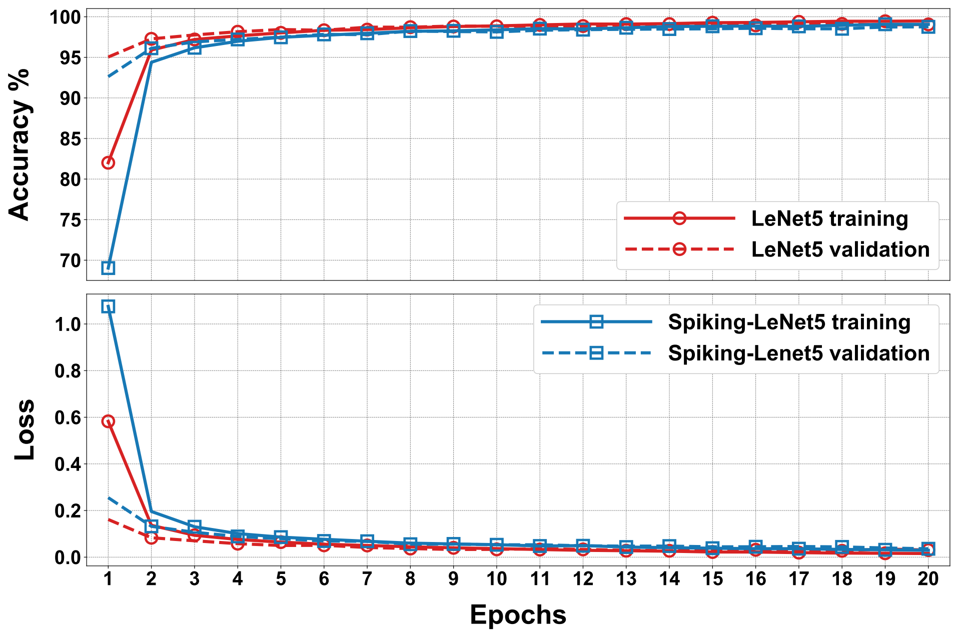


Fig. A.1 Accuracy and loss during training and validation of LeNet5 and Spiking-LeNet5 for the MNIST dataset. The figure shows accuracy and loss values across different training epochs.

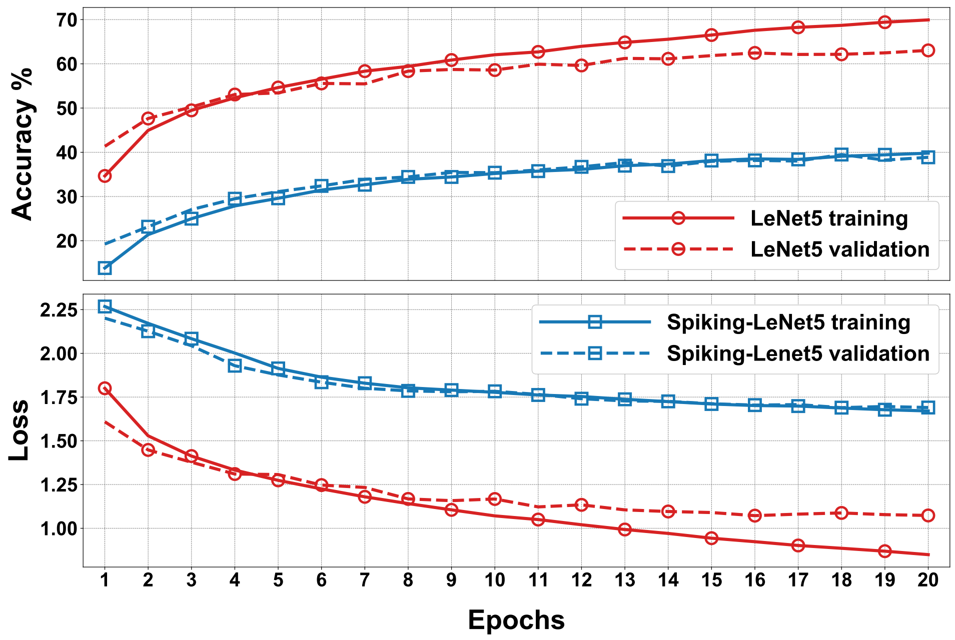


Fig. A.2 Accuracy and loss during training and validation of LeNet5 and Spiking-LeNet5 for the CIFAR10 dataset. The figure shows accuracy and loss values across different training epochs.

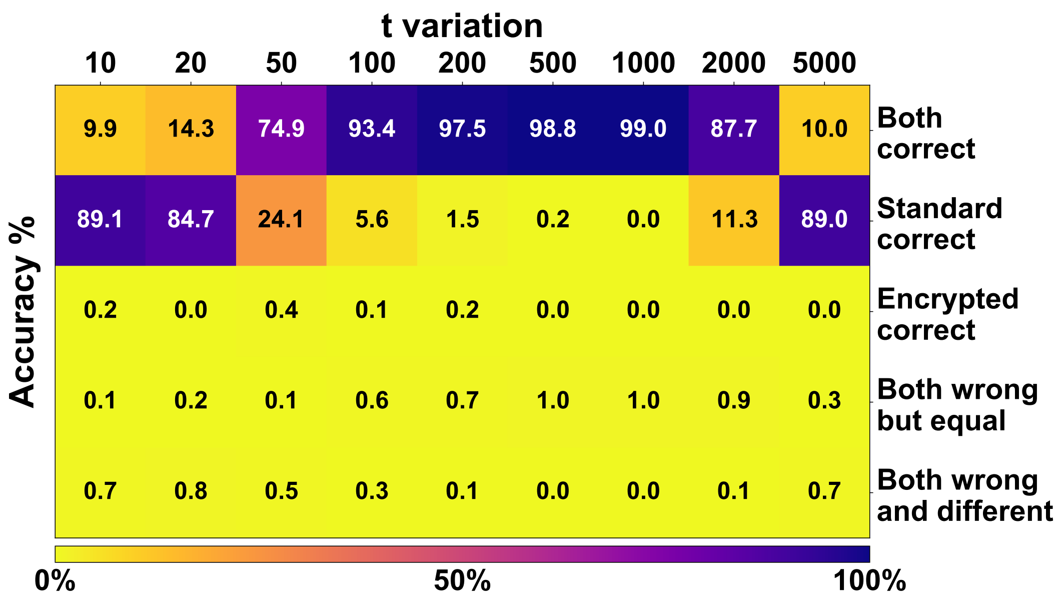


Fig. A.3 Comparison matrix for t variation and m set to 1024 for the MNIST dataset on encrypted LeNet5 model.

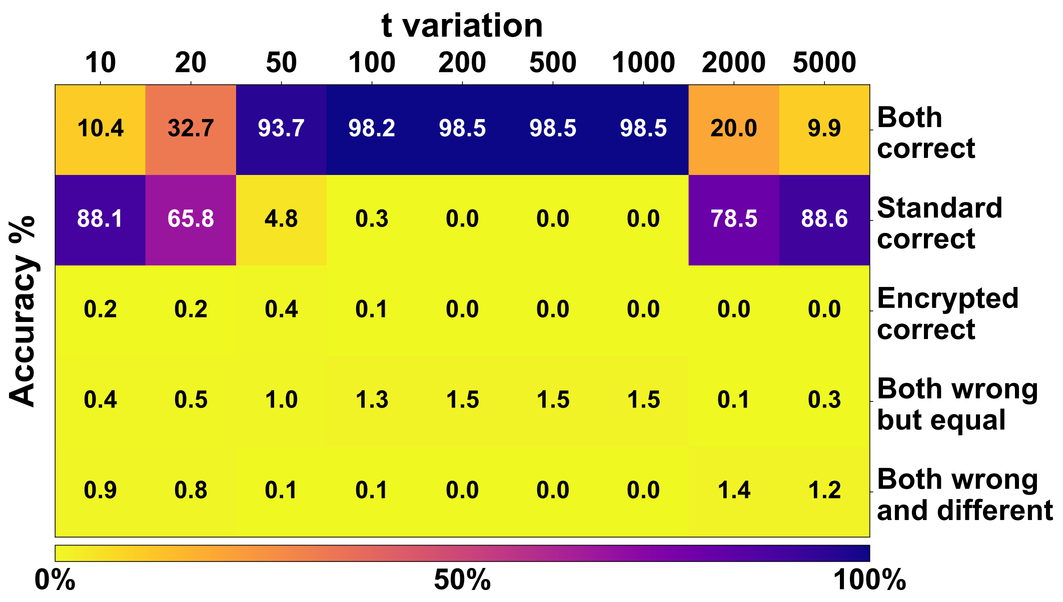


Fig. A.4 Comparison matrix for t variation and m set to 1024 for the MNIST dataset on encrypted Spiking-LeNet5 model.

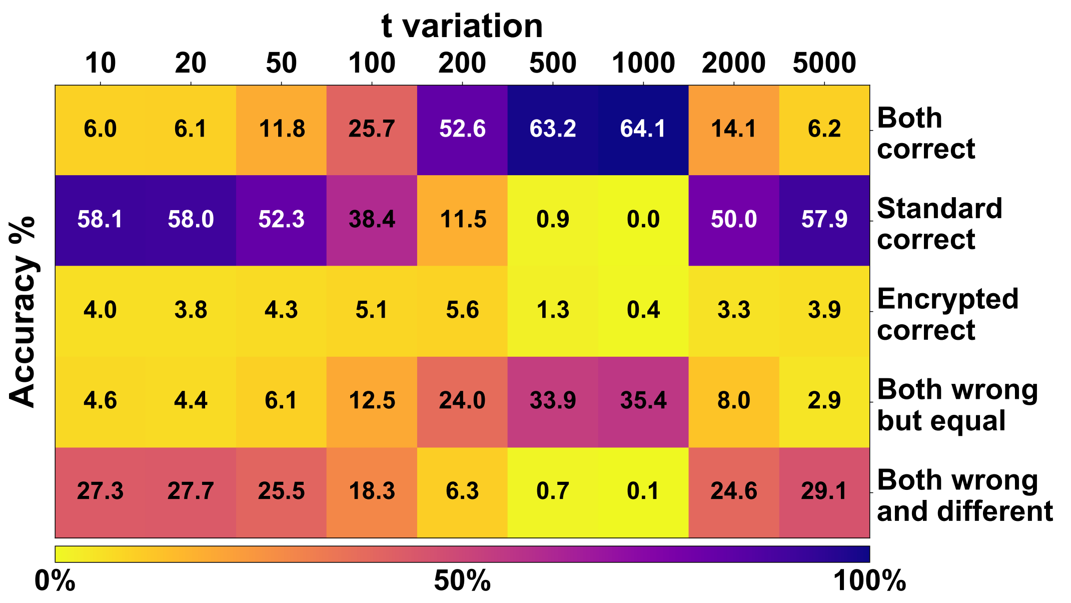


Fig. A.5 Comparison matrix for t variation and m set to 1024 for the CIFAR10 dataset on encrypted LeNet5 model.

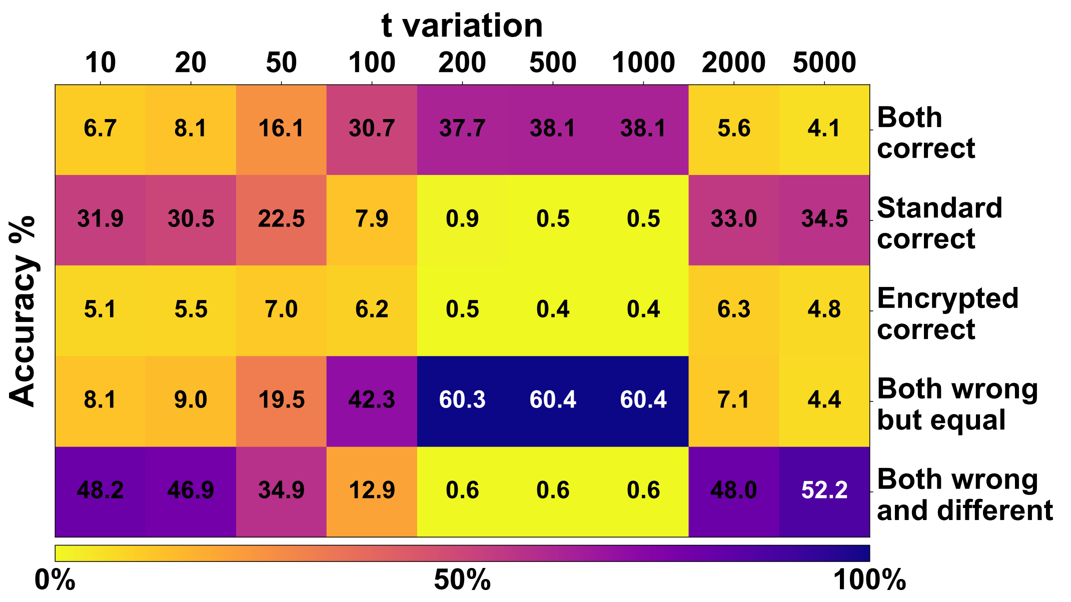


Fig. A.6 Comparison matrix for t variation and m set to 1024 for the CIFAR10 dataset on encrypted Spiking-LeNet5 model.

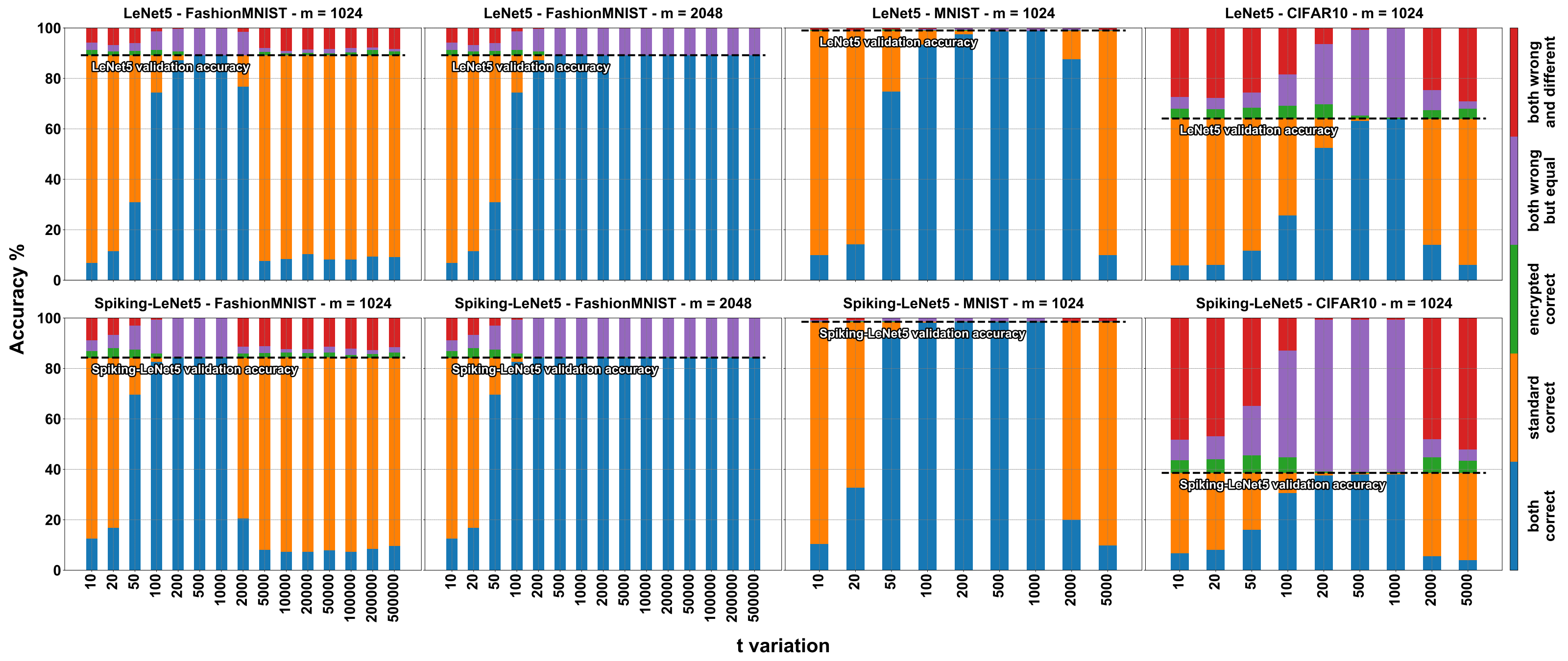


Fig. A.7 Stacked-bar comparison for all the datasets accuracy on encrypted LeNet5 and Spiking-LeNet5 models for t and m variation.

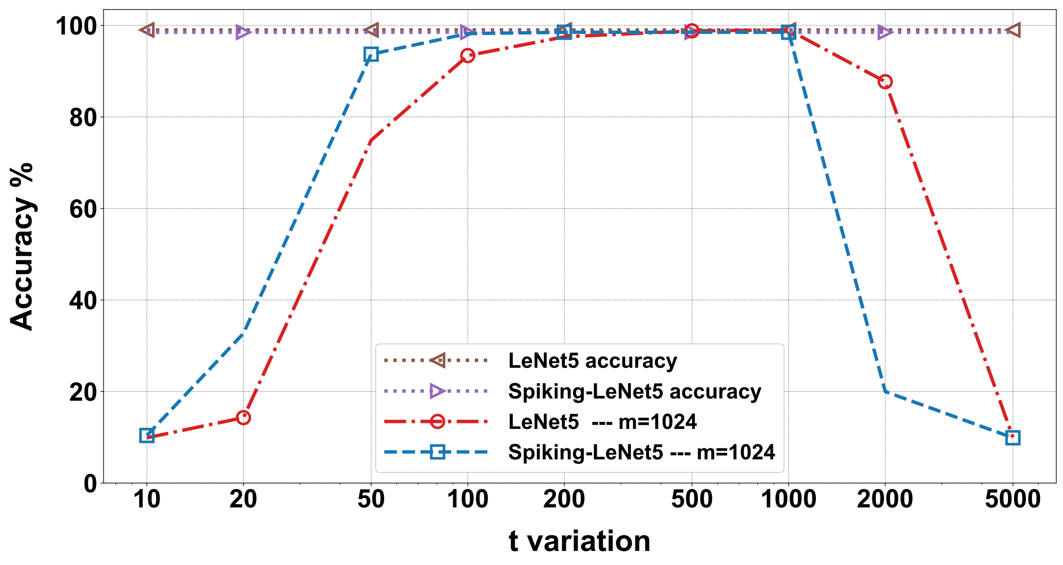


Fig. A.8 Comparison of MNIST accuracy between plaintext and encrypted versions of LeNet5 and Spiking-LeNet5 for t variations when both plaintext and encrypted versions classified correctly.

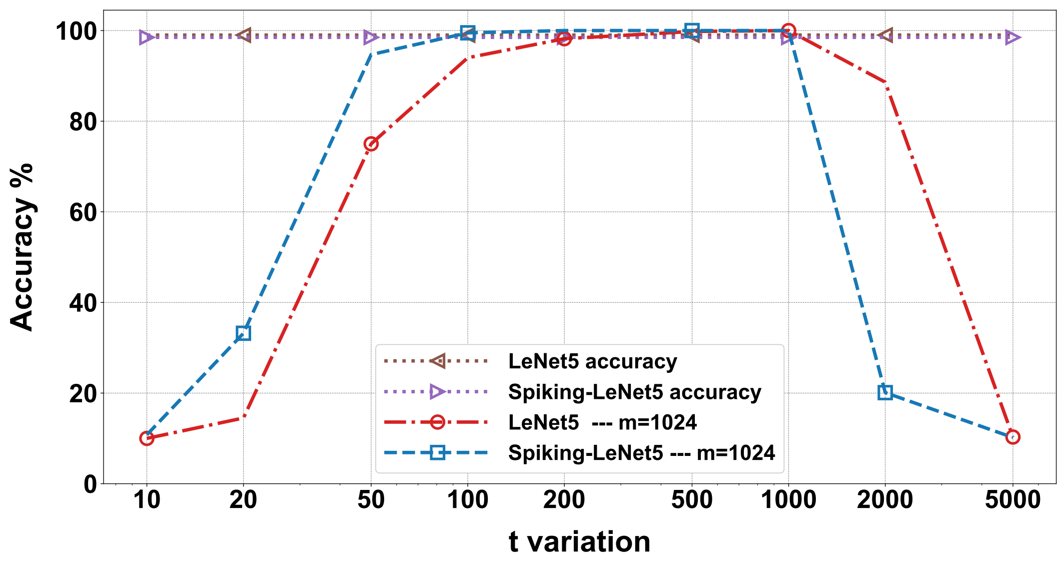


Fig. A.9 Comparison of MNIST accuracy between plaintext and encrypted versions of LeNet5 and Spiking-LeNet5 for t variations when both plaintext and encrypted versions coincide in both correct and incorrect classification.

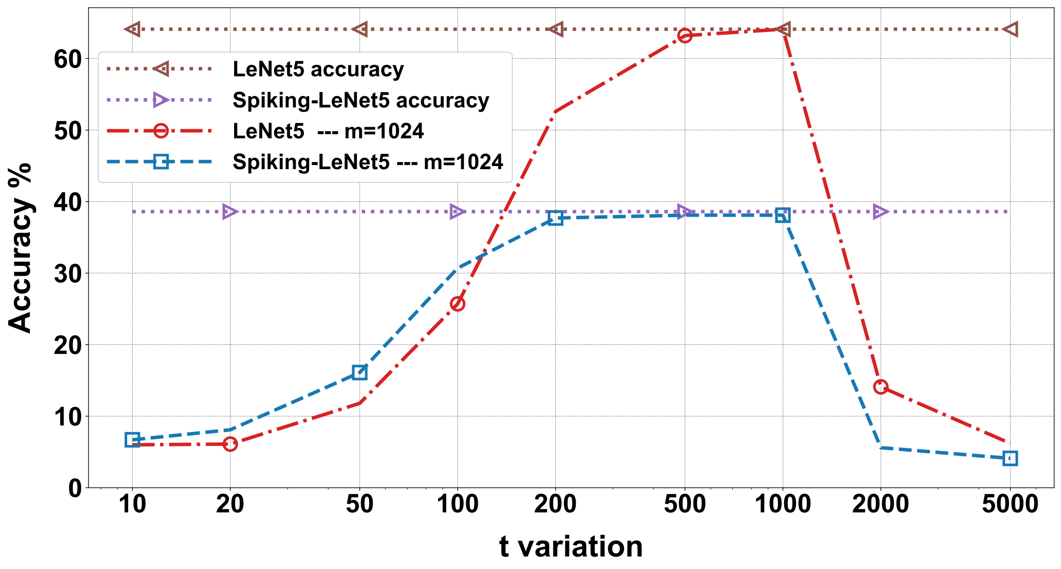


Fig. A.10 Comparison of CIFAR10 accuracy between plaintext and encrypted versions of LeNet5 and Spiking-LeNet5 for t variations when both plaintext and encrypted versions classified correctly.

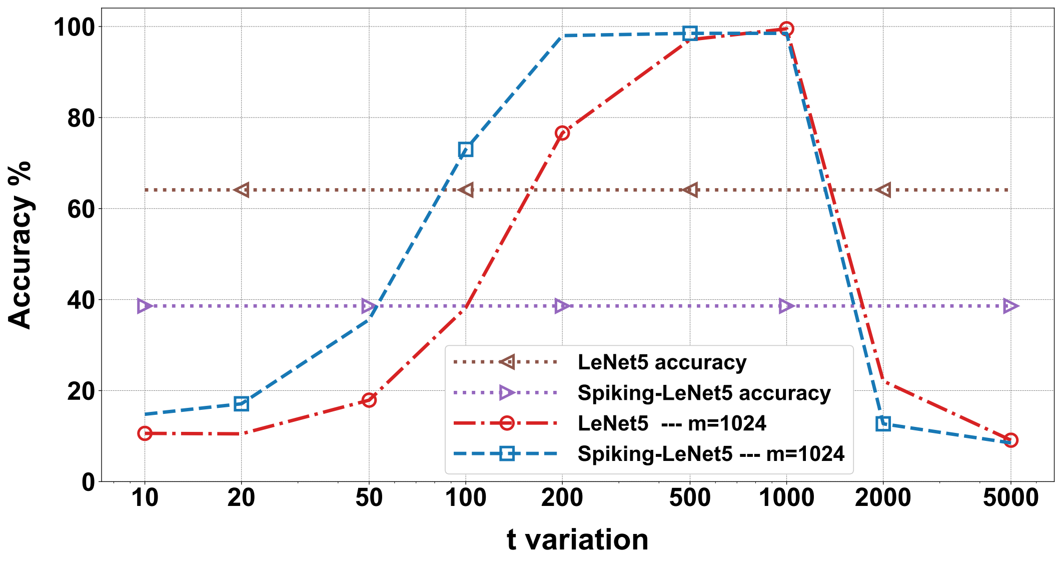
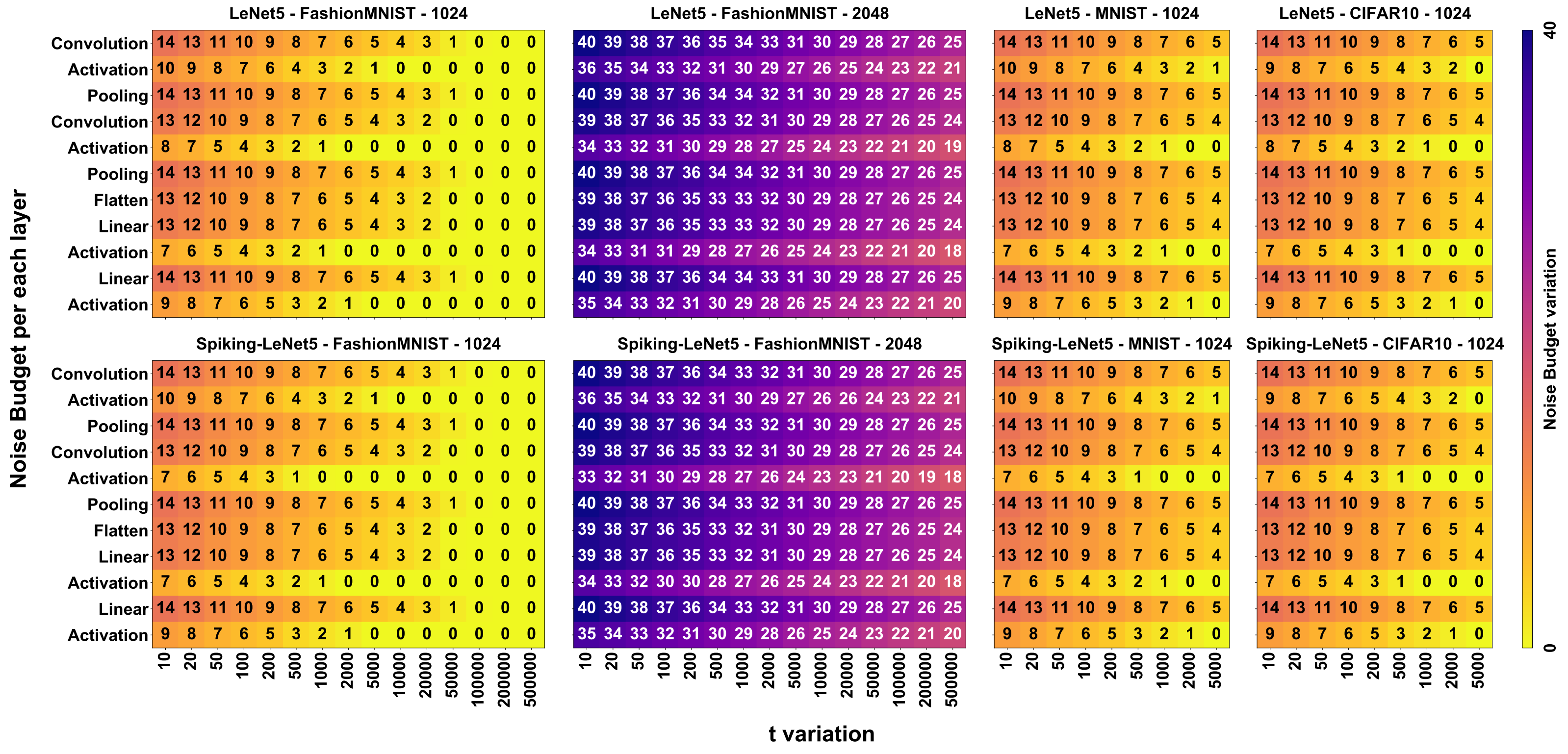


Fig. A.11 Comparison of CIFAR10 accuracy between plaintext and encrypted versions of LeNet5 and Spiking-LeNet5 for t variations when both plaintext and encrypted versions coincide in both correct and incorrect classification.



LeNet5 - FashionMNIST - 1024

	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000	200000	500000
Convolution	14	13	11	10	9	8	7	6	5	4	3	1	0	0	0
Activation	10	9	8	7	6	4	3	2	1	0	0	0	0	0	0
Pooling	14	13	11	10	9	8	7	6	5	4	3	1	0	0	0
Convolution	13	12	10	9	8	7	6	5	4	3	2	0	0	0	0
Activation	8	7	5	4	3	2	1	0	0	0	0	0	0	0	0
Pooling	14	13	11	10	9	8	7	6	5	4	3	1	0	0	0
Flatten	13	12	10	9	8	7	6	5	4	3	2	0	0	0	0
Linear	13	12	10	9	8	7	6	5	4	3	2	0	0	0	0
Activation	7	6	5	4	3	2	1	0	0	0	0	0	0	0	0
Linear	14	13	11	10	9	8	7	6	5	4	3	1	0	0	0
Activation	9	8	7	6	5	3	2	1	0	0	0	0	0	0	0

LeNet5 - FashionMNIST - 2048

	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000	200000	500000
Convolution	40	39	38	37	36	35	34	33	31	30	29	28	27	26	25
Activation	36	35	34	33	32	31	30	29	27	26	25	24	23	22	21
Pooling	40	39	38	37	36	34	34	32	31	30	29	28	27	26	25
Convolution	39	38	37	36	35	33	32	31	30	29	28	27	26	25	24
Activation	34	33	32	31	30	29	28	27	25	24	23	22	21	20	19
Pooling	40	39	38	37	36	34	34	32	31	30	29	28	27	26	25
Flatten	39	38	37	36	35	33	33	32	30	29	28	27	26	25	24
Linear	39	38	37	36	35	33	33	32	30	29	28	27	26	25	24
Activation	34	33	31	31	29	28	27	26	25	24	23	22	21	20	18
Linear	40	39	38	37	36	34	34	33	31	30	29	28	27	26	25
Activation	35	34	33	32	31	30	29	28	26	25	24	23	22	21	20

LeNet5 - MNIST - 1024

	10	20	50	100	200	500	1000	2000	5000
Convolution	14	13	11	10	9	8	7	6	5
Activation	10	9	8	7	6	4	3	2	1
Pooling	14	13	11	10	9	8	7	6	5
Convolution	13	12	10	9	8	7	6	5	4
Activation	8	7	5	4	3	2	1	0	0
Pooling	14	13	11	10	9	8	7	6	5
Flatten	13	12	10	9	8	7	6	5	4
Linear	13	12	10	9	8	7	6	5	4
Activation	7	6	5	4	3	2	1	0	0
Linear	14	13	11	10	9	8	7	6	5
Activation	9	8	7	6	5	3	2	1	0

LeNet5 - CIFAR10 - 1024

	10	20	50	100	200	500	1000	2000	5000
Convolution	14	13	11	10	9	8	7	6	5
Activation	9	8	7	6	5	4	3	2	0
Pooling	14	13	11	10	9	8	7	6	5
Convolution	13	12	10	9	8	7	6	5	4
Activation	8	7	5	4	3	2	1	0	0
Pooling	14	13	11	10	9	8	7	6	5
Flatten	13	12	10	9	8	7	6	5	4
Linear	13	12	10	9	8	7	6	5	4
Activation	7	6	5	4	3	1	0	0	0
Linear	14	13	11	10	9	8	7	6	5
Activation	9	8	7	6	5	3	2	1	0

Spiking-LeNet5 - FashionMNIST - 1024

	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000	200000	500000
Convolution	14	13	11	10	9	8	7	6	5	4	3	1	0	0	0
Activation	10	9	8	7	6	4	3	2	1	0	0	0	0	0	0
Pooling	14	13	11	10	9	8	7	6	5	4	3	1	0	0	0
Convolution	13	12	10	9	8	7	6	5	4	3	2	0	0	0	0
Activation	7	6	5	4	3	1	0	0	0	0	0	0	0	0	0
Pooling	14	13	11	10	9	8	7	6	5	4	3	1	0	0	0
Flatten	13	12	10	9	8	7	6	5	4	3	2	0	0	0	0
Linear	13	12	10	9	8	7	6	5	4	3	2	0	0	0	0
Activation	7	6	5	4	3	2	1	0	0	0	0	0	0	0	0
Linear	14	13	11	10	9	8	7	6	5	4	3	1	0	0	0
Activation	9	8	7	6	5	3	2	1	0	0	0	0	0	0	0

Spiking-LeNet5 - FashionMNIST - 2048

	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000	200000	500000
Convolution	40	39	38	37	36	34	33	32	31	30	29	28	27	26	25
Activation	36	35	34	33	32	31	30	29	27	26	26	24	23	22	21
Pooling	40	39	38	37	36	34	33	32	31	30	29	28	27	26	25
Convolution	39	38	37	36	35	33	32	31	30	29	28	27	26	25	24
Activation	33	32	31	30	29	28	27	26	24	23	23	21	20	19	18
Pooling	40	39	38	37	36	34	33	32	31	30	29	28	27	26	25
Flatten	39	38	37	36	35	33	32	31	30	29	28	27	26	25	24
Linear	39	38	37	36	35	33	32	31	30	29	28	27	26	25	24
Activation	34	33	32	30	30	28	27	26	25	24	23	22	21	20	18
Linear	40	39	38	37	36	34	33	32	31	30	29	28	27	26	25
Activation	35	34	33	32	31	30	29	28	26	25	24	23	22	21	20

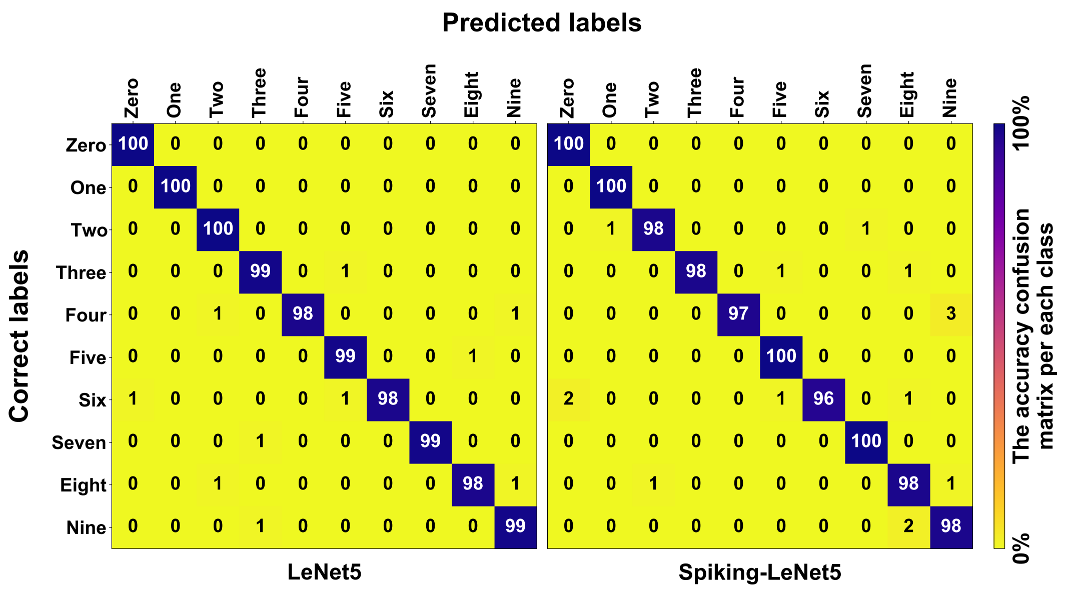
Spiking-LeNet5 - MNIST - 1024

	10	20	50	100	200	500	1000	2000	5000
Convolution	14	13	11	10	9	8	7	6	5
Activation	10	9	8	7	6	4	3	2	1
Pooling	14	13	11	10	9	8	7	6	5
Convolution	13	12	10	9	8	7	6	5	4
Activation	7	6	5	4	3	1	0	0	0
Pooling	14	13	11	10	9	8	7	6	5
Flatten	13	12	10	9	8	7	6	5	4
Linear	13	12	10	9	8	7	6	5	4
Activation	7	6	5	4	3	2	1	0	0
Linear	14	13	11	10	9	8	7	6	5
Activation	9	8	7	6	5	3	2	1	0

Spiking-LeNet5 - CIFAR10 - 1024

	10	20	50	100	200	500	1000	2000	5000
Convolution	14	13	11	10	9	8	7	6	5
Activation	9	8	7	6	5	4	3	2	0
Pooling	14	13	11	10	9	8	7	6	5
Convolution	13	12	10	9	8	7	6	5	4
Activation	7	6	5	4	3	1	0	0	0
Pooling	14	13	11	10	9	8	7	6	5
Flatten	13	12	10	9	8	7	6	5	4
Linear	13	12	10	9	8	7	6	5	4
Activation	7	6	5	4	3	1	0	0	0
Linear	14	13	11	10	9	8	7	6	5
Activation	9	8	7	6	5	3	2	1	0

Fig. A.12 NB values for each model, dataset, *m* and *t* variation.



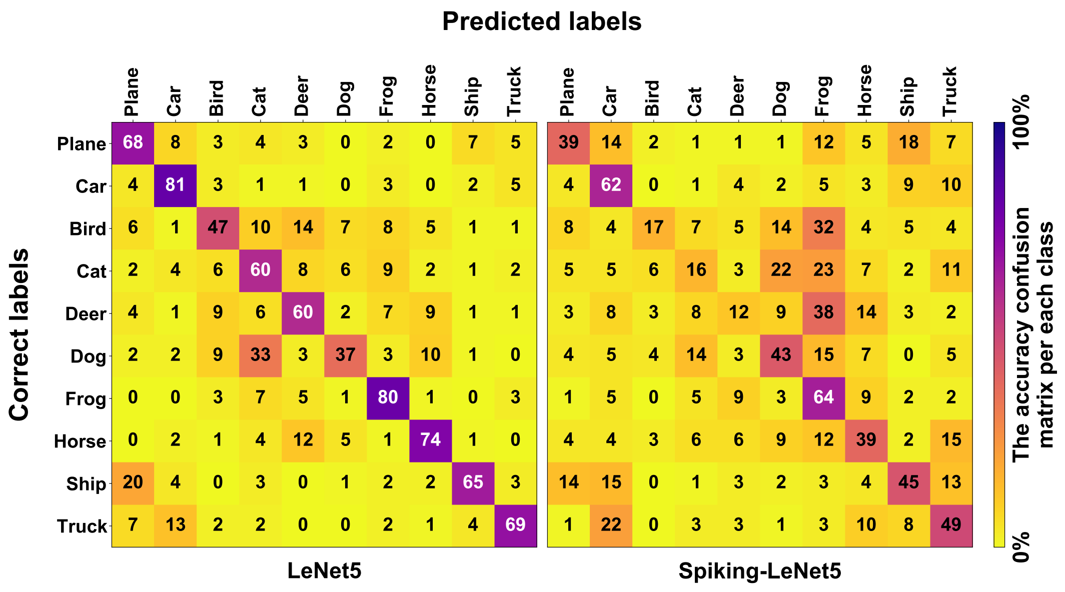
LeNet5:

	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine
Zero	100	0	0	0	0	0	0	0	0	0
One	0	100	0	0	0	0	0	0	0	0
Two	0	0	100	0	0	0	0	0	0	0
Three	0	0	0	99	0	1	0	0	0	0
Four	0	0	1	0	98	0	0	0	0	1
Five	0	0	0	0	0	99	0	0	1	0
Six	1	0	0	0	0	1	98	0	0	0
Seven	0	0	0	1	0	0	0	99	0	0
Eight	0	0	1	0	0	0	0	0	98	1
Nine	0	0	0	1	0	0	0	0	0	99

Spiking-LeNet5:

	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine
Zero	100	0	0	0	0	0	0	0	0	0
One	0	100	0	0	0	0	0	0	0	0
Two	0	1	98	0	0	0	0	1	0	0
Three	0	0	0	98	0	1	0	0	1	0
Four	0	0	0	0	97	0	0	0	0	3
Five	0	0	0	0	0	100	0	0	0	0
Six	2	0	0	0	0	1	96	0	1	0
Seven	0	0	0	0	0	0	0	100	0	0
Eight	0	0	1	0	0	0	0	0	98	1
Nine	0	0	0	0	0	0	0	0	2	98

Fig. A.13 Plaintext confusion matrix for MNIST. The class 8 is the one that misleads the model the most.



LeNet5:

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	68	8	3	4	3	0	2	0	7	5
Car	4	81	3	1	1	0	3	0	2	5
Bird	6	1	47	10	14	7	8	5	1	1
Cat	2	4	6	60	8	6	9	2	1	2
Deer	4	1	9	6	60	2	7	9	1	1
Dog	2	2	9	33	3	37	3	10	1	0
Frog	0	0	3	7	5	1	80	1	0	3
Horse	0	2	1	4	12	5	1	74	1	0
Ship	20	4	0	3	0	1	2	2	65	3
Truck	7	13	2	2	0	0	2	1	4	69

Spiking-LeNet5:

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	39	14	2	1	1	1	12	5	18	7
Car	4	62	0	1	4	2	5	3	9	10
Bird	8	4	17	7	5	14	32	4	5	4
Cat	5	5	6	16	3	22	23	7	2	11
Deer	3	8	3	8	12	9	38	14	3	2
Dog	4	5	4	14	3	43	15	7	0	5
Frog	1	5	0	5	9	3	64	9	2	2
Horse	4	4	3	6	6	9	12	39	2	15
Ship	14	15	0	1	3	2	3	4	45	13
Truck	1	22	0	3	3	1	3	10	8	49

Fig. A.14 Plaintext confusion matrix for CIFAR10.

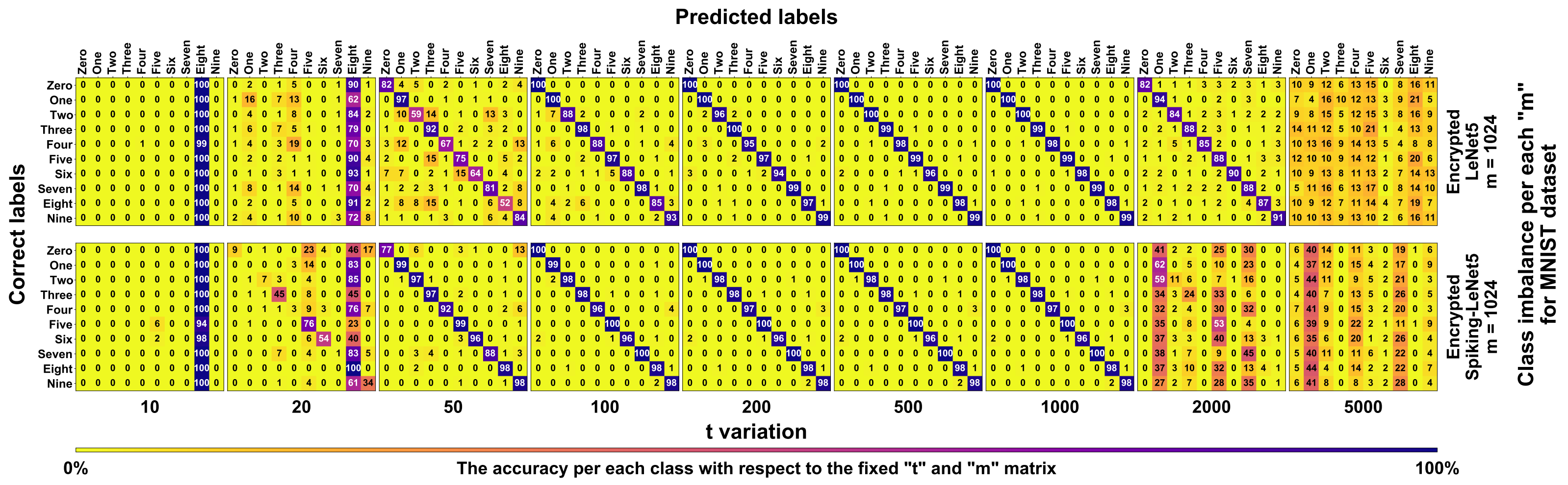


Fig. A.15 Encrypted confusion matrix for MNIST with t and m variation. It can be noticed that for low values of t, the results tend to concentrate on labels that resemble each other the most. Spiking-LeNet5 is less random than LeNet5 for low values of t.

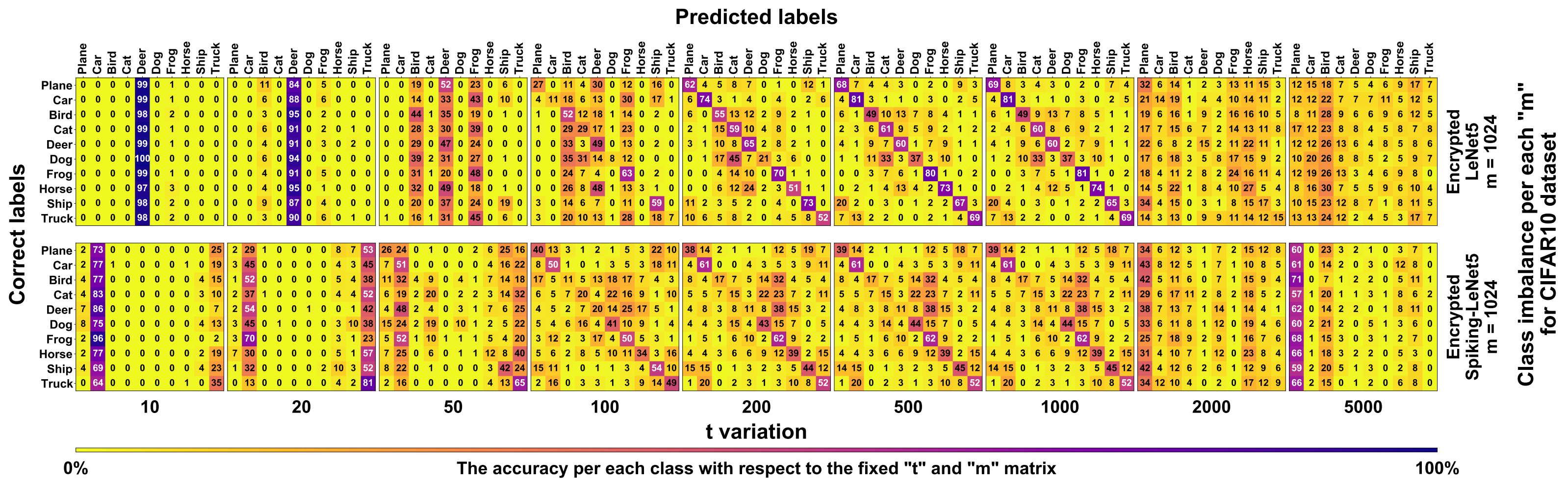


Fig. A.16 Encrypted confusion matrix for CIFAR10 with t and m variation. It can be noticed that for low values of t, the results tend to concentrate on labels that resemble each other the most. Spiking-LeNet5 is less random than LeNet5 for low values of t.

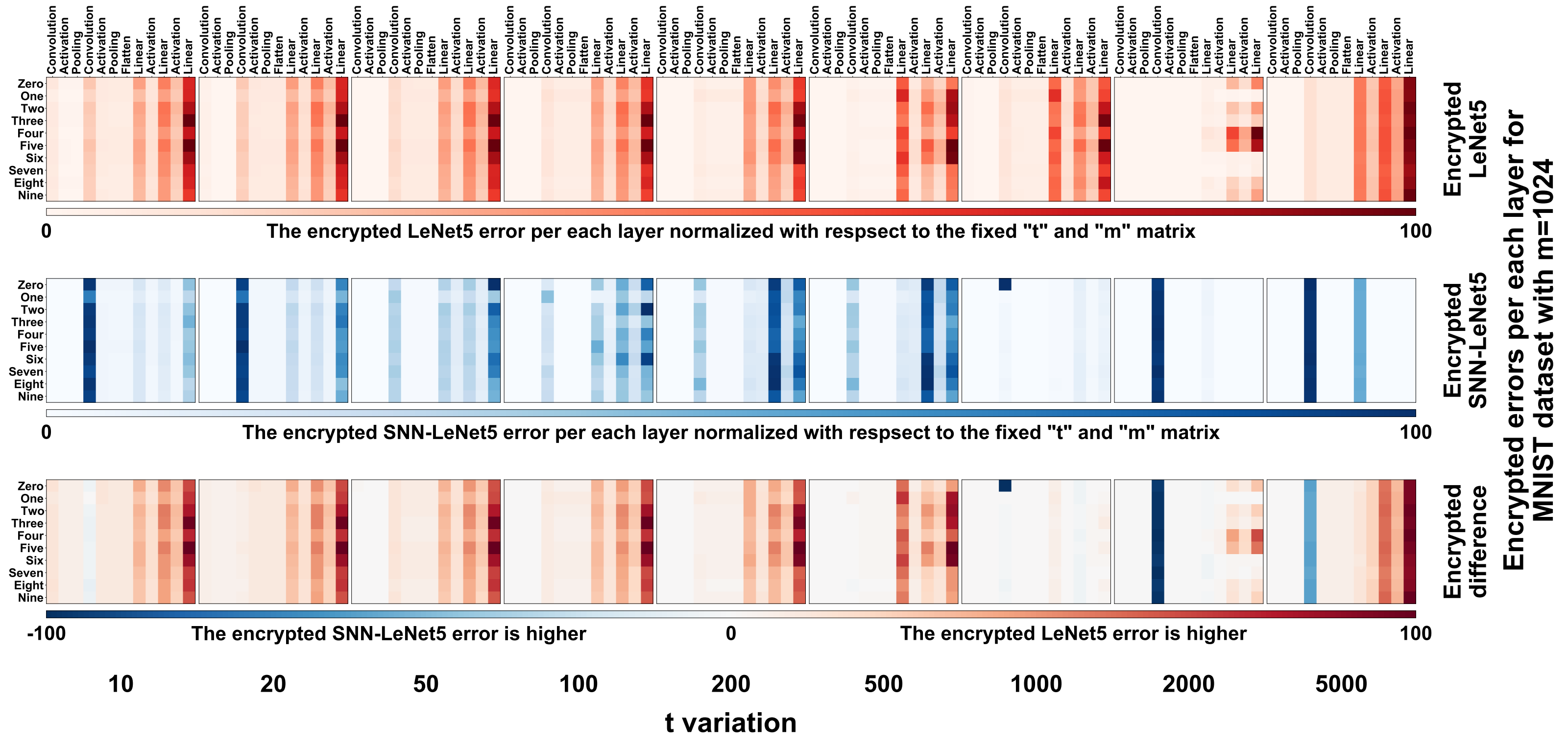


Fig. A.17 Errors layer-by-layer with MNIST and $m = 1024$. The top Red strip represents the errors in the layers of the LeNet5, the Blue strip in the middle represents the errors in the layers of the Spiking-LeNet5. The last strip at the bottom represents the difference between the errors in the layers of LeNet-5 and Spiking-LeNet5, where the Red parts indicate that LeNet5 has made more mistakes, while the Blue parts indicate that Spiking-LeNet5 has mainly made mistakes. It can be noticed that the third strip is predominantly Red, indicating that Spiking-LeNet5 generally performs better.

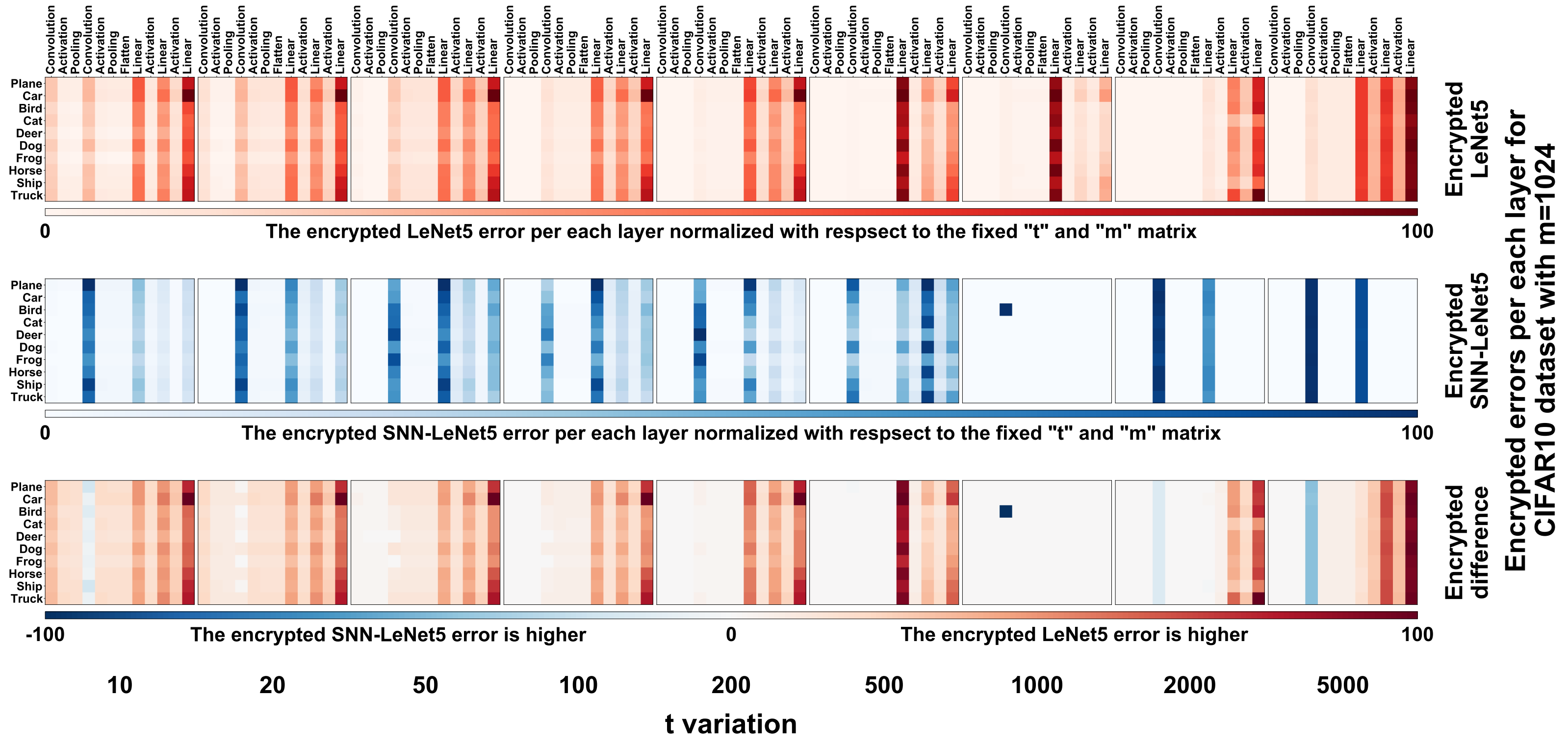


Fig. A.18 Errors layer-by-layer with CIFAR10 and m = 1024. The top **Red** strip represents the errors in the layers of the LeNet5, the **Blue** strip in the middle represents the errors in the layers of the Spiking-LeNet5. The last strip at the bottom represents the difference between the errors in the layers of LeNet-5 and Spiking-LeNet5, where the **Red** parts indicate that LeNet5 has made more mistakes, while the **Blue** parts indicate that Spiking-LeNet5 has mainly made mistakes. It can be noticed that the third strip is predominantly **Red**, indicating that Spiking-LeNet5 generally performs better.