POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Digital twins management with blockchain technology: a case study for a new protocol implementation

Tesi di Laurea Magistrale in
Aeronautical Engineering - Ingegneria Aeronautica

Author: **Niil Petruccioli**

Student ID: 905637
Advisor: Prof. Sergio Ricci
Co-advisors: Stefano Francesco Pitton
Academic Year: 2022-23

Acknowledgements

I would like to express my most sincere gratitude to all those who have made this possible.

First and foremost, I would like to thank my supervisor, Prof. Sergio Ricci, for this great opportunity, for their patience, support, and mentorship throughout this journey.

Secondly, I am grateful to MadeInBlock S.r.l., who allowed me to work on this thesis during the internship at their company. What I learned during this period has been fundamental.

Next, I would like to thank Stefano Francesco Pitton, co-supervisor, colleague, and friend. Your support, guidance, and constructive feedback have been invaluable and essential for this work.

Then, my deepest gratitude goes to Francesca. Without your love, support, encouragement and understanding all this would not have come to fruition.

Lastly, I am deeply grateful to my family. You always supported me and gave me strength. Without you all of this would not have been possible.

Abstract

Digital twin technology is a relatively new concept that has gained significant attention and traction in recent years. A digital twin is a virtual replica of a physical object, process, or system, bidirectionally connected to the physical counterpart, called physical twin. The fundamental principle of a perfect digital twin is that any information that would be available inspecting the physical twin is also available through the digital twin diagnosis and prognosis capability. A system difficult or impossible to be physically inspected, such as a vehicle in a space mission, can be inspected through its digital twin and input given to change the digital twin state will be reflected on the physical twin. Digital twins are powerful instruments but security problems linked to the data integrity of the digital models have to be taken into account. A malicious tampering of a digital twin may cause completely inaccurate and misleading predictions of its physical counterpart state, leading to serious security problems and potential mission failure. Blockchain is a new technology that can be used to solve this security issue. Blockchain is a type of distributed ledger technology in which a common ledger, structured as a chain of blocks, is shared among network participants. Participants form a network of interconnected machines that runs state-replication software to ensure the state saved in each machine is the same. Data stored in a (decentralized) blockchain are tampering resistant since participants can compare their ledgers and detect if a ledger version is fake or not. A blockchain framework will be proposed for the management of the life-cycle of digital twins taking as an example a digital twin realized by the author from a conceptual case study of a composite plate with embedded sensors subjected to tensile load. A basic digital twin based on a neural network has been realized for the structural health monitoring of the plate, considered potentially damaged by accidental damages. The resulting model will be stored in a custom blockchain which implements a protocol developed by the author that allows the distributed training of neural-network-based digital twins over authorized blockchain nodes with automatic update of the digital twin state stored in the blockchain.
Keywords: digital twins, blockchain, progressive failure analysis, artificial intelligence

Abstract in lingua italiana

La tecnologia digital twin è un concetto relativamente nuovo che ha ottenuto molta attenzione negli ultimi anni. Un gemello digitale è una replica virtuale di un oggetto, processo o sistema, connesso bidirezionalmente alla controparte fisica, chiamata gemello fisico. Il princio fondamentale alla base di un digital twin perfetto è che qualsiasi informazione ottenibile ispezionando il gemello fisico è anche disponibile tramite l'ispezione del gemello digitale e le sue capacità di diagnostica e prognostica. Un sistema difficile o impossibile da ispezionare fisicamente, come ad esempio un veicolo in una missione spaziale, può essere ispezionabile tramite il suo gemello digitale; qualsiasi input dato per modificare lo stato del digital twin si rifletterà sul gemello fisico. I digital twin sono strumenti molto potenti, ma vanno tenuti in considerazione problemi di sicurezza sull'integrità dei dati di questi modelli. Una manomissione del digital twin potrebbe comportare predizioni completamente inaccurate o fuorvianti sulla controparte fisica che potrebbero portare a importanti problemi di sicurezza o catastrofico fallimento della missione. La tecnologia Blockchain è un nuovo tipo di tecnologia che può essere usata per risolvere questi problemi di sicurezza. Blockchain è un tipo di Distributed Ledger Technology in cui un registro comune, strutturato in una catena di blocchi, viene condiviso tra i partecipanti alla rete. I partecipanti formano una rete di macchine interconnesse che utilizzano un software di replicazione dello stato in modo che lo stato di ogni macchina sia uguale. I dati salvati in una blockchain (decentralizzata) sono resistenti alla manomissione perchè i partecipanti alla rete possono verificare l'integrità dei dati comparando i loro registri e dedurre se un registro è stato manomesso. Un'applicazione blockchain viene proposta per la gestione del ciclo di vita dei digital twin. Viene preso come esempio un digital twin realizzato dall'autore basato su un caso studio di una piastra in composito, potenzialmente danneggiata, della quale si ricava un digital twin basato su rete neurale per il monitoraggio strutturale. Il modello risultante viene gestito da una blockchain che implementa un protocollo sviluppato dall'autore per il training distribuito di gemelli digitali basati su reti naurali con aggiornamento automatico delle modifiche nello stato della blockchain.

Parole chiave: digital twins, blockchain, progressive failure analysis, intelligenza artificiale

Contents

Introduction

Digital twin technology is a relatively new concept that has gained significant attention and traction in recent years. Publications have escalated exponentially between 2010 and 2020. A digital twin is a virtual replica of a physical object, process, or system that extends the dimension of its physical counterpart, the physical twin, to the virtual space, realizing a bidirectional connection in which one may influence the other. A digital twin must have a model capable of performing diagnosis and prognosis of its physical twin (through a model of the physical twin) and connections to establish automatic data flow between the two.

As the digital twin technology evolves, issues about the security of these advanced models have to be considered. In the era of Industry 4.0 blockchain technology can help enhance the protection of these models from malicious tampering and unauthorized changes.

From a conceptual case study of a damaged composite plate, it has been possible to derive a digital twin model for the structural health management of the plate in exam, and to use this model to develop a blockchain for the life-cycle management of the digital twin, including distributed training to be used when more data will be available after in-service collection by the physical twin sensors.

The thesis outline is the following.

In Chapter 1 the digital twin technology is reviewed. In this chapter will be discussed the evolution of the concept and the state of the art in terms of models used to generate digital twins and the application fields of the big players in the industry.

Chapter 2 reviews the blockchain technology. In this chapter the fundamentals of the technology will be presented along with their evolution.

In Chapter 3 the conceptual case study will be presented. It consists of a potentially damaged composite plate from which a digital twin model will be realized. The composite plate has been modeled using data from an experimental study of a specimen that has been found in literature. First, the results of the preliminary numerical analysis of this specimen will be reported, then the undamaged condition of the plate will be analyzed.

Chapter 4 contains the generation of the digital twin model. It is a feedforward neural network model trained from a database obtained from numerical analysis in which the composite plate is considered in different damaged states.

In Chapter 5 will be presented a blockchain application created for this digital twin life-cycle management (but also adaptable to any digital twin in general). This application implements a protocol for distributed training analysis on the network nodes that can be used by authorized users when the digital twin neural network model has to be retrained.

Chapter 6 contains the conclusions and a description of possible future implementations starting from what was developed in this thesis.

This thesis was developed during the author's internship at MadeInBlock S.r.l. .

1 | Digital twins

Digital twin technology is a relatively new concept that has gained significant attention and traction in recent years. The term *digital twin* was first introduced by NASA in a 2010 roadmap report, but concepts origins from Dr Michael Grieves who described it in 2002 [20]. The backbone of this concept is having an available replica aiding in the decision-making process and was already implemented by NASA in the Apollo program in the 1970s when NASA built two identical space vehicles and one was left on earth to reflect the state of the one sent in space: it was not a digital twin nonetheless it was an identical twin available for inspection at any time [51].

A digital twin is a virtual replica of a physical object, process, or system, where the physical counterpart is called *physical twin*. This replica should not be a simple mock of the object: it must be bidirectionally linked to its physical counterpart, meaning that each modification on the physical twin should be reflected on the digital twin and vice-versa.

Interest in digital twins started blooming in 2015 and saw in recent years an exponential trend [31, 45]. Key-factors of this evolution have been the technological advancements enabling the Industry 4.0 revolution, which are Internet-of-Things (IoT), big data, Artificial Intelligence (AI) and cloud computing [31]. All this has been possible thanks to the constant evolution in computational power during the years, that enabled all these new technologies.

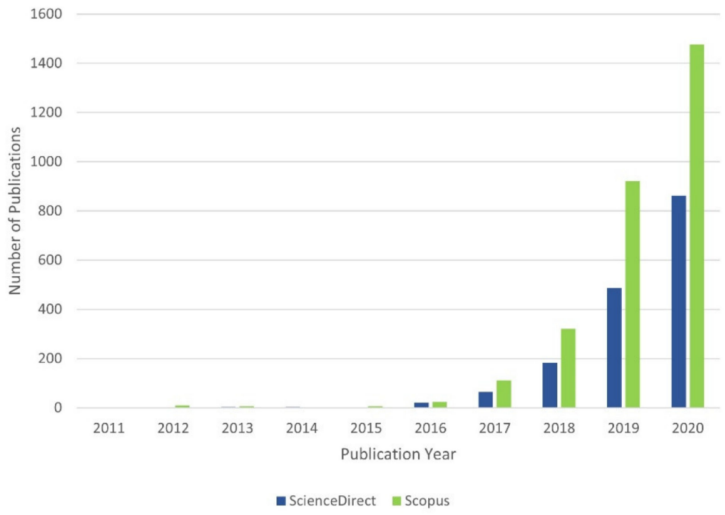


Figure 1.1: Publications related to digital twins between 2011 and 2020 from ScienceDirect and Scopus. From [45].

1.1. Evolution of the concept

Dr Michael Grieves proposed in 2002 what will be then called "digital twin", calling it "Mirrored Spaces Model". His work was in relation to a general Product Lifecycle Management system, and he theorized a virtual space linked to the real space in a bidirectional way, in such a way that a physical system can be linked to its virtual mirror. "Mirrored Spaces Model" has been changed in 2006 to "Information Mirroring Model" by Grieves [20, 45].

NASA, as Singh et al. [45] points out, was the first to use the term "digital twin" in its 2010 draft-version of technological roadmap [44], precisely under the roadmap section of 2023-2028 Simulation-Based Systems Engineering. NASA described the digital twin as "an integrated multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its flying twin.". The fundamental traits of NASA vision for digital twins were their "ultra-realistic" nature and integration with data coming from "on-board integrated vehicle health management (IVHM) system, maintenance history, and all available historical/fleet data" to allow on-board systems "mitigating damage or degradation by recommending changes in mission profile to increase both the lifespan and the probability of mission success". NASA's digital twin concept arises from the complex nature of its missions, where the object that is sent into space is very expensive and almost always unreachable by humans or the inspection is too dangerous to be performed.

Moreover, NASA highlighted the importance of having a reliable simulation technology to make decision choices in the designing phase where they are simpler to introduce and less expensive than modifications to be introduced in the production phase. That concept results in what is now called Digital Twin Prototype.

NASA recognised as key enabler for digital twins the developing and implementation of the "best-physics model", that defines as "the most accurate, physically realistic and robust models" that can create an exact replica of the physical twin. Application NASA sees for this technology are (i) the possibility to simulate future missions before the actual start of the mission, (ii) monitoring the system during operation through its digital twin, (iii) using the digital twin to perform prognosis and diagnosis on its physical counterpart, and (iv) use of the digital twin for certification through simulation, in conjunction with continuous health monitoring of the physical twin.

In 2012 NASA and United States Air Force (USAF) [19] defines further that the digital twin has to be based upon ultra-high fidelity models which include also the atomic scale in order to capture the material microstructure and manufacturing defects.

Even if the original concept of digital twins is theorized as based upon extremely sophisticated physics-based models, in the 2010s, as Li et al. [31] points out, there has been a constant improvement of data-driven models supported by the continuous enhancement of computational power, leading to wide applications of these models for diagnosis, manufacturing, and decision-making processes due to their classification performance and high adaptability.

Singh et al. [45] and Li et al. [31] found that publications about digital twins saw exponential growth starting blooming in 2015. This has however generated confusion about the concept of digital twins. Many authors (roughly half of the publications as Singh et al. [45] reports) were in fact implementing a "digital shadow" or a "digital model" instead of a digital twin while stating they were indeed realizing a digital twin. This distinction has been defined by Kritzinger et al. [28] based upon the implementation level of the bidirectional connection between physical and digital world: (i) *digital model*, where no connection is established; (ii) *digital shadow*, where only the connection from physical to digital is implemented (the digital twin can't reflect automatically itself on the physical twin); (iii) *digital twin*, where bidirectional connection is realized. Confusion is also worsened by the different various terms used by authors to describe digital twins in their own terms. As Singh et al. [45] reports, these are: "digital model", "layout", "counterpart", "doppelganger", "clone", "footprint", "software analogue", "representation", "information constructs", or "simulation" of its physical counterpart.

Digital twin concept was born as a general PLM framework but its first instantiation and definition was in the aerospace field. Since then it has been applied to many other fields and "aircraft", "vehicle" and "airframe" were replaced by "system", "machine", "product", "object", "entity", "asset", "device" [45]. Today digital twins are applied in many fields. Li et al. [31] finds out that literature focuses on three particular areas which are smart manufacturing, healthcare and smart cities; Thelen et al. [48] analyzes the growth in research interest among different fields and finds that manufacturing is the leader followed by mechanical, civil, aerospace, energy and healthcare fields.

In 2016 Michael Grieves and John Vickers [20] give their definition of digital twin: "The Digital Twin is a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level. At its optimum, any information that could be obtained from inspecting a physical manufactured product can be obtained from its Digital Twin.". This definition is clear about the potentiality of a well-realized digital twin, and it's left general for any physical object, but lacks the part in which the bidirectionality has to be enforced.

As the concept has grown in popularity, different nuances have been proposed. Michael Grieves and John Vickers [20] described two different digital twins: (i) *Digital Twin Prototype* (DTP) that exists before the physical counterpart is created and it is used to digitally test the object and chose the best object's parameters easily and inexpensively at design phase; (ii) *Digital Twin Instance* (DTI) that is created when the physical counterpart is realized, realizing the bidirectional link between the two entities. Singh et al. [45] also finds that a digital twin has been applied at different system levels: (i) Unit level, where the digital twin is linked to a material or component of the entire system; (ii) System level, where many unit-level digital twins are condensed to represent the whole system, being it for example an aircraft, a production line or a factory; (iii) System-of-Systems level, which integrates many system digital twins of different fields to follow the physical entity throughout its life-cycle. Many other nuances of the digital twin concepts can be found in [45] based on the maturity level. Depending on decision capabilities the following categories can be made: (i) *Pre-Digital* Twin, which is a prototype digital twin; (ii) *Digital Twin*, which assists with high-level decision-making; (iii) *Adaptive Digital Twin*, which can learn from the preferences and priorities of human operators when providing solutions to physical twin problems; (iv) *Intelligent Digital Twin*, which can recognize patterns in the operational environment and produce more sophisticated and autonomous solutions than the Adaptive Digital Twin. Li et al. [31] describes also the *Cognitive Digital Twin* as a particular kind of intelligent Digital Twin which possesses abilities of perception, attention, memory, and reasoning and can operate independently of the physical twin. [31]

describes also the *Basic Digital Twin*, which is the entry-level for digital twins, having the capability of simulation, real-time state synchronization, and data collection from its physical but limited to coverage only a certain aspect of the physical twin, not having the possibility of evolving automatically with its counterpart when the environment changes and lacking the ability of autonomous problem-solving thus being not independent of operator interactions. [31] points out that among all the different levels of sophistication currently the most used is the Basic Digital Twin.

1.2. State of art

1.2.1. Digital twin modelling

Digital twins models divide into three categories: (i) physics-based models; (ii) data-driven models; (iii) hybrid models.

Physics-based models are based upon the mathematical links between the physical variables that determine the underlying physics of the model. Their strength resides in this aspect: they are based upon physics. The downside is that prior knowledge is needed to model correctly the problem, and this highly affects the model quality. High efforts have to be made in designing experiments to understand the physics at the problem base and researching the best way to model the problem.

Data-driven models are based upon observed data input-output relations rather than on the underlying physics principles. They are typically developed through machine learning algorithms, a field of Artificial Intelligence. These models are essentially black-boxes when it comes to reconstructing the input-output links, in the sense that since these are not based upon physical principles the relations between variables the model produce become most of the time completely detached from the natural laws. This represents the main downside of these models: when the model does not produce the expected outputs the reason behind the malfunction becomes hard to understand. The other main drawback is that the quality of the model is heavily dependent on the quality of its input data: since these algorithms learn from the data they see, they will learn wrong input-output relations if training data quality is low, resulting in a bad model. On the other hand, pros of data-driven models are astonishing. They enable modelling of very complex systems easily, and offer a way to model systems even in cases where the underlying mechanisms or relationships are not fully understood. Moreover, these models can continuously learn and improve as new data becomes available and comes with natural prediction capabilities since the intrinsic ability to make predictions on new, unseen data based on the

relationships learnt.

Hybrid models combines physics-based models and data-driven models, embedding known physical laws and principles into the modelling process. By incorporating physics constraints, hybrid models can capture the system's behaviour more accurately and handle cases where limited or noisy data are available. Moreover, the resulting model has better interpretability thanks to the physics embedded. Two hybrid modelling approaches are described by Li et al. [31]: (i) physics-informed neural networks (PINN) incorporate known physics equations into the training process, calculating the loss function from these equations; (ii) data-driven physics-based model (DDPBM), where physics-based sub-models are used for simulation and a data-driven model connects the sub-models.

1.2.2. Application fields

In the aerospace field, General Electric owns four patents for digital twins, two of which are for wind farms, invented a digital twin for a wind farm and developed an augmented reality interface to manage multiple digital twins [47]. NASA, EASA, U.S. Air Force and Royal Canadian Air Force (RCAF) are all working on applying DTs to Operation and Maintenance [31]. USAF developed CFD model-based DTs for F-22 development [31]. Airbus developed an assembly line DT to monitor the production process [47]. Royal Canadian Air Force used digital twins for their fleet management [31]. In the automation field, Siemens patented a software that enables a machine to interact with humans and interpret human behaviours realizing a digital twin of a human that can be plugged into the automation control system [47]. In the energy field, Siemens developed DT for the planning, operation, and maintenance of a power grid system in Finland. British Petroleum developed DT for oil/gas facilities monitoring and control, one of which is for a production facility in Alaska [47]. In the civil infrastructures field, Siemens developed a DT for a wastewater treatment plant to monitor pipes in real-time [47]. In the mechanical field, General Electric developed a DT for a locomotive life-cycle management from which also conditions of each component can be obtained in real-time [47]. In the healthcare field, GE Healthcare developed a DT to manage hospital bed planning and work allocation [47]. Dassault Systems launched the Living Brain and Living Heart projects which use DTs to reproduce the patient's brain and heart [46].

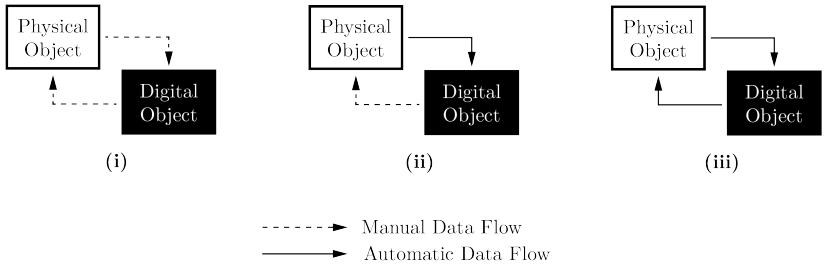


Figure 1.2: Digital model (i), digital shadow (ii) and digital twin (iii) representation. Adapted from [28].

2 | Blockchain technology

Blockchain technology is a relatively new technology, born in 2008, and constantly evolved over the years. Blockchain started with financial applications, but it can be a versatile technology. In this chapter, the technology fundamentals will be reviewed along with an overview of the evolution of the technology and its potentialities.

2.1. Definition

A common term used to describe a blockchain is *distributed ledger*. Distributed ledger technology (DLT) is a multi-party system in which a shared database (the ledger) is managed and maintained with no central operator or authority, despite parties who may be unreliable or malicious ("adversarial environment"). Blockchain technology is often considered a specific subset of the broader DLT universe [23, 39].

A blockchain can be defined as a distributed system based upon deterministic state-machine replication, immutability of state history and decentralized administration.

It is a distributed system since it is built upon a network of interconnected machines in a peer-to-peer architecture. Machines joining the network are called "nodes". Each node of the network is a replica of the same state-machine, so it has the same state of the other nodes. Replicas are created by state-machine replication.

State-machine replication is a technique used to create fault-tolerant services [43]. Where security does not need to be enhanced, the simplest way of implementing a network service is using a client-server architecture: client asks the server for data or to perform some specific action, and the server satisfies the client's request. In this implementation the server is a critical part: if it faults, then the service will be no longer available. Server can fault by simply going offline, or worse by arbitrarily censuring the client(s) requests. The server in this implementation represents a single-point of failure, i.e. a point of the service that will bring the system to total failure if it fails. To solve this singularity the state-machine replication is implemented.

A finite state-machine is a system that contains a finite number of states, and at any

given time it can be in one only of its states. The finite state-machine is here intended as a Mealy machine, i.e. a finite state-machine whose next state that can be reached depends only on the current state and the input received. After an input is received, the output may be a rejection of input, i.e. no modification of the machine state occurs, or the transition from the current state to a new state [8, 29].

Transition from one state to another must be deterministic: applying the same inputs to the same state must always produce the same output. Under this assumption, state-machine replication can be performed. Replicas of the same machine will be interconnected forming a network of nodes in which a broadcasting mechanism will ensure that each node receives the same input so that each node will transit to the same output. In this way the whole network can be perceived as a unique entity evolving after an input is provided.

The state of a blockchain is the state of the data stored in the blockchain. A blockchain can store whatever data it is programmed to manage, and its storage can be seen as a database saved in the physical memory of each node.

To coordinate, each machine replica must receive the same inputs and must process these in the same order. Inputs are in fact non-commutative, meaning that generally changing the order of inputs does not lead to the same output. Inputs given to the blockchain are called "transactions" and are provided by the users of the blockchain. Users send transactions when they want to change the state of the blockchain. A transaction contains information about what to change. Transactions are ordered and grouped in blocks. Once a new block is created it is broadcasted to all the network nodes and these will process the transactions it contains.

The state replication implies that two problems must be solved: (i) each node must receive blocks in the same order, and (ii) there must be an agreement on which transactions have to be included in the block. These two problems are solved by a fundamental part of a blockchain, its consensus algorithm. The most used are called Proof-of-Work and Proof-of-Stake. These will be discussed later in section 2.5. The implementation of the consensus algorithm is called *consensus engine*. The broadcast protocol that is generated by the consensus engine is an *atomic broadcast*. In this kind of broadcast, all nodes of the network receive the inputs in the same sequence and when the network nodes receive an input: (i) all correct nodes accept the inputs and reach the same state, or (ii) all correct nodes reject the inputs and abort the state change [10].

Once consensus is reached the new block is added to the blockchain history: the new block is appended to the previous block, creating a chain of blocks. When the block is created

it cannot be modified and the full chain integrity is based upon cryptographic techniques that will be discussed later. Nodes of the blockchain store the full blockchain history and through cryptography it is possible to state and verify that no data manipulation is possible on data stored on-chain. Being the state transition deterministic, one can obtain any state at time t from the known initial state at time t_0 applying all transactions that happened between time t_0 and time t.

The only way that makes it possible to modify the data history is by taking control of the blockchain and rewriting the history from a given time to the current time. This is indeed possible if the administrator of the blockchain is a centralized entity that has full control over it. Blockchains, as commonly intended, rely for this reason on decentralized administration.

Blockchains can be of three kinds [6]: (i) public blockchains, i.e. blockchains where data are publicly available to read and any users can write without censorship of transactions and take part in the consensus process; (ii) private blockchains; i.e. blockchains that are managed by a private entity in which only authorized users can take part; (iii) consortium blockchains, i.e. blockchains managed by a group of enterprises or organizations that are a hybrid between public and private ones.

Public blockchains must be decentralized for obvious reasons, otherwise, they would be completely unreliable and no different from classic client-server architecture where service could be interrupted or data altered. Private blockchains are not decentralized: their administration depends on the private group and can be described as a "traditional centralized system with a degree of cryptographic auditability attached" [6]. Consortium blockchains are a hybrid between public and private, and decentralization depends on how many different entities participate in the consensus process. consortium could easily take control of data history and change anything in the blockchain. This could ease management operations for the consortium in case some particular situation arises and all network participants agree, or it could be a complete disaster if arbitrary data manipulation is made under the hood.

2.2. Cryptography in blockchains

Cryptography is the backbone of blockchains. Three things are extensively used: hash functions, Merkle trees and digital signatures.

2.2.1. Hash functions

Hash functions are used everywhere: they are for example in blocks generation, in addresses generation, and in digital signatures. A hash function is a function that maps an arbitrary-length message to a fixed-length message digest. It also exists hash functions which produce variable-length outputs but are not used in blockchain applications. Digest is also called checksum or simply hash. A not secure hash function may produce hash collisions, i.e. it may associate the same digest to two different messages, or, when the hash function is said to be invertible, the original message could be derived from its hash. A hash function to be secure for practical applications must always preserve two fundamental properties: (i) *Collision resistance*, so it must not produce hash collisions; (ii) *One-wayness*, so it must not be invertible. What is really meant for "it must not" is that it should be computationally infeasible for an attacker to find a way to break the function hash property in a reasonable amount of time [1, 11]. Hash functions, to be resistant to inversion, produce digests that are really different when the original message is only slightly different. An example is here reported for the SHA-256 hash function in table 2.1.

Original message	SHA256 digest
Hash functions	`63090708fdd4b33f4bb5920a232bfbfc5f85e7c3e80021ec5a03bcd3377271e7`
Bash functions	`cf07c4d4c07bda647be4808f26e5aef90bd53efc5baa1ebdf56c7b2437ccd22f`
Hash function	`9a8d115eeaf6b78f876c3c822b8209c3cca4d79d76f2af59fa36ec4c2860c4e6`
hash function	`04ce4af53ae1b0f46451715ffc43092eab5b23330a6584462d0723732b147c4b`

Table 2.1: Very different digests produced by very similar messages.

Hash functions are widely used today in digital communications as a way to ensure data integrity, secure authentication and digital signatures. Hash functions are used in protocols that require authentication, such as TLS, SSL, SSH, PGP, S/MIME, and IPsec. They are also used in version control systems such as Git [12].

Hash Function	Maximum Message Length [bits]	Digest Length [bits]
MD5	2^{64}	128
SHA-1	2^{64}	160
SHA-256	2^{64}	256
SHA-512	2^{128}	512

Table 2.2: Properties of common hash functions.

2.2.2. Merkle tree

Merkle trees, developed by Ralph Merkle in his PhD thesis at Stanford University in 1979 [33], are trees data structures used to represent data and to prove that a datum is in the data. In Merkle trees each extremity node (leaf) is labelled with the hash of a data group's datum, and each inner node is labelled with the hash of the combination of its children labels: the root, called Merkle root, will represent the hash of the entire group. Demonstrating that a leaf is part of the tree will require few inputs compared to the overall data group size: the leaf's hash and the hashes of the inner branches, which are the nodes encountered in the path from the leaf to the root [30]. This type of demonstration allows to prove that a datum is part of a data group without the need of sending or revealing the full data group. The proof in fact needs only to reveal the datum, the Merkle root and the hash of the inner sub-trees involved in the proof. Figure 2.1 pictures a Merkle tree and the Merkle proof for a specific datum.

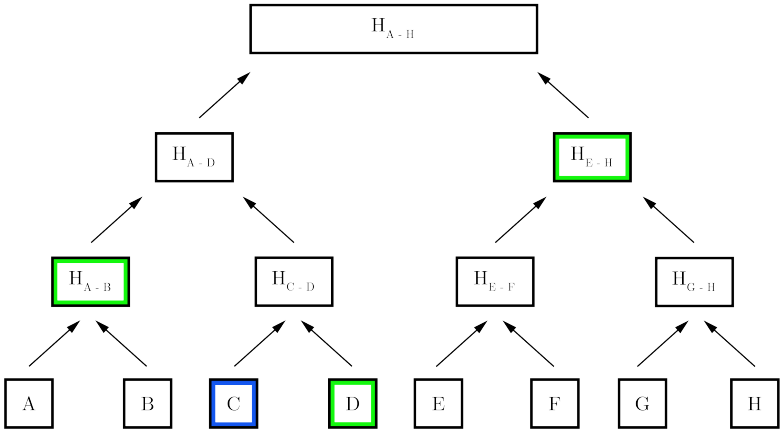


Figure 2.1: A Merkle tree structure. Blue: data one wants to prove to be inside the Merkle root. Green: data needed for the proof other than the blue one. Adapted from: [17].

Merkle trees in blockchains are used to build the Merkle root of the transactions, which will be saved in block information along with the transactions of the block. Merkle proof can be used by clients who need fast verification that a transaction is included in a block and for speed or memory reasons can't download the entire block and build the Merkle root. In this situation what can be done is to provide the client the block header from which it will know the Merkle root, and provide the Merkle proof.

Merkle proof can also be used to ensure data integrity without storing the data on the blockchain, but only the Merkle root of the data. The proof that a datum is inside the data, so inside the tree identified by the stored Merkle root, will be provided sending the

datum and the hash needed to compute the Merkle root [17]. It is worth noting that the original work of Ralph Merkle was supported, among others, by the U.S. Air Force Office of Scientific Research [33].

2.2.3. Digital signatures

Digital signatures are constantly used in blockchains to digitally sign transactions or during block-proposing phase in Proof-of-Stake consensus algorithms. Digital signatures are based on public-key cryptography, also known as asymmetric cryptography. Asymmetric cryptography evolves from symmetric cryptography.

2.2.3.1. Symmetric cryptography

For symmetric cryptography it is intended that a single secret key, called also "key", is involved in the encryption and decryption of messages. A cryptographic algorithm for encryption and decryption is called *cipher*. Encryption is based upon a one-to-one mapping function E between the original message m, called *plaintext*, and its encrypted version c, called *ciphertext*, such that $c = E(m)$. Since $E()$ is a one-to-one mapping, its inverse $D()$ must exist such that $D(c) = m$ [49].

Let's consider an alphabet of 26 letters. The simplest kind of one-to-one mapping is a substitution table that associates each letter of the alphabet with a letter of the same alphabet. Knowing the substitution table it is possible to transform the plaintext into the ciphertext and vice-versa; in this example the secret key is the substitution table. This kind of encryption method is called *substitution cipher*, and it was developed by ancient civilizations: a kind of this encryption scheme, called "Atbash" can be found in the Bible in the Old Testament (written around 600 BC). When actual words are used as keys for the encryption of an alphabetic text, the encryption usually works by making the original text and the key the same length and then adding to a letter of the plain text the corresponding letter of the key (in the modulus of the alphabet). Doing the same operation for the ciphertext will return the original message. An example of that is the Vigenère cipher.

Many cryptographic systems have been broken, meaning that it has been found a way to compute the key from the ciphertext. Cryptographic security divides into two categories. A system can be computationally secure when it is infeasible to break with a feasible amount of resources (currently available computational power and time). Computationally secure algorithms will eventually be broken by an attack with unlimited computational power. A system that instead will resist even an attack with unlimited

computational power is defined *unconditionally secure* [14]. The only known unconditionally secure algorithm is the *one-time pad*. It is a symmetric encryption scheme, but it requires a different key each time, of length equal to the message being encrypted and randomly chosen with perfectly uniform distribution among the key space [14]. Implementing this scheme was already impractical for many applications in the '80s, and it is impractical as well in blockchain applications where each second tens to hundreds of transactions must be digitally signed and verified.

Another problem with symmetric encryption is that the secret key must be known both to the sender and the receiver. Symmetric cryptography relies in fact on a secure way to share the secret key. If the communication channel is compromised, then all the efforts to build a strong encryption algorithm are useless. Blockchains are systems based on finding a way to operate in a trustless environment between many participants, and symmetric encryption is not efficient in this context.

2.2.3.2. Asymmetric cryptography

Due to the problem of key distribution the idea behind public and private key pair arises: it would be much more secure to have a couple of keys where one, called *public key*, can be shared with anyone, while the other, called *private key*, will remain secret. This couple of keys is such that: (i) whatever is encrypted with one key can only be decrypted with the other, and (ii) it is infeasible knowing the public key to derive the private key [14]. Being this key pair available, two important things can be done:

1. A person can send a private message to another one simply by encrypting the message with the receiver's public key: in this way only the receiver will be able to decrypt it.

2. A person can hash a message and encrypt it with its private key, then send to anyone the following package: the original message, its public key and the encrypted hash (along with the specification on what hash function was used if that is not a standard). Anyone who receives this package can hash the message and decrypt the encrypted hash with the sender's public key. If the two hashes match, then this acts as proof that the message received has not been manipulated during transmission and that the sender owns the private key coupled with the public key he received: in this way the receiver can say that the sender "digitally signed" the message.

This new branch of cryptography is called asymmetric cryptography or *public-key cryptography* and was first introduced by W. Diffie And M. Hellman in 1976. They designed a scheme to exchange public keys among two users and derive from these a common

secret key to be used in symmetric encryption, called Diffie-Hellman key exchange, and also theorized digital signatures [15]. The first practical digital signature scheme was provided by R. L. Rivest, A. Shamir, and L. Adleman in 1978, known today as RSA Digital Signature [40]. RSA makes use of the fact that finding large (e.g. 200 digit) prime numbers is computationally easy, but factoring the product of two such numbers appears computationally infeasible [14]. Today RSA is of impractical use due to the advance in computational power that makes the algorithm rely on huge public keys to remain secure. That leads to problems in storing and transmitting keys among the network for modern applications. This lead to more efficient digital signature algorithms, such as elliptic curve ones.

2.2.3.3. Elliptic curve digital signature

Elliptic Curve Digital Signature Algorithm (ECDSA) offers security equivalent to RSA using much smaller key sizes [22]. ECDSA is based upon Elliptic Curve Cryptography (ECC) that was developed independently by N. Koblitz and V. Miller in 1985 [22]. An accurate description of ECC and ECDSA can be found in Kobltiz et al. [27]. In this section, an overview of ECC and ECDSA will be provided based on [2, 27, 32].

An elliptic curve E defined over a finite field F_p of p elements is the set of all solutions (x, y) in $F_p \times F_p$ to an equation:

$$E : y^2 = x^3 + ax + b \tag{2.1}$$

where $a, b \in F_p$ and $4a + 27b \neq 0$, together with a special point O called the point at infinity [27]. Point at infinity is such that for any point P in the curve $P+O = O+P = P$.

Scalar multiplication of a point P on the elliptic curve by a number m, denoted as $m \cdot P$, is defined as adding P to itself $m - 1$ times. The order of an elliptic curve is the number of the curve's point, while the order m of a point P on the elliptic curve is the value m such that $m \cdot P = O$. [32]

In nowadays modern cryptosystems p is a huge prime number. National Institute of Standards and Technology (NIST) recommends five elliptic curves for use in the elliptic curve digital signature algorithm [11], where:

$$
\begin{aligned}
p_{192} &= 2^{192} - 2^{64} - 1 \\
p_{224} &= 2^{224} - 2^{96} + 1 \\
p_{256} &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\
p_{384} &= 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\
p_{521} &= 2^{521} - 1
\end{aligned}
$$

A very important property of elliptic curves is that for two points $P1$ and $P2$ on the elliptic curve, the addition of $P1$ and $P2$ is a point $P3$ that is in the elliptic curve. This leads to a fundamental trait of ECC: the scalar multiplication of a point P on the elliptic curve and a positive integer k results in another point Q on the elliptic curve. The inverse of this operation, i.e. computing k known P and Q, leads to the Elliptic Curve Discrete Logarithm Problem (ECDLP) [2].

For cryptographic applications a point called *generator point* G is chosen on the elliptic curve. For security reasons, only generators whose order is a prime number are used in commercial deployments[32]. Most standards also require that the cofactor, which is the result of dividing the number of points of the curve by the order of the generator, must be 1 or a small number such as 2, 3, or 4 [32]

When an elliptic curve is chosen (in terms of coefficients a and b and generator point G), a private-key is then a number k in $[1, n-1]$, where n is the order of the generator point, and the related public-key P is the multiplication between k and G:

$$P = kG \tag{2.2}$$

This relation is the core concept behind the digital signature scheme: converting a key to another is trivial for the private-key owner, but infeasible for any attacker. The best-known algorithm to solve ECDLP is the *rho*-method that takes about $\sqrt{\pi n}/2$ elliptic curve additions [27]. No subexponential-time algorithm is known to solve the ECDLP in a properly selected elliptic curve group [2]. Blockchains based on ECC typically relies on 256 bit keys with cofactor 1, meaning that there are roughly 10^{77} possible different private-keys (while the number of atoms in the universe has been estimated as something between 10^{78} to 10^{82}). Koblitz et al. [27] in 2000 concluded that 180 bits were sufficient for medium-term security.

Bitcoin [34] and many other blockchains uses secp256k1 which parameters are listed in table 2.3.

Parameter	Value
p	115792089237316195423570985008687907853 269984665640564039457584007908834671663
a	0
b	7
G	550662630222773436695787188951685343262 506034537775941755001873603891167292 40, 326705100207588169780830851305070431844 712733806592432759389043357573374824 24
n	115792089237316195423570985008687907852 837564279074904382605163141518161494337
h	1

Table 2.3: secp256k1 elliptic curve parameters.

2.2.3.4. Addresses

An address represents an user of the blockchain application. The address is a string derived from hashing the public key. It usually starts with a human-readable part ($bc1$ for Bitcoin [34], $0x$ for Ethereum [5], $cosmos1$ for default Cosmos [21] blockchain implementations) and is then followed by alphanumeric characters.

Since the address is derived from the public key hashing it is unique. When a transaction is made, a block will record the transaction and the address who signed it. A user can sign transactions from an address as long as it owns the private key linked to that address. Being the private key of an address lost, the possibility to send transactions from that address will be lost as well, meaning that whatever functionality that requires only the authorization of that given address will no longer be available.

2.3. Blocks

A *block* is the fundamental entity of a blockchain. Blocks collect transactions sent to the blockchain. Each block of a blockchain is linked to the previous block, forming a chain of blocks, from which the name.

A new block is produced after some time is passed from the last block generation. The average time needed to generate a new block is called *block time* and is blockchain specific. In Proof-of-Work blockchains (described in section 2.5.1) the block time is not fixed

since it depends on the difficulty of the cryptographic puzzle and the available computational power to solve it. To keep the block time as much as constant the difficulty is periodically adjusted. In Proof-of-Stake blockchains (described in section 2.5.2) the block time is constant, defined by the blockchain developers. It is worth noting that block time may experience fluctuations also in PoS blockchains in situations of network congestion, where an unusually high load of transactions has to be processed. In many blockchains there is a limit on the maximum block size that is in the order of some megabyte. Since transactions are normally lightweight (250 bytes), a block can contain hundreds or thousands of transactions. It may also contain zero transactions if the blockchain has not received any transaction during the block creation time. Today public blockchains do not wait for at least one transaction to be received. This implementation decision has been taken primarily because block creation comes with rewards from the blockchain protocol. This is a concept tightly connected with the fundamental need public blockchains have to be the more decentralized as possible: a secure blockchain is based on many different entities which participate in consensus and verify the operations of other entities in a trustless environment; this blockchain will be worth to be used, but maintaining a high level of security is a heavy task, so the blockchain protocol will reward participants for their efforts in the form of tokens that can be sent from a user to another (from which the term *cryptocurrency* come), and can also be used in the protocol's decision-making process (called *governance*) when updates have to be done.

The usual structure of a block contains a *block header* and a set of transactions. The block header contains identifying information for the block, which usually are block hash, previous block hash, block id (the block "height"), block timestamp (Unix time of block creation), block proposer/miner, and the Merkle root of the transactions included in the block.

Block time is also called *latency*, it is the amount of time needed for a user that sent a transaction to have the certainty that the transaction has been included in the block. The more the block time is high, the more the user will have to wait until the next block is added to the chain. Blockchains then have to make a compromise between transaction speed (the latency) and block size: setting a limited block size for a fast blockchain helps in handling network congestion (the downside is that the transaction may be rejected if block size limit is reached, so the user would have to try again, and this is something that must be handled in the design phase).

2.4. Blockchain layers

A blockchain can be seen as having three different layers:

1. Application layer
2. Consensus layer
3. Networking layer

Application layer defines what the blockchain state replication has to be performed. It contains the logic that transforms an input into an output. Different blockchains have generally different application layers. State-trasition is managed by the application layer.

Consensus layer defines how to reach agreement on what the next state has to be. The consensus layer will send the new block to the application layer when agree is reached upon inputs and inputs order (thus when the block transactionsm are defined and agreed upon).

Networking layer defines the protocols to exchange data between the network nodes.

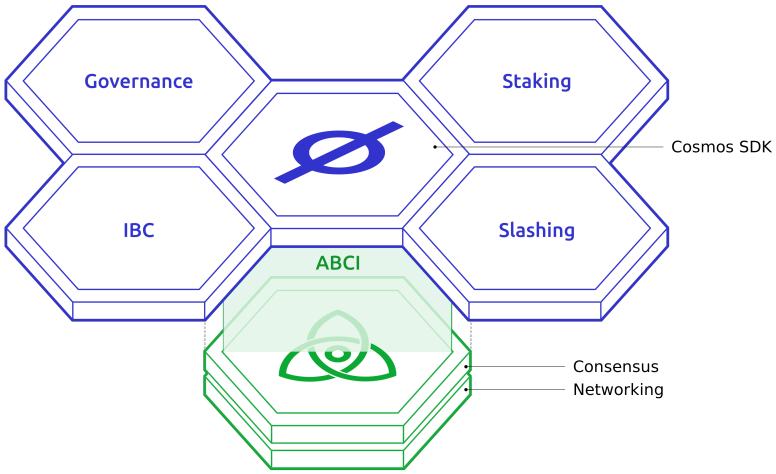


Figure 2.2: Cosmos typical blockchain layers. Application layer on top, consensus and networking layer on bottom. From [9].

2.5. Consensus

Consensus is the fundamental problem to solve for state machine replication, in which the multiple nodes of the network must agree on what the next state has to be. Reaching agreement is trivial when both the participants and the network are completely reliable, but in a real application this is not the case [18]: the participating processes (the nodes)

may crash or the network may be unreliable so that messages are not broadcasted as they should be, or processes may act maliciously sending distorted messages or arbitrarily censoring messages or even attempt to control the chain or rewrite the history to take profit at the expenses of the honest participants [16, 42, 50].

2.5.1. Proof of Work

Bitcoin [34] was the first distributed system for "electronic cash" payment to implement a solution in the peer-to-peer system to the problem of double spending (spending the same amount more than once). In centralized systems the problem does not exist: the central database is in charge of tracking users' balances, but in a peer-to-peer network where nodes can act malicious, the solution was not obvious. Bitcoin invented the concept of blockchain, in which transactions are saved in blocks linked one to another. Consensus mechanism it implemented is a difficult cryptographic puzzle that has to be solved, and once solved it gives the right to the user who solved it to add a new block to the chain (users who participate in this process are called miners for this reason). This mechanism is called Proof-of-Work (PoW). Moreover, to prevent forks in the chain that could rewrite the transaction history, it introduces the *Nakamoto law*, called after the unknown Bitcoin creator Satoshi Nakamoto. The Nakamoto law makes miners consider as the right one the longest chain of blocks: honest miners will continue to add blocks to the honest branch of the chain, while malicious miners will have to work creating new blocks to reach the honest chain length. Since creating blocks requires great computational power this is not inviting, and will eventually fail if malicious miners do not control more than half of the total computational power because the honest branch will evolve faster [42]. Nakamoto law solves the double-spending attack, in which a user pay for some goods that he receives, and then alters the chain history to revert its balance.

What it is actually meant for "cryptographic puzzle to be solved" in Bitcoin is finding a number, called *nonce*, to be included in the block header such that the resulting block hash starts with a protocol-defined minimum number of leading zeros. Actually these zeros are 18, where the total length of the hash is 64 hexadecimal characters. There exists no efficient way to solve this problem, meaning that it must be solved by trial and error by miners, leading to the huge computational power needed and to a block time of ten minutes. The number of minimum leading zeros is a parameter that is dynamically adjusted to maintain the block time as much as constant.

2.5.2. Proof of Stake

Bitcoin is extremely slow and energy-consuming. However, it maintains fiercely these fundamental traits, acting as a secure bay for exchanging "digital gold". Many blockchain projects are born inspired by Bitcoin, and many of them required faster transaction speed and lighter resources. For this reason new consensus algorithms have been developed that are faster and do not require computational power to solve the PoW. These algorithms require the *staking* of something: users "put at stake" something that acts as proof that they will behave honestly in the protocol, otherwise they will be punished by the protocol which will destroy totally or partially what they staked (performing what is called *slashing*). These algorithms are called Proof-of-Stake (PoS). In PoS, usually a block proposer is chosen by the algorithm among those who have a stake. Variants of PoS in which only a delegated subgroup of users participates in the consensus protocol are called Delegated-Proof-of-Stake (DPoS). Variants in which the stake is simply an authorization released by some entity are called Proof-of-Authority (PoA).

2.5.3. Tendermint consensus

Byzantine fault is a type of fault that can occur in a distributed system where some components may deliberately attempt to play against the system by performing malicious actions, such as sending incorrect or conflicting information or deliberately delaying or dropping messages. Systems that operate in this condition are called Byzantine fault-tolerant systems. *Practical Byzantine Fault Tolerance* (PBFT) is a consensus algorithm designed for Byzantine fault-tolerant systems. With PBFT the system continues to operate correctly and reaches a consistent agreement despite the presence of faulty or malicious nodes. Practical Byzantine Fault Tolerance (PBFT) solves BFT consensus by considering essentially two rounds of agreement. Consensus rounds start when a client sends to a node an input. This node, the proposer, broadcasts a request to the network to initiate the consensus rounds. The message the proposer sends is called *pre-prepare message* and nodes respond brodcasting a message called *prepare message*. When a node receives enough prepare messages (over a certain threshold) it broadcasts another message called *commit message*. When enough commit messages are received then the client request is accepted and the state changed. PBFT is considered the first high-performance consensus algorithm with an optimal byzantine fault tolerance [13].

Tendermint [4] is a consensus algorithm similar to PBFT. In Tendermint the nodes that participate in the consensus rounds are called "validators". The algorithms choses in a round-robin manner a validator that will propose a block. Then consensus is reached

similarly to PBFT rounds. First, the chosen proposer has a timeout to broadcast a new block, and if it does not provide any block in time a new proposer is chosen. Then, two phases of voting similar to PBFT are performed.

Tendermint acts as a standalone consensus algorithm. It is the application layer that is in charge to define the validator set that participates in the consensus protocol, how to update changes in the validator set and how to punish byzantine validators from evidence Tendermint reports to the application. Tendermint is mostly used in blockchains that define at application level a staking mechanism to generate a validator and a slashing mechanism to punish malicious validators.

2.6. Blockchain evolution

Blockchain technology phases are often defined as Blockchain 1.0, Blockchain 2.0 and Blockchain 3.0. The first blockchain created was *Bitcoin* back in 2008, which was created with the unique goal in mind to find a secure way to exchange digital cash. The blockchain layers (defined in section 2.4) were fused in a single entity and for this reason it is defined as "monolithic architecture". Variants of Bitcoin have been made with similar intents and structures (for example Bitcoin Cash, Litecoin, and Bitcoin SV). All these blockchains were based on PoW. This phase is defined Blockchain 1.0.

Blockchain technology phase shift came with *Ethereum* in 2015 [5], which introduced the smart contract concept. Ethereum application layer is a virtual machine (VM) capable of executing softwares as a Touring-complete machine, but it is also a replication state machine so that each node is the same VM replicating the same softwares. These softwares are called *smart contracts* and can be thought as softwares installed in the blockchain that allows generalizing the application layer,building upon it new applications that users can request to use. These applications are called *decentralized applications*. This phase is defined Blockchain 2.0.

Blockchain 3.0 is the current phase, where blockchains point to greater scalability, interoperability between different blockchains, and advanced smart contract integration. Many blockchains in this phase come whith a specific intent and are independent one from another, such as in *Cosmos* ecosystem [21]. This allows updating blockchain specific functionalities in a more agile way, without having to reach consensus over a single general-purpose platform as was before.

Blockchain technology is constantly improving towards more secure protocols. Its applications are mostly for decentralized finance, but its nature makes it fit also where high

security standards for privacy and data integrity are to be perceived, as for digital twins management in the aerospace field. This will be discussed in chapter 5.

3 | Case study: a damaged composite plate

The case study under exam consists of a composite plate with a central cutoff, instrumented with embedded fiber optics Bragg sensors, damaged and under tensile load. Damages reduce the strength properties of the plate and a reduction in the ultimate load is expected, correlated with the damage entity. A digital twin for the plate under exam has to be realized. The digital twin will provide information about the health state of the plate in terms of ultimate load and stiffness and will predict the stiffness for a configuration in which the load is incremented.

This case study is a conceptual study: plate under exam has not been built and the physical twin is represented by the finite element model of the plate.

3.1. Progressive failure analysis in Msc Nastran

MSC Nastran SOL400 allows performing nonlinear progressive failure analysis. In SOL400, MATF card allows the definition of the failure criterion to be used for composite materials, and the degradation model to be considered. ITYPE entry of MATF card is used to specify if no degradation, progressive degradation or immediate degradation has to be used. MATF entries allow the definition of the post-failure *residual stiffness fraction* (RSF) that will be applied to the failed element's stiffness properties. Indicating with k_i the stiffness of the undamaged element's ply, and with k_f the stiffness of the failed element's ply, the following link will be established by Nastran:

$$k_f = RSF \cdot k_i \tag{3.1}$$

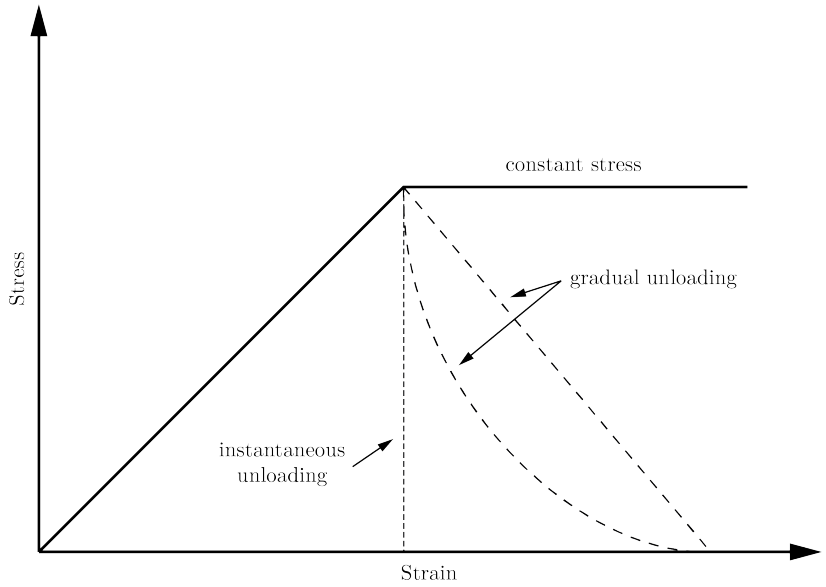


Figure 3.1: Degradation models for progressive failure analysis. From [36].

Nastran will compute also the Total Damage TD at any requested element and for each lamina of the element. TD is the complement of the Residual Stiffness Fraction of the element. In case of immediate degradation:

$$TD = \begin{cases} 0 & \text{if element did not fail} \\ 1 - RSF & \text{if element failed} \end{cases} \tag{3.2}$$

A progressive representation of the damages inside the composite can be obtained by visualizing the maximum TD of an element. This will show if failures inside elements' laminae have been reached.

Failure indexes are computed by Nastran based on the failure criteria selected. A failure index will move from zero to one, where one means that failure has been reached. For composite materials, failure indexes are computed for each ply, giving insights on how far the different laminae are from failure.

3.2. Problem description

The plate of this case study has a length of 1200 mm and width of 600 mm. The plate has an internal rectangular cut-off, centred in the plate center, of 450 mm length and 150 mm width. Plate geometry and laminate coordinate system are represented in figure 3.2.

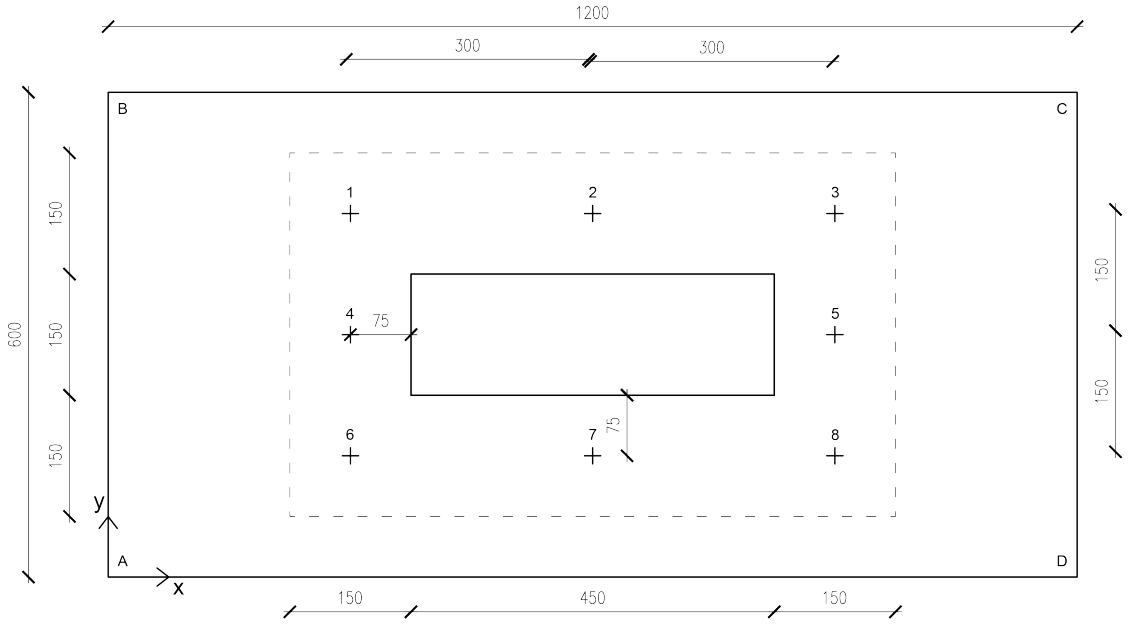


Figure 3.2: Geometry of the composite plate. +: FBG sensors. Units in mm.

The plate is made of composite material obtained from lamination of unidirectional graphite-epoxy plies and it is instrumented with 8 embedded fiber Bragg grating (FBG) sensors disposed around the central cutoff as depicted in figure 3.2. These sensors will measure the local strains in the x direction of the global coordinate system, which is aligned with the longitudinal direction of the plate.

Composite material choice has been made based upon an experimental study found in literature that has been used to validate the preliminary analysis results of the numerical simulations performed with MSC Nastran. Composite material used is T300/1034-C, which properties are defined in table 3.1.

Property	Value
Longitudinal Young's Modulus E_{xx}	146858 [MPa]
Transverse Young's Modulus E_{yy}	11376 [MPa]
In-Plane Shear Modulus G_{xy}	6185 [MPa]
Out-of-Plane Shear Modulus G_{yz}	6185 [MPa]
Out-of-Plane Shear Modulus G_{xz}	6185 [MPa]
Poisson's Ratio ν_{xy}	0.3
Ultimate longitudinal tensile strength X_t	1730.5 [MPa]
Ultimate longitudinal compressive strength X_c	1379 [MPa]
Ultimate transversal tensile strength Y_t	66.5 [MPa]
Ultimate transversal compressive strength Y_c	268 [MPa]
Ultimate shear in-plane strength S	133.7 [MPa]

Table 3.1: T300/1034-C material properties.

Lamination sequence is $[0/(\pm 45)_3/90_3]_s$, for a total of 20 plies, and each ply has a thickness of 0.1308 mm, resulting in a laminate of thickness 2.616 mm.

Plate is clamped on the left edge (AB). On the right edge (CD) the plate is free only to translate in the global x direction. Plate will be loaded in traction along the longitudinal direction resulting in an in-plane state of stress. Load is applied as imposed displacement on the right edge. Considering an undamaged condition in which the plate is in perfect state, without any manufacturing defects or damages, it is assumed that the plate may be loaded while in a damaged state resulting from manufacturing defects or small accidental damages. The maximum damage entity is assumed to be that which does not reduce the ultimate load of the plate under 70% of the undamaged ultimate load. Damages are assumed to be located only in the region defined by the dashed line in figure 3.2. Sensors are distributed evenly around the cutoff, inside the possible damaged region, providing insights into the plate's structural health condition. The ultimate load of the plate is 268.8 kN (it has been determined by numerical analyses on the undamaged model of the plate).

The composite plate is intended to operate in a loaded condition in the range of 100 kN to 150 kN.

3.3. Finite element model

The numerical model realized is a finite element model and analyses are performed using MSC Nastran. The finite element model of the plate has been realized with Femap. Plate has been modelled with shell elements of type CQUAD4 due to the nature of plane state of stress. Mesh elements are square elements of 10 mm edge. The resulting model consists of 6588 nodes and 6346 elements.

Each sensor is represented by the element in which the sensor is located, for a total of 8 elements representing each a different sensor. Strains measured from these are obtained from the analysis strains outputs of these elements. Boundary constraints are imposed as single point constraints with SPC1 cards.

Material property definition in Nastran is obtained with cards PCOMP and MAT8. MAT8 card is used to define the material properties at ply level. PCOMP card is used to define the lamination sequence to build up the material property that will be passed to each CQUAD4 element.

The degradation model considered is immediate degradation. Material failure is considered using the MATF card. Failure criterion used is a max strain criterion, with max strains derived from the material max strengths reported in table 3.1.

3.4. Preliminary study

Results of Nastran analysis have been validated by recreating a literature case example [7] that describes experimental tensile tests on a specimen with a central hole. In this test the specimen is loaded until complete failure and the force-displacement curve is obtained. This test has been numerically reproduced also by [41] and a similar test by [36]. Figure 3.3 provides a visual representation of the test. Specimen material is the same reported in table 3.1. The specimen has 203.2 mm length, 25.4 mm width, and the central hole is 6.35 mm in diameter. It is clamped at one edge and left free only to translate in the longitudinal direction at the other edge.

A numerical model is realized with shell CQUAD4 elements using MAT8 and PCOMP cards to define the lamination sequence and MATF to define the degradation model. Mesh of this model consists of 2638 nodes and 2480 elements. Figure 3.4 provides a detail of the hole's mesh.

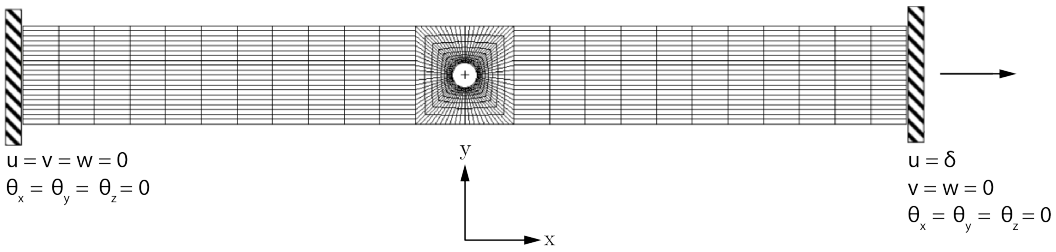


Figure 3.3: Visual representation of tensile test on specimen described in [7]. Adapted from **??**.

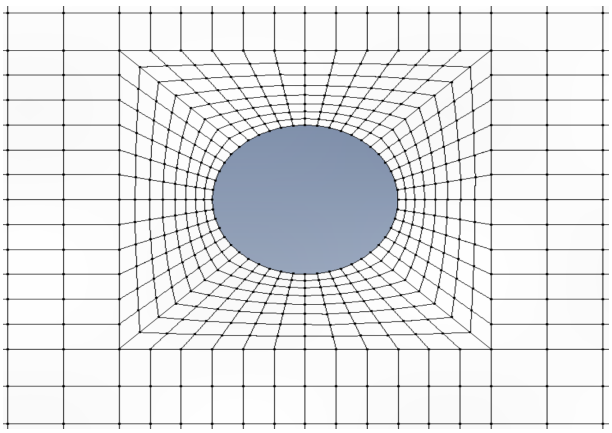


Figure 3.4: Detail of hole's mesh in the finite element model used in preliminary analysis tests.

3.4.1. Preliminary numerical analysis

Analyses have been performed using MSC Nastran with SOL400. The specimen is clamped at the left edge. Right-edge nodes have been connected to an external node through rigid RBE2 links. Displacement has been imposed on this external node. Constraint force and total displacement are requested as outputs on this external node.

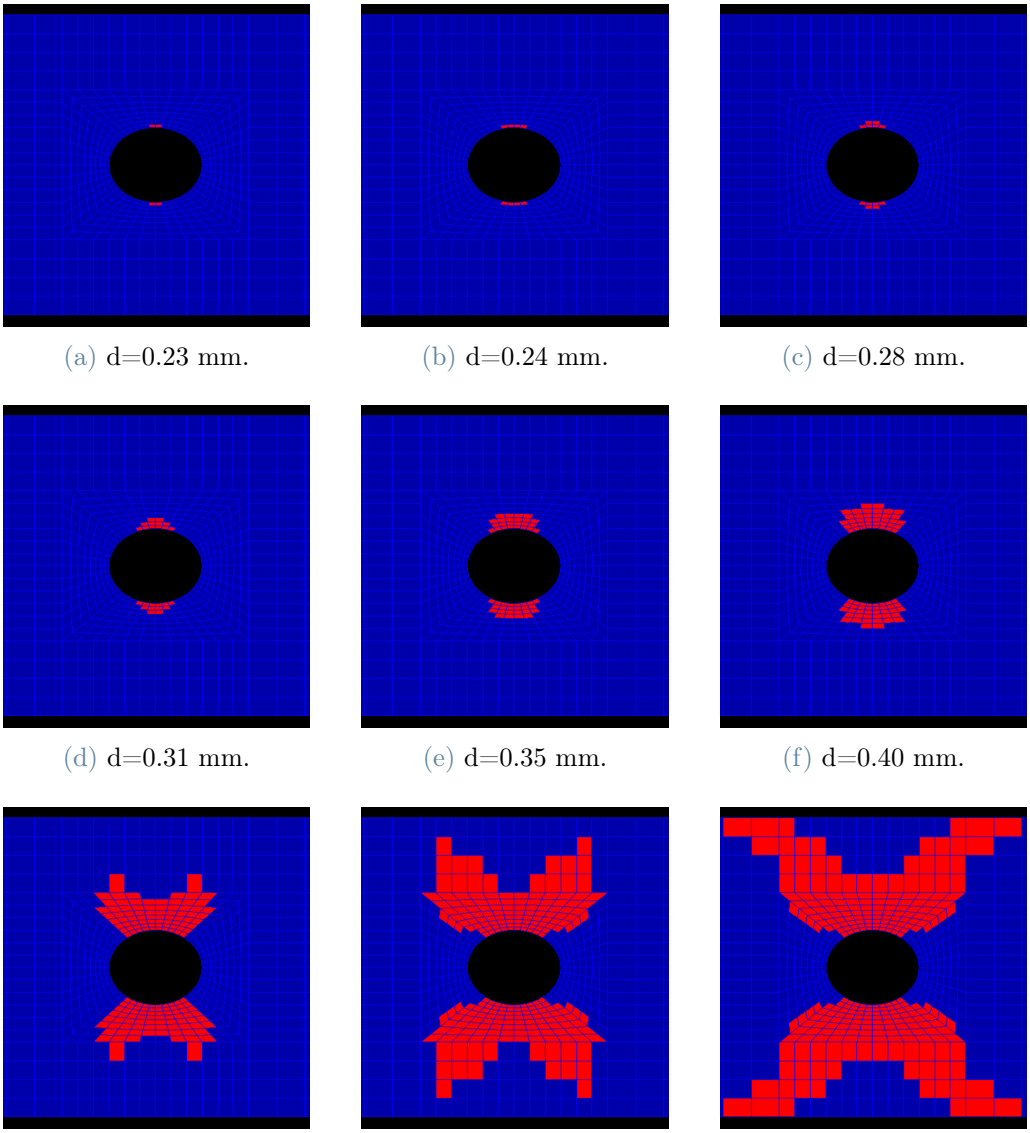
A constant step in displacement has been considered, incrementing the displacement up to 2 mm with 60 loading steps. Analyses have been performed using the modified Newton-Raphson algorithm implemented in MSC Nastran. This algorithm is based on the full Newton-Raphson method but does not update the stiffness matrix at each iteration. In the solution used, the stiffness matrix is updated every 5 iterations of the method, which is half of the default Msc Nastran implementation. Convergence criteria used are on the error in computed load and on total work, with thresholds set to 0.01, which is an order of magnitude lower than the default implementation in MSC Nastran [35].

Failure criterion used is the maximum strain criterion and RSF has been varied to find an RSF that produces a load-displacement curve similar to the experimental case. RSFs

considered are 0.01, 0.02, and 0.05.

3.4.2. Results

3.6 shows the results obtained. The ultimate load found in [7, 41] for this problem is 15700 N. With RSF 0.05 the resulting curve fails to capture the ultimate failure point, so it is discarded. For RSF 0.01 and 0.02 the complete failure is clear. Figure 3.5 shows the progression of the failure in the specimen. These are elements in which at least one ply failed. These are the 90° plies that are the first subjected to failure for this load condition, but 0° plies remain intact and will continue to carry the load. First ply failure can be detected happening at 0.24 mm. Complete failure happens when also the two external 0° plies fail.



(a) d=0.23 mm. (b) d=0.24 mm. (c) d=0.28 mm.

(d) d=0.31 mm. (e) d=0.35 mm. (f) d=0.40 mm.

(g) d=0.45 mm. (h) d=0.49 mm. (i) d=0.50 mm.

Figure 3.5: Progression of failed elements at different displacements d. Red: failed elements. Blue: intact elements.

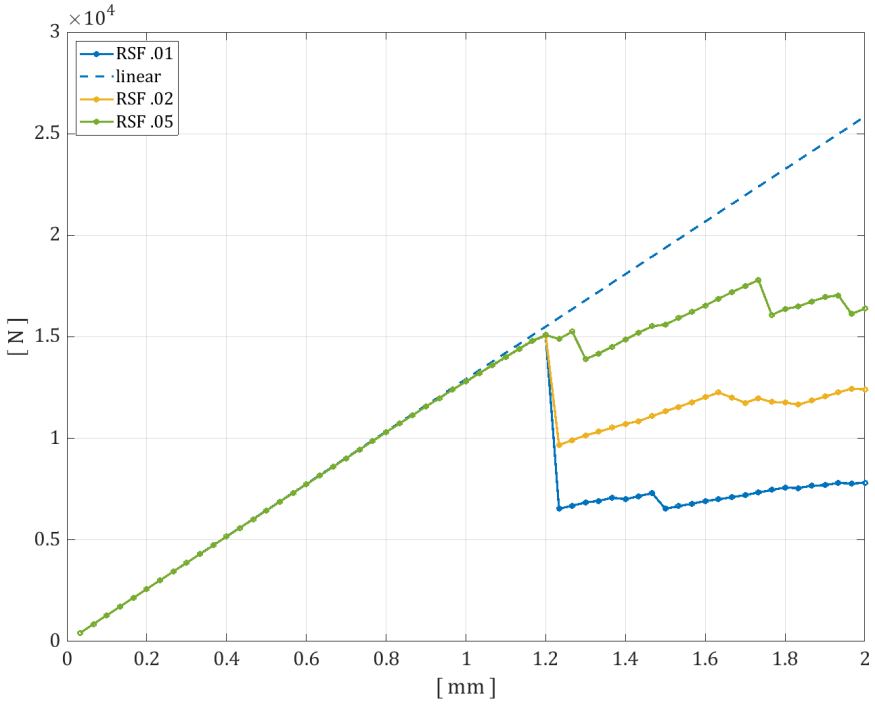


Figure 3.6: Load-displacement curve of the preliminary analyses on the specimen.

The ultimate load is 15'083 N and the ultimate displacement is 1.2 mm. Checking the analysis results for the 0° plies, the total damage of these plies shows that the 1.2 mm is the last step in which these have not failed yet, thus confirming that the 1.2 mm is the ultimate displacement for the specimen.

Results show agreement with the experimental results of [7] thus the preliminary phase is considered concluded and the RSF value of 0.01 will be used in the case-study plate model.

3.5. Numerical analysis on the undamaged plate

Analyses have been performed using MSC Nastran SOL400. Analyses have been performed in the same way as done for the preliminary analyses on the specimen, described in section 3.4.1. Here a constant displacement increment has been considered, incrementing the displacement by 0.01 mm in each load step up to 7 mm displacement for a total of 70 steps. RSF considered is the one obtained from the preliminary analysis, which is 0.01.

3.5.1. Results

Figure 3.7 shows the load-displacement curve obtained from the undamaged plate. Ultimate load is 268'832 N and ultimate displacement is 6.3 mm. The dashed line in figure 3.7 pictures the linearized behaviour of the undamaged plate. Linearized load at failure point is 280 kN. The reduction in stiffness is 4% with respect to the linear case. Also for this failure point it has been verified from the results that the 0° plies failed in the step after the ultimate load point.

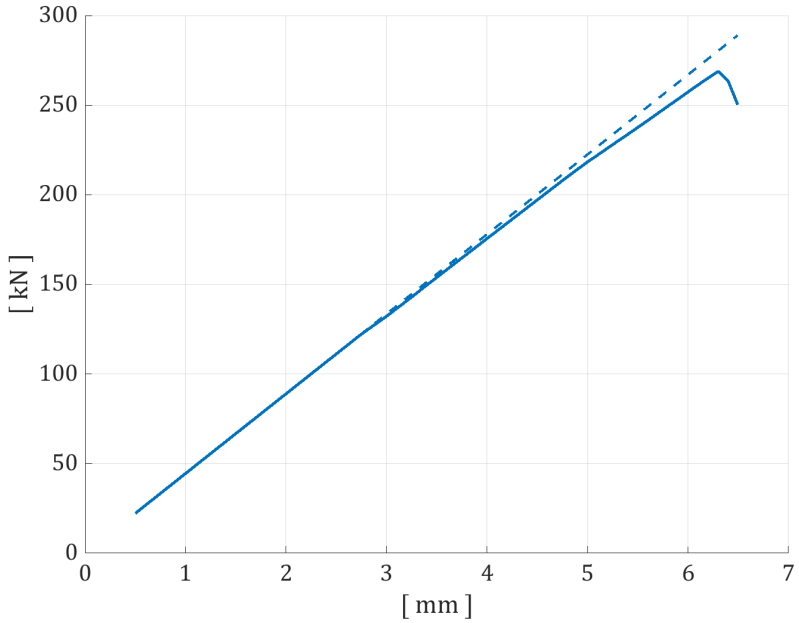


Figure 3.7: Load-displacement curve of the undamaged plate. Dashed: linear case.

Figure 3.8 shows the sensors measured strains in the global x direction (aligned with the tensile load). It can be noted that these overlap, resulting in three curves. This is due to the symmetric state of stress generated in the plate, since no damaged part is considered by now. The higher deformations are measured by sensors 2 and 7, which are the central ones. Sensors 4 and 5 measure the smallest deformations. Sensors 1, 3, 6, and 8 are those located at the corners and measure deformations which are slightly less than those of the central area. It can be noted from sensors 4 and 5 that a sharp increase in strains is registered. This is due to ply failure in that region.

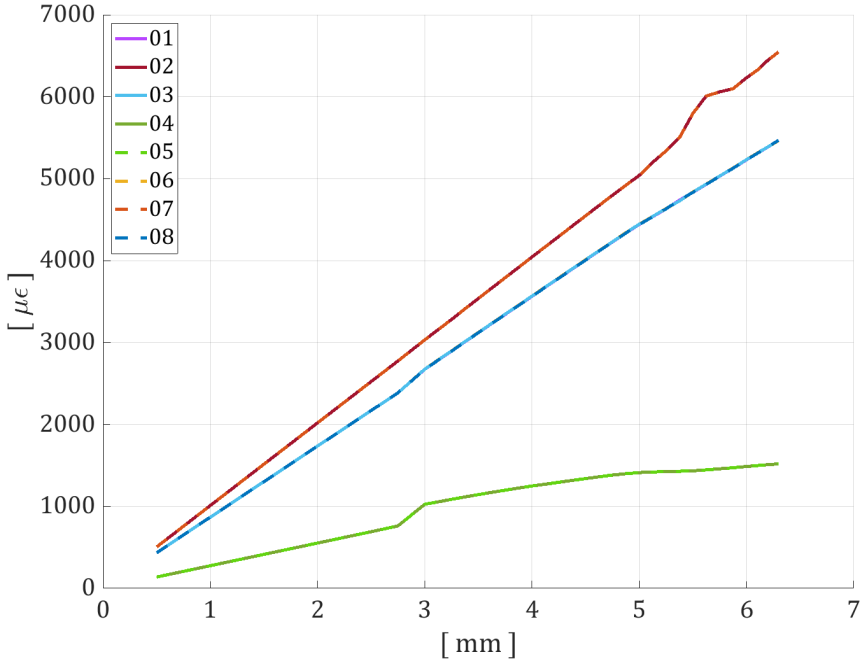


Figure 3.8: Strains measured by the embedded sensors in the global x direction in the undamaged plate analysis.

4 | Digital twin model generation

The digital twin model realized is a neural-network model built using PyTorch library [38]. Digital twin model has as input parameters the strains measured from the FBG sensors and produces as output: (i) the prediction of the ultimate tensile load; (ii) the ratio between the plate stiffness and the undamaged linear model stiffness; (iii) a prediction of the same ratio of point (ii) that will be observed if the applied load is increased of 20%.

4.1. Database generation

To generate the digital twin model, first a database resulting from different analyses on the composite plate has been generated. Database has been created from many progressive failure analyses of the composite plate finite element model in different damaged configurations. The finite element model approach follows what has been done by Kapteyn and Willcox in [25] and [24], where a digital twin model for a UAV is presented, constructed upon a model library obtained by varying the stiffness in selected regions to simulate the outcome of some damage. The composite plate region subjected to damages has been divided into twelve patches as depicted in figure 4.1. Each patch is represented in the finite element model as a different region with different material properties. To simulate a degradation, the patches' stiffness properties have been scaled down by a knock-down factor. To obtain the dataset the following has been done: (i): a knock-down factor is chosen; (ii) the knock-down factor is applied to a single patch material properties to simulate the degradation; (iii) the resulting model is analyzed with Nastran; (iv) output set values are obtained from the analysis results; (v) a new knock-down factor is chosen and steps repeated. Each patch has been analyzed for a knock-down factor varying from 10% to 60% with a 2% step. Resulting analyses have been post-processed to eliminate analyses failed due to convergence problems or violating the assumptions made in section 3.2.

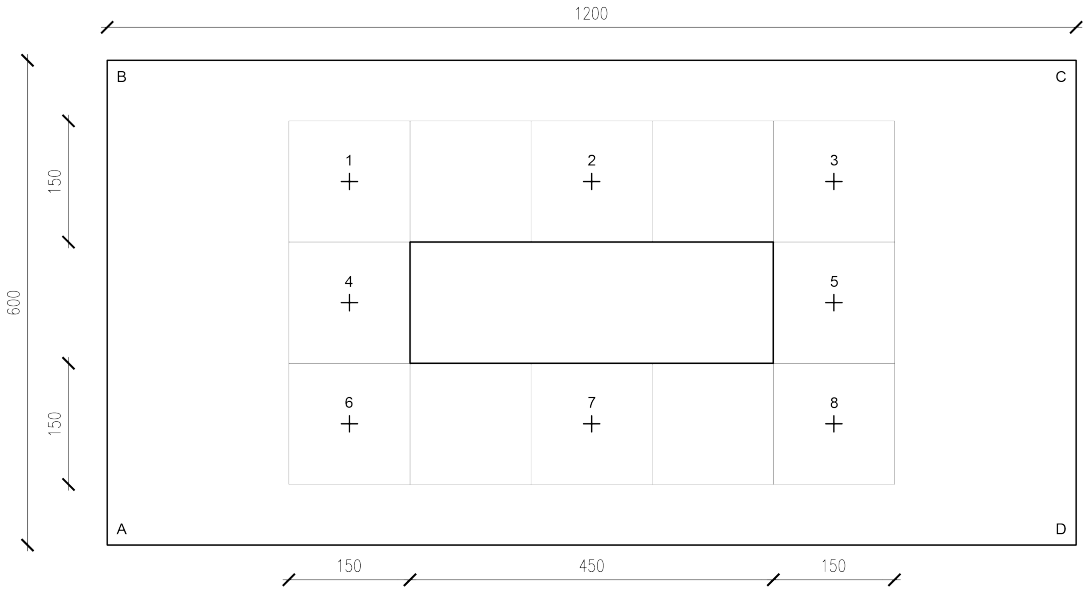


Figure 4.1: Patches division of the plate region exposed to damages.

4.1.1. Numerical analysis for the damaged models

Numerical analyses done follows what was done for the undamaged case in section 3.5. Here a multistep approach has been used for the displacement step increments. This has been done to reduce computational time. Load increment steps considered are finer going toward the ultimate failure region, where failure is expected. For each step, step parameters are managed with NLSTEP entry. Five analysis steps have been considered, with 5, 4, 20, 32 and 10 load increments each. Displacement increments that have been imposed in the different analysis steps (a.s.) are: (i) 0.3 mm in a.s. 1; (ii) 0.125 mm in a.s. 2; (iii) 0.1 mm in a.s. 3; (iv) 0.0625 mm in a.s. 4; (v) 0.05 mm in a.s. 5.

Many analyses terminated without reaching convergence. Results of these have been analyzed to detect if the last converged result was in a situation of entire or near-entire failure, or far from it. Failure or near-failure situations have been detected by analyzing the resulting total damages and failure indexes on sensor elements. External plies, which are lamina 1 and lamina 20, are the only two oriented at 0 degrees, thus fully aligned with the tension direction and then the most load-carrying ones. What was found in the analysis is that analysis failed for two reasons: (i) convergence was not reached and failure indexes on sensor elements were far from failure; (ii) convergence was not reached but failure indexes showed that the plate elements completely failed, or the failure was

total in internal plies while external plies were very near to fail with a failure index above .95. Results from the first situation have been discarded, while results from the second have been considered valid and the last converged values where considered as the ultimate load and displacement for that damage state.

Figure 4.2 reports the envelope of all the analysis for which failure has been possible to be asserted, both from converged or not converged analysis. Unloading part after failure has been post-processed to give a better representation. Envelope is obtained by filtering out analysis that did not reach the failure point and analysis that did result in a reduction of more than 30% of the ultimate load (thus violating the assumption on the maximum damage entity stated in section 3.2). Figure 4.3 shows the strains measured for a damaged condition defined by a stiffness reduction of 20% on the patch of sensor 1. Strain curves' complete overlap is no more present since the damaged patch alters the stress distribution with respect to the undamaged case.

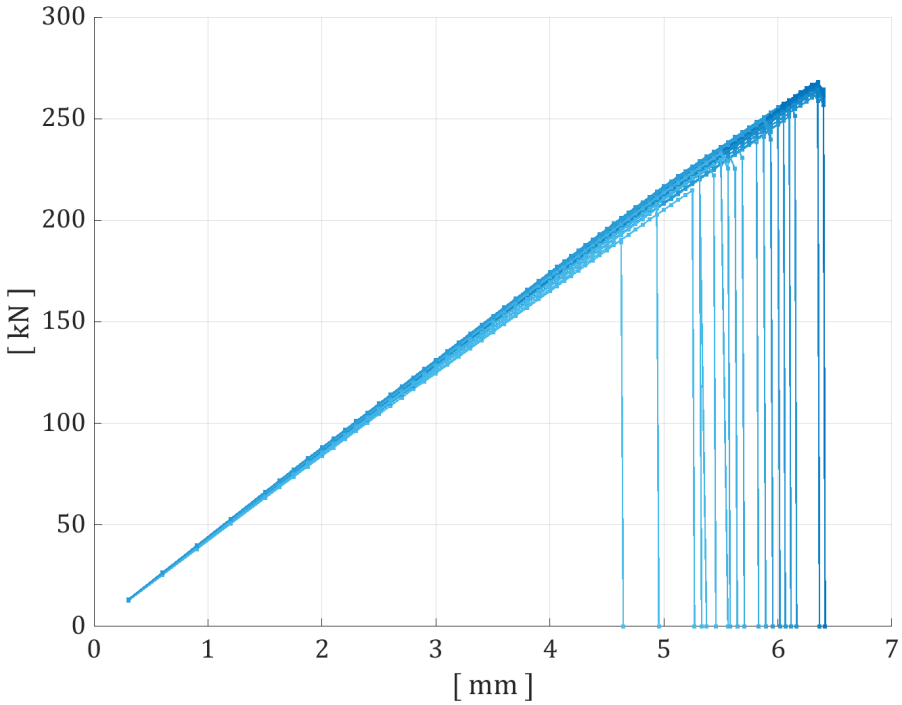


Figure 4.2: Load-displacement envelope for the damaged plate.

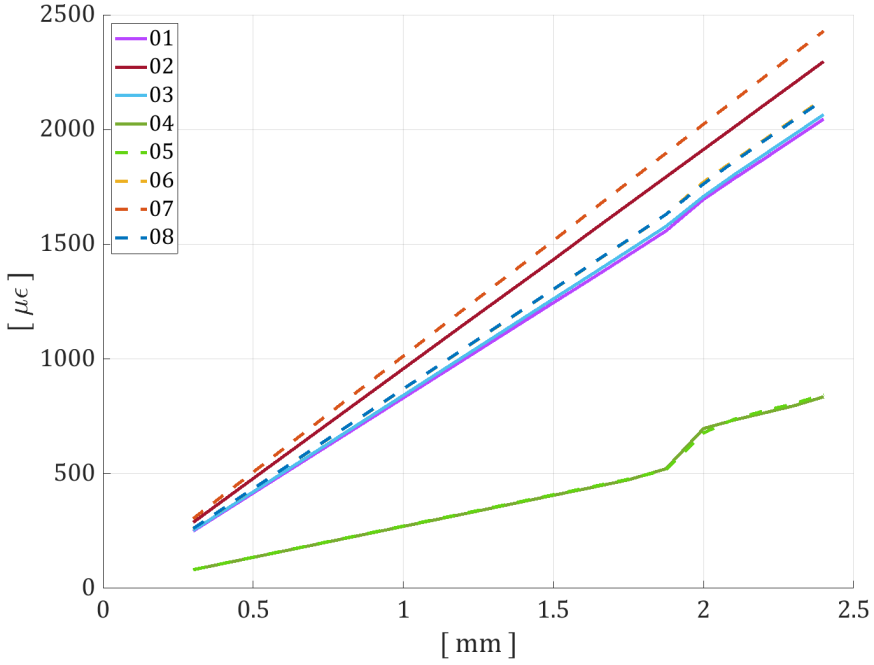


Figure 4.3: Strains measured by the embedded sensors in the global x direction in the damaged plate analysis (stiffness reduction of 20% on patch of sensor 1).

4.2. Feedforward neural network model

Neural networks are models inspired by the functioning of the biological brain. A feedforward neural network, also called Multi-Layers Perceptron (MLP), is a neural network composed of interconnected nodes, called *neurons*, organized in layers. A great description of these models is given in Pitton [37], from which many of the notions here reported derive, and the codebase he developed for his Master of Science thesis has been used as starting point for the neural network model design.

A neuron is an element that receives in input some values and produces an output value based on the weighted sum of the inputs it receives and a non-linear function. This linear function, called *activation function*, maps the weighted sum of the inputs plus a constant value for the node, called bias, to an output value.

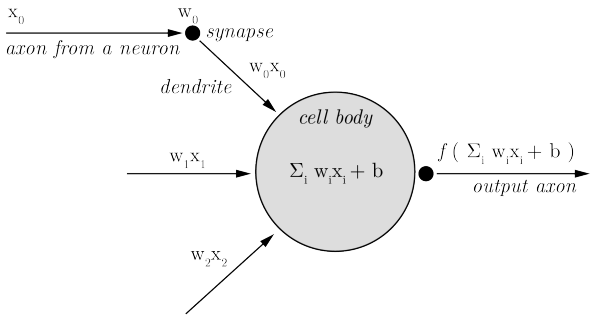


Figure 4.4: Visual representation of a neural network neuron.
From https://cs231n.github.io/neural-networks-1/ .

Adjacent layers of neurons are connected one to another, each node of a layer being fully connected to all the nodes of adjacent layers by different weighted connections. Inputs received from the first layer in the model will be processed and transmitted to the following layers, eventually resulting in output values that will be the prediction of the neural network based on the input received. From this behaviour the name "feedforward neural network" is derived.

Training is how the neural network learns to make good predictions. Training process is based on an optimization problem of the network parameters, which are the weights of the neuron connections and the biases of each neuron. The objective function to be optimized is called *cost function* and the most commonly used in regression problems is the Mean Squared Error (MSE). Defining as N the total number of inputs, $\hat{y}$ the correct output and y the predicted output, the MSE can be defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{4.1}$$

In network training, input data, called training data, are given to the neural network model, and the output y_i of each input data is obtained. In supervised learning the training algorithm in the training phase knows what the real outputs $\hat{y}_i$ are, and thus the MSE can be computed. The cost function has not a global minimum because it is possible to exchange hidden layers or also neurons inside a layer without changing the value of the cost, and many plateau regions generally exist in the cost function space. The optimization problem is solved using a technique called backpropagation, which propagates the error backwards to allow gradient computation and updating of the network parameters. Parameters updating in backpropagation can be performed using Gradient Descent (GD) or second-order methods to obtain the steepest direction in the decrease of the loss func-

tion. Using GD the entire dataset is passed to the network in the feedforward process and then cost function and sensitivities are computed. Defining the cost function as $\mathcal{L}(\theta)$, function of the network parameters θ, these parameters are updated from their actual value at time t to the new value at time $t+1$ with the following relation:

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial(L)}{\partial \theta} \tag{4.2}$$

α is called *learning rate* and rules how fast the optimization process is conducted. Going fast may result in skipping from a not-good local optimum to another, while going slow may result in getting stuck in a not-good local optimum, thus the learning rate must be chosen with care and tests should be done to find the best one for the network configuration. Hyperparameters are parameters that affect the resulting network model. The learning rate is one hyperparameter, along with the number of neurons in layers and the number of layers.

4.2.1. Optimization algorithm

From Gradient Descent many optimization algorithm variants have been created. The algorithm used for the neural network of the digital twin is Adam, which derives from the union of SGD with momentum and adaptive gradient.

Stochastic Gradient Descent (SGD) is a variant of the Gradient Descent in which computation time is greatly reduced since it does not use all the dataset to compute the gradient but uses an estimate computed from a randomly selected subset of the data at each iteration. This approach introduces noise in the parameters updating, which can help the optimization to escape local minima and improve generalization. For this reasons it is widely used in neural networks training.

SGD with momentum is a variant of SGD. In this technique it is included a term that gives to the "updating direction" an inertial property to make the updating process more robust against oscillations in parameters changhes. This term takes into account the previous update's direction and magnitude and influences the current update. Its formulation is the following:

$$m_{t+1} = \gamma m_t + \alpha \frac{\partial(L)}{\partial \theta} \tag{4.3}$$

$$\theta_{t+1} = \theta_t - m_{t+1} \tag{4.4}$$

m is called *momentum term*. *gamma* determines the influence of the previous direction on the current update and it is generally set between 0 and 1, where higher values give more weight to the past direction. Momentum term in this algorithm is said to be accumulated because it results in an exponentially weighted sum of the past gradients. The resulting updating step will be proportional to the accumulated momentum.

Adam (Adaptive Moment Estimation) is one of the most used optimizers and considers both a momentum part from an exponential moving average of the gradient and a moving average of squared gradients for each parameter [3]. It demonstrated great training performances in the training of deep neural networks. Defining g_t as the gradient at iteration t, β_1 and β_2 as decaying weights, m_t and v_t as the estimate of the first moment and the estimate of the second moment of the gradients:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.5}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.6}$$

A bias correction is then introduced as follows:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad ; \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4.7}$$

The parameters updating then results from the following:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4.8}$$

Adam optimizer is the one used in the digital twin model generation.

4.2.2. Dataset pre-processing

An important aspect of neural network training is the definition of three sets to be used in the procedure. Dataset must be split into training set, validation set, and testing set to cover each aspect of the training and validation phases.

Training set contains the data on which the training is based. From this data, the error for the cost function will be computed and backpropagated in the updating of the parameters.

Validation set contains data used in the training phase, but not used in backpropagation. They are treated as "unseen" data at each iteration and depending on the results obtained

from this set it is possible to understand if the network training is improving the model quality or not, meaning it is just overfitting the training data. Overfitting means that the model performs exceptionally well on the training data but fails to provide good predictions on new, unseen data. With the validation set, one can see during the training process if the results coming from this set improves or not, and then stop the training if needed and change the network parameters or hyperparameters and try a new training.

Testing set is used once training is complete to obtain results on unseen data and validate the training results.

Data must be also normalized to remove the different orders of magnitude of the different inputs and outputs. This will improve convergence. Data can then be rescaled back to obtain the real values.

4.2.3. Model generation

Since the case study plate's standard operation range varies from 100 kN to 150 kN, the output quantities included in the dataset are obtained for input loads in this region, extended also by 20 kN to add an extra margin to the digital twin model knowledge. The resulting dataset is composed of 400 design points and has been split into training, validation and testing sets considering 60% for training, 20% for validation and 20% for testing.

Cost function used for the optimization problem is the MSE and optimization is performed using Adam with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e-8$ set as found in the original Adam implementation [26]. Activation functions considered are rectified linear units (ReLU). Weights have been initialized with Xavier initialization.

4.2.4. Accuracy metrics

Metrics used to analyze the model quality are here presented.

In the following relations, y_i represents the correct output related to sample i, $\hat{y}_i$ the predicted output related to sample i, and $\bar{y}$ the mean of all the correct outputs.

R square error (R^2), called also coefficient of determination, defines the overall accuracy of the model and it is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{N}(\bar{y} - y_i)^2} = 1 - \frac{MSE}{variance} \tag{4.9}$$

The more R^2 is close to 1 the more the prediction errors the model produces are small compared to the total variation of the dependent variables (the correct outputs), resulting in a good capability of the model to predict in general correct results.

Mean Absolute Percentage Error (MAPE) represents the average absolute error between the predicted outputs and the correct ones, obtained as the absolute of these quantities' difference. A small MAPE means that errors are small in average, thus the smaller the MAPE the more accurate the model predictions are.

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}|\frac{\hat{y}_i - y_i}{y_i}| \cdot 100 \tag{4.10}$$

Relative Maximum Absolute Error (RMAE) is a local indicator of the accuracy which indicates if the model fails to predict accurate values for some specific output.

$$RMAE = \frac{max_i(|\hat{y}_i - y_i|)}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\bar{y} - y_i)^2}} \tag{4.11}$$

4.2.5. Results

Some tests have been done to obtain the best model configuration. The number of hidden layers has been varied from two to seven. Neurons per layer have been varied from 20 to 200. Best results are obtained from 3 hidden layers and 30 neurons per layer with a learning rate of 0.0001. Table 4.1 resumes the best results obtained and figure 4.5 the MSE loss evolution during the training process.

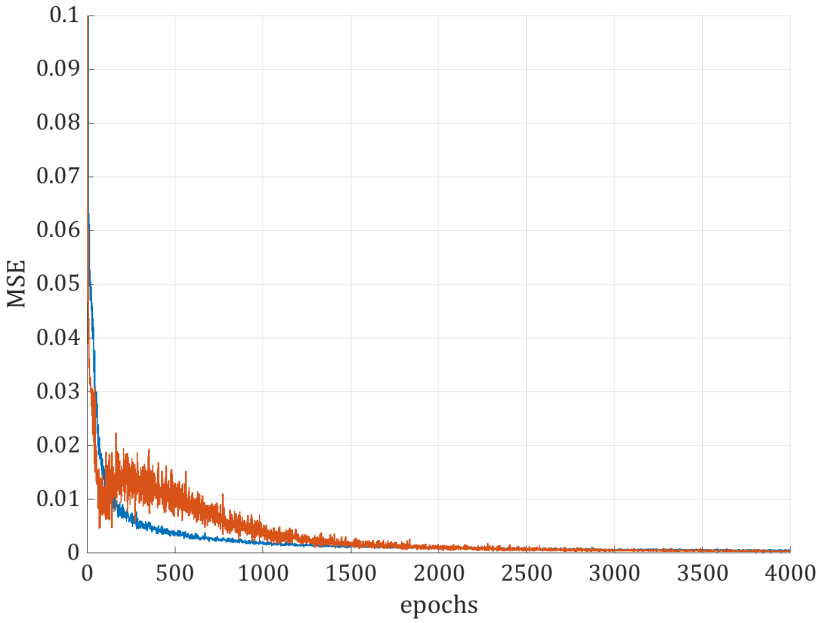


Figure 4.5: MSE of training set (blue) and validation set (red) during the training process.

Metric	Train	Test
R2	0.991	0.846
MAPE	0.002	0.008
RMAE	0.427	0.891

Table 4.1: Performance metrics of the best neural network model generated.

For the testing set, R2 is high and MAPE is very low, meaning the model captures very well the dataset relations. RMAE is high, meaning that there are points in the dataset that the network fails to correctly predict.

Analyzing the results, it has been noted that the accuracy of the model is very good for the stiffness prediction and performs in general very well for the ultimate load prediction, but some points of the testing set lead to an error in the range of 10% to 12% on the ultimate load. These points are a small number compared to the whole testing set (12 out of 88). For the stiffness predictions, the network shows very high accuracy. Errors on these are close to zero, where max errors are close to 1% in very few points. The resulting overall performance is thus very good, but the spikes in the life prediction error should be addressed for the digital twin model to be the best reliable as possible. The dataset should be refined, adding more analysis and trying to consider smaller patches. Analysis used to generate the database could be differentiated in load increments so that more

values of failure points can be taken into account. Also, the sensors' placement should be investigated to find the optimal position. These are beyond this case study purpose, so these refinements have not been done in this work.

5 | Digital twin life management and distributed training on blockchain

Digital twins are powerful instruments but safety issues have to be taken into account. What will happen if someone performs malicious actions corrupting the digital twin? The digital twin will produce wrong predictions and operators will probably make wrong choices based on that. What if the digital twin has a structural health monitoring purpose and the ability to limit the physical twin when the diagnosis it produces says that the physical twin's integrity is compromised? The tampered digital twin would limit the operations of the physical twin, or completely stop it. This could escalate into catastrophic disasters if the physical twin is an aircraft engine or surface control system or any critical component of a complex system. These issues could be limited by tracking any digital twin update and by checking before taking automatic actions if the digital twin is the correct one or if it has been corrupted.

In situations where a consortium of different corporations cooperate in the creation and updating of a complex digital twin, then a blockchain solution can enhance the DT life-cycle management's security. The different entities inside the consortium would be the blockchain validators and if sufficient decentralization is reached then the resulting blockchain network will securely store the updatings history. Also other information other than the updates history could be stored, and this will depend on the specific situation.

Blockchain technology can also be used as an orchestrator for distributed analysis. This chapter describes the development and implementation of a blockchain that implements a protocol for distributed training of neural-network-based digital twins.

5.1. Vesta

Vesta is a blockchain that the author developed to manage the digital twin life-cycle. It integrates also a distributed training protocol that will be presented in section 5.3. It is available at https://github.com/niilptr/vesta.

Vesta is named after the Vesta, the Roman goddess of the sacred fire. Role of Vestals was to maintain the sacred fire always lit, as it was an effigy of gods protection over Rome. As in the Vesta cult, the role of Vesta nodes is analogous to Vestals' duty: as Vestals had to maintain the sacred fire lit to preserve Gods' protection, Vesta nodes have to maintain the chain secure to protect the digital twin integrity.

Vesta has been built using the Interchain (formerly known as Cosmos [21]) developing stack. The application layer has been developed using the Cosmos-SDK framework [9], while consensus and networking layers are handled by Tendermint. The core functionalities of the application are creation, updating and deletion of the digital twin in the nodes' memories. The updating may be manual, which will be performed by an authorized operator, or automatic when distributed training is requested.

A digital twin is represented in Vesta as a data structure that records four fundamental properties that will be described in the following list.

1. The digital twin *name*. This may be trivial when only one digital twin has to be stored, but in a real application a digital twin may be composed itself of many other digital twins. Let's think about an aircraft: the aircraft digital twin will comprehend the power-plant digital twin that will comprehend the engine(s) digital twin that maybe will comprehend a rotor subgroup digital twin. Indexing all of these by a name provides an easy-to-use implementation.

2. The digital twin *creator*. This field will contain the address who first instantiated the digital twin and it will be used to grant or deny access to twin modification.

3. The digital twin *hash*. This is the identifier of the digital twin and the representation of the twin itself. No other information will be stored on-chain. For simple dummy twins this could be a limitation and maybe one would like to store all the twin files, but for real-use applications this would be impractical: each node would have to store terabytes of data, and storage and storage handling has a cost. This solution solves the problem of storage management leaving intact the fundamental property of data integrity. The full digital twin will be stored in remote server(s), and its hash will be saved on-chain. If any malicious manipulation of the twin happens, the

on-chain hash would differ from the remote twin hash and this will prevent using compromised twins.

4. The digital twin *last updater*. This field will contain the address who last modified the twin and will help tracking the history of the twin's modifications to detect any problems if any happens, and to take actions in response, mainly where authorized addresses arbitrarily compromise the digital twin.

5.2. Messages

Users will interact with the blockchain by sending messages to it. Messages that can be sent to manage the digital twin life-cycle are reported below.

1. `create-twin`: this message has to be used when a new twin has to be instantiated.
2. `update-twin`: this message will be used when the creator or any other authorized address has to update the twin hash. This will happen where some authorized modifications have been done to the twin stored in the remote server(s), and the new hash has to be stored on-chain.
3. `delete-twin`: This message will be sent by the creator or any other authorized address when the digital twin has been dismissed and there is then no means to continue storing it on-chain.
4. `train`: This message will be sent by the creator or any other authorized address when the digital twin needs to be retrained. During lifetime operation data will be constantly collected from the physical twin sensors and the database updated. This creates the possibility to retrain the digital twin model to obtain better accuracy of its predictions.

Along with these messages, other two "satellite" messages have been implemented to handle the distributed training. These will be explained in section 5.3.

5.3. Distributed training protocol

For *distributed training* it is here intended a neural network training scheme based on parallelizing different training jobs taken from a training population among different nodes of a blockchain.

Distributed training procedure will start from a transaction to the blockchain that will trigger the training process and the updating of the twin state.

In Vesta implementation the distributed training is based upon a group of authorized accounts, called *trainers* that will start a local training job on their machines and agree on the training best result. In a real-case scenario, each authorized account will be linked to a machine whose hardware architecture could be specifically designed for this job. This machine could be a node of the network or a machine with a secure communication channel established with the respective trainer node. This is a choice that will depend on the specific real implementation.

Handling distributed training in a blockchain is problematic due to the two different natures of the task involved. Neural network training is a non-deterministic process that depends on how the hyperparameters are initialized. Training results from the same training configurations (the choice of hyperparameters values) could be made deterministic when the randomization seed is fixed, but another problem remains: in distributed training, even if locally deterministic, results will be different from trainer to trainer since each trainer will perform a different training job from the whole training jobs population. Blockchain nodes will instead have to reach consensus among a new state, deterministically. Each trainer will have in memory its own version of the training result, and the state cannot be directly updated from different nodes' memories. Moreover, results must be validated by the blockchain protocol, otherwise it would simply be more convenient to use a central authority and get rid of the distributed ledger. Two problems arise:

1. A way to make the blockchain application informed about all local results must be implemented.
2. A way to validate and elect the best result is needed to reach consensus on what the next twin state must be.

In dealing with problem 1, each trainer has to inform other nodes about its results. Delivering results on-chain is not possible because only the training hash can be stored. The solution to this problem comes from the nature of the implemented twin management architecture, where a central server(s) is used as not-fully trusted side store.

The distributed training procedure will act in the way described below. This procedure is inspired by Tendermint algorithm [4].

1. When the training message is received the training phase is started.
2. In training phase, blockchain application triggers each trainer node to start the local analysis.
3. Local analysis program makes trainers who complete the training upload their results to the central server.

4. In training phase, blockchain application triggers the following behaviour on nodes authorized to read the central server:
   (a) they check periodically if all trainers have uploaded their results on the server;
   (b) in case they witness that all trainers completed their job they inform the blockchain sending a specific `training-phase-ended` message. This transaction is registered in the blockchain.
5. In training phase, if a sufficient number of nodes, greater than a predetermined threshold, agrees on the ending of the training phase, then training phase is ended and validation phase is started.
6. Eventually a predetermined timeout for the training phase is reached and validation phase is started.
7. In validation phase, blockchain triggers each authorized node to get the training results from the central server and select the best result following a predetermined strategy. If the result is not valid then the next best result is analyzed. Eventually, each node selects a result (or a `none` result) and informs the blockchain by sending a specific `best-result-is` message.
8. Eventually a predetermined timeout for validation phase is reached and validation phase is ended.
9. Blockchain application checks periodically for agreement on the best result. If agreement is reached (greater than a predetermined threshold) the twin state is updated with the new state. If no agreement is reached, or agreement on `none` result is reached, then the twin state is left unchanged.

5.3.1. Protocol implementation

To manage the distributed training phases a state variable representing the training state is saved in the blockchain store. This variable in the current implementation is called `training_state` and it is a structure that stores the following information.

1. *Training configuration file hash.* It will be used by the training software.
2. *Training state*: `true/false`. It will be set to `true` when the request is received and to `false` when training phase ended.
3. *Validation state*: `true/false`. It will be set to true when the training phase ends.
4. *Training starting time.* It will be used to deduct if training timout has been reached.

5. *Validation starting time.* It will be used to deduct if validation timout has been reached.
6. *Traing phase ended confirmation map.* This map will store the confirmations sent from trainers about the ending of the training phase. It will be used to check if agreement is reached on the training phase end.
7. *Best training result confirmation map.* This map will store the new model results hash confirmed by the different trainers. It will map each trainer to the new model hash they selected as the best valid training result. It will be used to check if agreement is reached on the new model hash.

An authorized user that wants to request the training will upload in the remote server the training configuration file. This file contains the dataset to be used, and for each authorized trainer (or a subset of them), the training hyperparameters that each trainer will set for its training, Then the user will send the `train` message to the blockchain. This message has the following fields:

1. `name`, the name of the digital twin to train;
2. `hash`, the training configuration file hash.

When a `train` message is received, the local training software is started on the trainers' nodes. This software will do the following.

1. Fetch the remote server for the training configuration file.
2. Verify that the hash of this file is the same specified in the `train` message, otherwise, training will be aborted.
3. Select from the training file the training configuration requested for the trainer.
4. Run the training.
5. Generate the results file when training is completed.
6. Upload the results file to the remote server.

The results file that is uploaded to the remote server contains the trainer identifier, the resulting model data (weights and biases of the different layers), the resulting values for the accuracy metrics, the hash of the model data and the order to be used in computing the hash from the model data (since the hash depends on the order in which model data are concatenated, this last information is necessary to properly recompute the hash). The model hash will be used in validation phase, where trainers will recompute the model

hash and see if it matches with the one found in the results file. If no match is found then the resulting model will be considered not valid.

In validation phase, the application makes the trainer fetch the remote server for all the trainers results files, compute a score for each resulting model found, and check if the best result is valid or not (if not valid, the next will be tried until no one is left). When results have to be validated, the application makes trainers call the training software that will validate the results by performing the following steps.

1. Compute the results' hash following the hash computation scheme provided by the results file.
2. Check if the computed hash matches the results' hash reported in the results file. If not, the results will be considered not valid.
3. Load the model data and run the model against the testing set, which is the same used in training phase, specified by the training configuration file.
4. Compute the accuracy metrics and check if these match the resulting values found in the results file. If not, the results will be considered not valid.
5. Return the response to the application about the validity of the results.

Eventually, the application will make the trainer send the confirmation message on what is the best result found.

To handle the different phases of the distributed training, at the beginning of each block a phase-specific logic is triggered by the Application Blockchain Interface (ABCI). This is the application part in charge of handling updates from the consensus engine and transmitting them to the application state-transition logic. Figure 2.2 gives a visual representation of this interface.

In training phase, at the beginning of each block, the application will make trainer nodes to check the remote server for training results uploaded by the trainers. If a trainer sees that all the trainers completed their task then the application will make it broadcast the confirmation message. At the beginning of each block the application checks if enough confirmations have been broadcasted, and in case it will start the validation phase. Trainers who already confirmed are made to remain silent by the application, avoiding sending double messages.

In validation phase, at the beginning of each block, the application makes trainers fetch the remote server to get the training results, pick the best one, validate it or discard it and analyze to the next one, and eventually send the confirmation message (which

includes the hash of the result). At the beginning of each block the application checks if enough trainers agree on the same result, and if agreement is reached then it will stop the validation phase and update the digital twin model with the new hash. Trainers who already confirmed their best-found result are made to remain silent by the application, avoiding sending double messages.

Figure 5.1 gives a representation of the application handling of the `train` message. Figure 5.2 gives a representation of what the application does at the beginning of each block in training phase. Figure 5.3 gives a representation of what the application does at the beginning of each block in validation phase.

The training software used for training the model and validating the results has to be the same among the authorized nodes. One could also develop a software that will be part of the blockchain application. It could be included at source code level or developed as smart contracts to be loaded on the blockchain. The easiest way is the one used in this implementation, where the training software is external from the blockchain application. The training software is triggered by the application, it reads the training configuration file from the remote server, runs the training and automatically uploads the results; in validation phase it reads from the remote servers the results files, selects the best result, validates it and broadcasts the confirmation from the trainer account.

The threshold implemented in the current state of the application for agreement on training ending and validation ending is 2/3 of the authorized trainers. The time for the training phase should be determined based on the complexity of the digital twin's neural network model. In the current implementation it is a pre-defined parameter, but it can be made an input of the training request the user sends to the blockchain.

Trainers, in the current implementation, own an access token that enables the connections with the remote server APIs. The results file they generate in the current implementation is not digitally signed by them. This could (should) be added in a real-case implementation of this protocol to enhance security, providing a way for the application to check the results file for its integrity. If not implemented, someone could for example modify the accuracy metrics score in the file, making the validation fail for this result.

Trainers are authorized addresses that may be different from validators of the chain. It is possible to set specific rules in the application code to allow the registration of validators accounts only. This can be useful to link the two entities and ease the punishment logic implementation of the two sets.

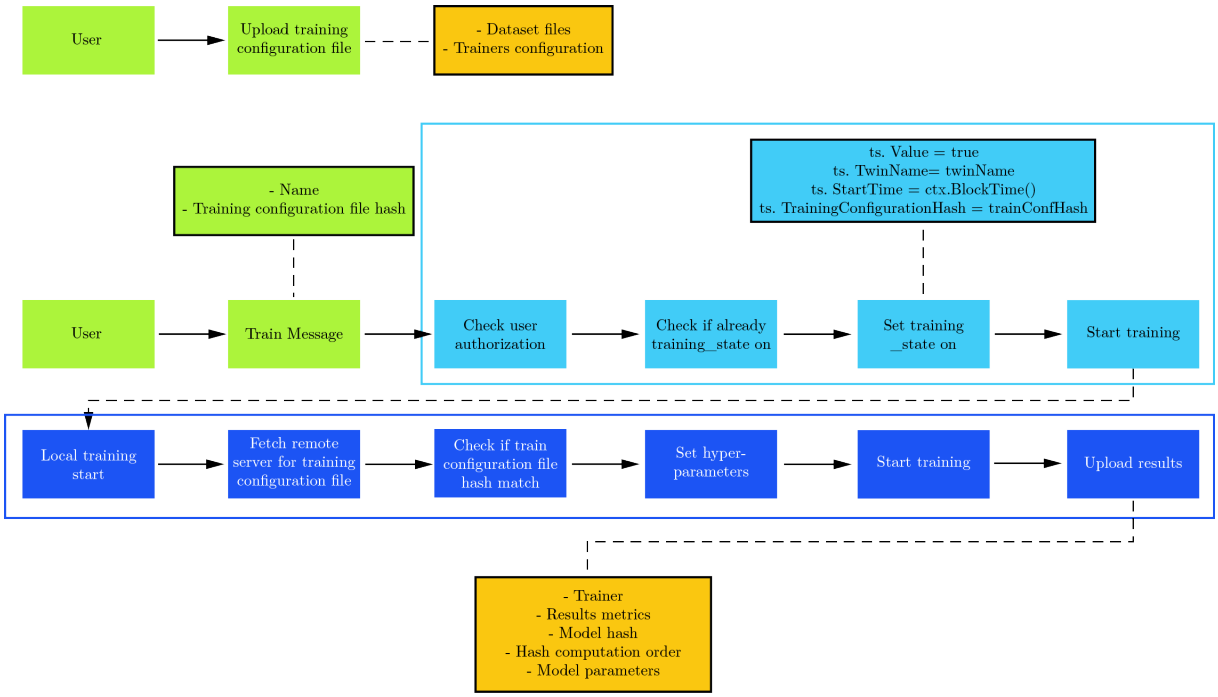


Figure 5.1: Representation of train message handling of the application. Green: user side. Light-blue: blockchain application side. Blue: local machine side. Orange: remote server side.

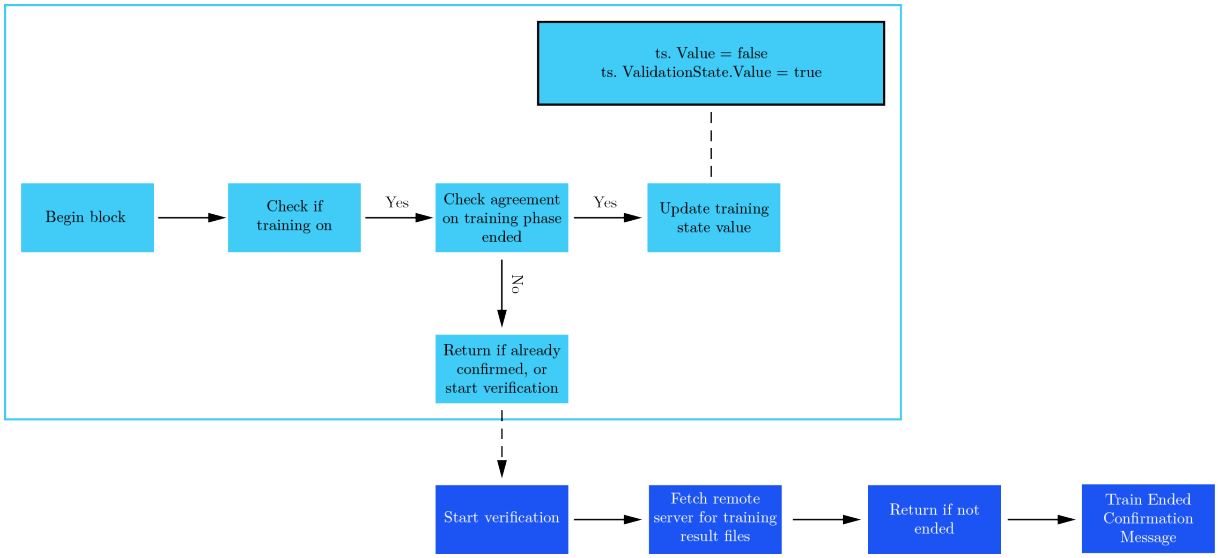


Figure 5.2: Representation of what is done by the application at the beginning of each block in training phase. Light-blue: blockchain application side. Blue: local machine side.

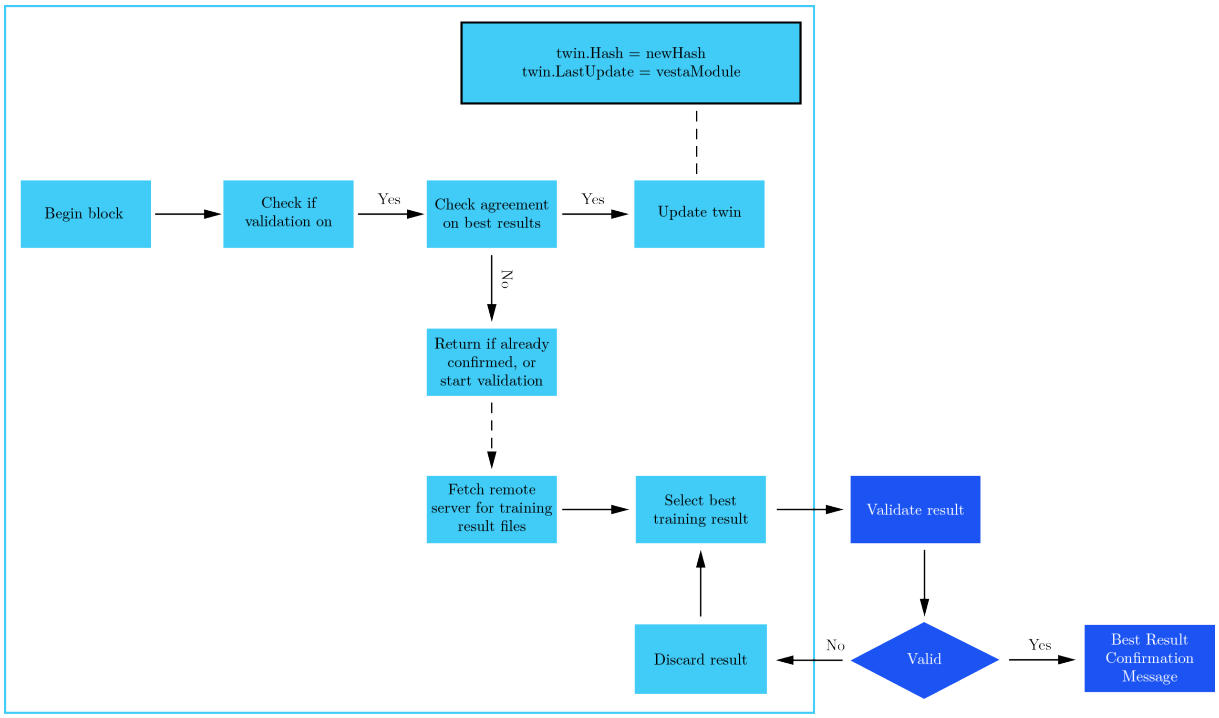


Figure 5.3: Representation of what is done by the application at the beginning of each block in validation phase. Light-blue: blockchain application side. Blue: local machine side.

5.3.2. Results

The protocol has been tested on a local network with 3 trainers. Tests have been conducted to test the application handling of byzantine failures.

1. Not allowed accounts were correctly detected by the application when training was requested including unauthorized accounts in the training configuration file.
2. The application correctly refused to start the training procedure when the training configuration file uploaded to the remote server and the one specified in `train` message were different.
3. A test where a results file was modified after its uploading has been done. Results file was altered by changing the network parameters' values. The application correctly detected the hash mismatch discarding the altered results.

Training tests have been performed requesting for each trainer, in the training configuration file, a different learning rate value to be used in the training process of the digital twin model developed in chapter 4. Values set as learning rate were 0.0001 for trainer one, 0.0002 for trainer two, and 0.0005 for trainer three. Block-time has been set to 6 seconds.

Training requested has been correctly handled: agreement was reached in the different phases, results were correctly uploaded to the remote private server and the twin hash was been correctly updated with the new one.

Twin is created from an authorized account who sends the message `create-twin twin00 2fd5316da2a85285e35e3e4785bf1fefab14c409a3d9c8222881dabb3fa366fb`, where the first input (twin00) is the name given to the twin and the second its current hash. Stored variables can be read by users with specific queries defined at application level. Figure 5.4 shows the query output for the `twin00` data.

```
nil@iceshard:~/chains/vesta$ vestad query twin show-twin twin00
twin:
  creator: vesta15xhpaax757zz76x8p96ku2gwuyvh4wz0d5h400
  hash: 2fd5316da2a85285e35e3e4785bf1fefab14c409a3d9c8222881dabb3fa366fb
  last_update: vesta15xhpaax757zz76x8p96ku2gwuyvh4wz0d5h400
  name: twin00
```

Figure 5.4: Resulting twin generated by the `create-twin` message.

Train is then requested by its creator who sends the message `train` with inputs `twin00`, the name of the twin to be trained, and the training configuration file hash. Figure 5.5 shows the logs' outputs of the three trainers when they acknowledged that all trainers have uploaded their results files, resulting in the `training-phase-ended` confirmation message broadcasting. Figure 5.6 shows the logs' outputs of the three trainers in validation phase, when they verified the results from the training phase and found the best valid result, resulting in the `best-result-is` confirmation message broadcasting.

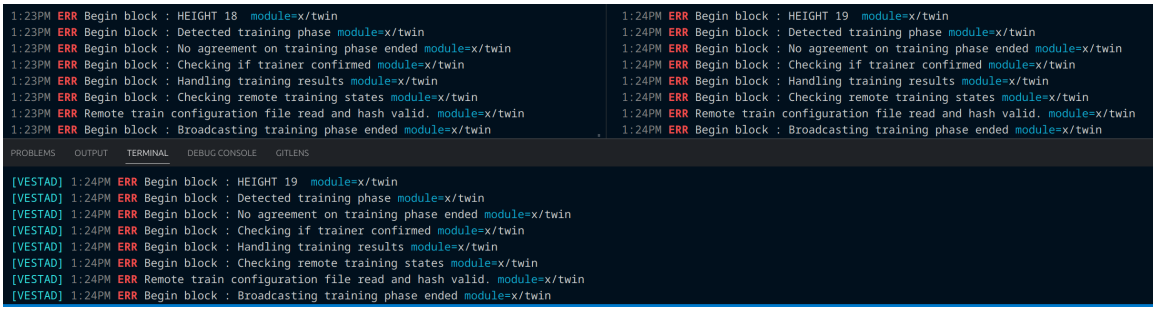


Figure 5.5: Logs of training test, during training phase, showing the block height at which trainers broadcasted the training-phase-ended confirmation message. Bottom: trainer1; left: trainer2; right: trainer3. Logs are produced at error level to filter out other modules logging information.

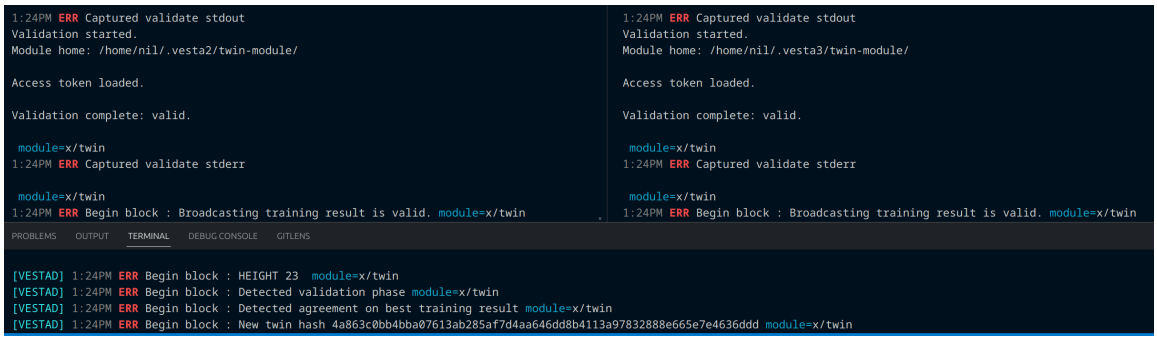


Figure 5.6: Log of training test, during validation phase, showing the block in which trainers completed their validation and broadcasted the best-result-is confirmation message. Bottom: trainer1; left: trainer2; right: trainer3. Logs are produced at error level to filter out other modules logging information.

Figure 5.7 shows the query output for the `training_state` data at the end of the training procedure. Registration of trainers' confirmation messages has been correctly handled and correctly stored in this state variable. From the field `training_phase_ended_confirmations` it can be seen that all trainers sent a `training-phase-ended` confirmation message. From the field `validation_state.map_validators_bestresulthash` it can be seen that all trainers sent a `best-result-is` confirmation message for the same training result (`4a863c0bb4bba07613ab285af7d4aa646dd8b4113a97832888e665e7e4636ddd`).

Figure 5.8 shows the query output for the `twin00` data at the end of the training process. The new `last_update` is the training module account. The new twin hash is the one the trainers agreed upon.

```
training_state:
  start_time: 2023-06-28T11:22:42.760503402Z
  training_configuration_hash: 9ad5206e68b401d62b68a9b829d4ef04ec3377128ad3e195b8730f3aaa447698
  training_phase_ended_confirmations:
    vesta15xhpaax757zz76x8p96ku2gwuyvh4wz0d5h400: true
    vesta1u89k93p3q7fj4l5u5um3sas2ege9wmt3tk6l64: true
    vesta1vh9m4j59gwnwxw9vye003hy3sjfh2lp25s000f: true
  twin_name: twin00
  validation_state:
    map_validators_bestresulthash:
      vesta15xhpaax757zz76x8p96ku2gwuyvh4wz0d5h400: 4a863c0bb4bba07613ab285af7d4aa646dd8b4113a97832888e665e7e4636ddd
      vesta1u89k93p3q7fj4l5u5um3sas2ege9wmt3tk6l64: 4a863c0bb4bba07613ab285af7d4aa646dd8b4113a97832888e665e7e4636ddd
      vesta1vh9m4j59gwnwxw9vye003hy3sjfh2lp25s000f: 4a863c0bb4bba07613ab285af7d4aa646dd8b4113a97832888e665e7e4636ddd
    start_time: 2023-06-28T11:24:02.632523912Z
    value: false
  value: false
```

Figure 5.7: Results of the training procedure saved in `training_state` variable, obtained through the query command `vestad query twin show-twin twin00` to read it from the blockchain store.

```
twin:
  creator: vesta15xhpaax757zz76x8p96ku2gwuyvh4wz0d5h400
  hash: 2fd5316da2a85285e35e3e4785bf1fefab14c409a3d9c8222881dabb3fa366fb
  last_update: vesta1w2en589shlyum57mqypfvfq5c7sdsk4dyuqhxn
  name: twin00
```


Figure 5.8: Resulting twin state after distributed training updating process.

Training has been requested for a training set of 70 samples. The machine used in this test has 8 Cores at 2.80 GHz cpu and 16 GB Ram DDR4. The total time needed for the training phase to complete has been 102 seconds, while total updating procedure requested 126 seconds. A single training of the digital twin model requested a maximum of 28 seconds when performed stand-alone, without involving the chain training. Total time increased since the local machine were running three trainings in parallel, one for each of the three trainers, and the blockchain application.

6 | Conclusions and future works

In this thesis, the implementation of a blockchain solution for digital twin life-cycle management has been proposed. Vesta, a blockchain built using Cosmos-SDK, has been used to generate and manage the state of a digital twin. The main target of this solution is consortia of enterprises that are involved in the developing and management of shared digital twins and that choose to operate in a trustless environment to enhance security over data integrity and ownership, but also choose to adopt central databases to store the digital twin models. Models saved in these databases will have their unique identifier (hash) saved also on the blockchain ledger so that any unauthorized modification of the models saved in the central database will be detectable and consequent actions can be taken.

The blockchain developed also integrates a protocol to orchestrate distributed training for neural-network-based digital twin models. The description and implementation of this protocol are given in this thesis. The blockchain solution proposed is based upon authorized addresses that can handle their digital twins or request training to the network providing the input data to generate a pool of analyses that will be parallelized on specific nodes of the blockchain. After training is completed, results will be uploaded to the central database, leading to the updating of the digital twin model identifier stored on the blockchain. This last operation will be automatically performed by the blockchain application after the latter has performed validation of the results uploaded.

This work started from a review of the digital twin technology and the blockchain technology, which composes the first part of the thesis.

A digital twin model has been created starting from a conceptual case study of the structural health monitoring of a composite plate. The digital twin model developed is a feedforward neural network model trained on the nonlinear analysis performed to obtain the load-displacement curves of the model in different damaged conditions. The digital twin model that has been developed accomplishes the prediction of the structural integrity inside the loading operative range of the composite plate. The development of this model composes the second part of this thesis.

In the third and last part, the blockchain protocol has been described. The blockchain application realized shows that blockchain technology is powerful and versatile and can be used to secure and manage digital assets of aerospace nature other than financial ones.

6.1. Future works

The described blockchain architecture can be extended in many of its aspects.

1. Vesta distributed training protocol can be extended to any optimization algorithm that starts from a pool of analysis that can be parallelized, and from which a unique best result has to be obtained. This includes also any analysis job in general that fits this analysis procedure.
2. The application features can be extended by implementing logics to automatically trigger the physical twin management from the digital twin diagnosis analyses.
3. Zero-knowledge proofs (ZK proofs) are an active research field in blockchain technology. These are methods based on finding a way to demonstrate the truth of a statement without disclosing any information beyond the fact the statement is true or false. For example, with a ZK proof one could prove possession of certain data without revealing the actual data. ZK could be used to add an extra layer of protection for a digital twin saved, providing proof of its existence without revealing anything about it.

Bibliography

[1] S. Al-Kuwari, J. Davenport, and R. Bradford. Cryptographic hash functions: Recent design trends and security notions. *IACR Cryptology ePrint Archive*, 2011:565, 01 2011.

[2] M. Amara and A. Siad. Elliptic curve cryptography and its applications. In *International Workshop on Systems, Signal Processing and their Applications, WOSSPA*. IEEE, may 2011. doi: 10.1109/wosspa.2011.5931464. URL `https://doi.org/10.1109%2Fwosspa.2011.5931464`.

[3] S. Bock, J. Goppold, and M. Weiß. An improvement of the convergence proof of the adam-optimizer, 2018. URL `https://arxiv.org/abs/1804.10587`.

[4] E. Buchman. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. University of Guelph, 2016.

[5] V. Buterin. Ethereum: A next generation smart contract and decentralized application platform. 2014. URL `https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf`.

[6] V. Buterin. On public and private blockchains, 2015. URL `https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains`.

[7] F.-K. Chang and K.-Y. Chang. A progressive damage model for laminated composites containing stress concentrations. *Journal of Composite Materials*, 21(9):834–855, sep 1987. doi: 10.1177/002199838702100904. URL `https://doi.org/10.1177%2F002199838702100904`.

[8] T. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, may 1978. doi: 10.1109/tse.1978.231496. URL `https://doi.org/10.1109%2Ftse.1978.231496`.

[9] Cosmos. Cosmos sdk features. URL `https://v1.cosmos.network/sdk`.

[10] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple

message diffusion to byzantine agreement. *Information and Computation*, 118(1): 158–179, apr 1995. doi: 10.1006/inco.1995.1060. URL https://doi.org/10.1006%2Finco.1995.1060.

[11] Q. H. Dang. Secure hash standard. Technical report, jul 2015. URL https://doi.org/10.6028%2Fnist.fips.180-4.

[12] S. Debnath, A. Chattopadhyay, and S. Dutta. Brief review on journey of secured hash algorithms. In *2017 4th International Conference on Opto-Electronics and Applied Optics (Optronix)*. IEEE, nov 2017. doi: 10.1109/optronix.2017.8349971. URL https://doi.org/10.1109%2Foptronix.2017.8349971.

[13] O. Dib, K.-L. Brousmiche, A. Durand, E. Thea, and E. Hamida. Consortium blockchains: Overview, applications and challenges. 09 2018.

[14] W. Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, may 1988. doi: 10.1109/5.4442. URL https://doi.org/10.1109%2F5.4442.

[15] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, nov 1976. doi: 10.1109/tit.1976.1055638. URL https://doi.org/10.1109%2Ftit.1976.1055638.

[16] J. R. Douceur. The sybil attack. In *Peer-to-Peer Systems*, pages 251–260. Springer Berlin Heidelberg, 2002. doi: 10.1007/3-540-45748-8_24. URL https://doi.org/10.1007%2F3-540-45748-8_24.

[17] Ethereum. Merkle proofs for offline data integrity, 2021. URL https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains.

[18] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, apr 1985. doi: 10.1145/3149.214121. URL https://doi.org/10.1145%2F3149.214121.

[19] E. Glaessgen and D. Stargel. The digital twin paradigm for future nasa and u.s. air force vehicles. April 2012. ISBN 978-1-60086-937-2. doi: 10.2514/6.2012-1818.

[20] M. Grieves and J. Vickers. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In *Transdisciplinary Perspectives on Complex Systems*, pages 85–113. Springer International Publishing, aug 2016. doi: 10.1007/978-3-319-38756-7_4. URL https://doi.org/10.1007%2F978-3-319-38756-7_4.

[21] E. B. Jae Kwon. A network of distributed ledgers. URL https://v1.cosmos.network/resources/whitepaper.

[22] N. Jansma. Performance comparison of elliptic curve and rsa digital signatures. 05 2004.

[23] H. Kakavand, N. K. D. Sevres, and B. Chilton. The blockchain revolution: An analysis of regulation and technology related to distributed ledger technologies. *SSRN Electronic Journal*, 2017. doi: 10.2139/ssrn.2849251. URL https://doi.org/10.2139%2Fssrn.2849251.

[24] M. Kapteyn, D. Knezevic, D. Huynh, M. Tran, and K. Willcox. Data-driven physics-based digital twins via a library of component-based reduced-order models. *International Journal for Numerical Methods in Engineering*, 123(13):2986–3003, jun 2020. doi: 10.1002/nme.6423. URL https://doi.org/10.1002%2Fnme.6423.

[25] M. G. Kapteyn and K. E. Willcox. From physics-based models to predictive digital twins via interpretable machine learning, 2020. URL https://arxiv.org/abs/2004.11356.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

[27] N. Koblitz, A. Menezes, and S. Vanstone. The state of elliptic curve cryptography. *Designs, Codes and Cryptography*, 19(2/3):173–193, 2000. doi: 10.1023/a:1008354106356. URL https://doi.org/10.1023%2Fa%3A1008354106356.

[28] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018. doi: 10.1016/j.ifacol.2018.08.474. URL https://doi.org/10.1016%2Fj.ifacol.2018.08.474.

[29] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996. doi: 10.1109/5.533956. URL https://doi.org/10.1109%2F5.533956.

[30] H. Li, R. Lu, L. Zhou, B. Yang, and X. Shen. An efficient merkle-tree-based authentication scheme for smart grid. *IEEE Systems Journal*, 8(2):655–663, jun 2014. doi: 10.1109/jsyst.2013.2271537. URL https://doi.org/10.1109%2Fjsyst.2013.2271537.

[31] L. Li, S. Aslam, A. Wileman, and S. Perinpanayagam. Digital twin in aerospace

industry: A gentle introduction. *IEEE Access*, 10:9543–9562, 2022. doi: 10.1109/access.2021.3136458. URL https://doi.org/10.1109%2Faccess.2021.3136458.

[32] V. G. Martínez, L. Hernández-Álvarez, and L. H. Encinas. Analysis of the cryptographic tools for blockchain and bitcoin. *Mathematics*, 8(1):131, january 2020. doi: 10.3390/math8010131. URL https://doi.org/10.3390%2Fmath8010131.

[33] R. C. Merkle. *SECRECY, AUTHENTICATION, AND PUBLIC KEY SYSTEMS*. UMI Research Press, 1982, 1979.

[34] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[35] M. Nastran. *MSC Nastran 2018 Nonlinear (SOL 400) User's Guide*. Hexagon, 2021.

[36] D. E. Perlini. *Design of M-346 aircraft's vertical tail in CFC by means of progressive failure analysis*. Politecnico di Milano, december 2014.

[37] S. F. Pitton. *Artificial intelligence techniques for the optimization of variable stiffness cylindrical shells*. Politecnico di Milano, april 2019.

[38] Pytorch. -. URL https://pytorch.org/.

[39] M. Rauchs, A. Glidden, B. Gordon, G. C. Pieters, M. Recanatini, F. Rostand, K. Vagneur, and B. Z. Zhang. Distributed ledger technology systems: A conceptual framework. *SSRN Electronic Journal*, 2018. doi: 10.2139/ssrn.3230013. URL https://doi.org/10.2139%2Fssrn.3230013.

[40] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, feb 1978. doi: 10.1145/359340.359342. URL https://doi.org/10.1145%2F359340.359342.

[41] C. M. Saggese. *Finite Element Progressive Failure Analysis of composite structures*. Politecnico di Torino, - 2019.

[42] S. Sayeed and H. Marco-Gisbert. Assessing blockchain consensus and security mechanisms against the 51% attack. *Applied Sciences*, 9(9):1788, apr 2019. doi: 10.3390/app9091788. URL https://doi.org/10.3390%2Fapp9091788.

[43] F. B. Schneider. The state machine approach: A tutorial. In *Fault-Tolerant Distributed Computing*, pages 18–41. Springer-Verlag. doi: 10.1007/bfb0042323. URL https://doi.org/10.1007%2Fbfb0042323.

[44] M. Shafto, M. Conroy, R. Doyle, E. Glaessgen, C. Kemp, J. LeMoigne, and L. Wang. *Modeling, Simulation, Information Technology and Processing Roadmap*. may 2010.

[45] M. Singh, E. Fuenmayor, E. Hinchy, Y. Qiao, N. Murray, and D. Devine. Digital twin: Origin to future. *Applied System Innovation*, 4(2):36, may 2021. doi: 10.3390/asi4020036. URL https://doi.org/10.3390%2Fasi4020036.

[46] D. System. Dassault system. URL [https://www.3ds.com/virtual-twin/life-sciences-healthcare](https://emea01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.3ds.com%2Fvirtual-twin%2Flife-sciences-healthcare&data=05%7C01%7C%7Cdb2d066b63fd45dd103b08db7a540ff7%7C84df9e7fe9f640afb435aaaaaaaaaaaa%7C1%7C0%7C638238273685134540%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJ
3D%7C3000%7C%7C%7C&sdata=05qseTF0NjY%2FzN0xGKZ%2B1X8ONGZ4urp2l3KBBotiW5w%3D&reserved=0).

[47] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee. Digital twin in industry: State-of-the-art. *IEEE Transactions on Industrial Informatics*, 15(4):2405–2415, apr 2019. doi: 10.1109/tii.2018.2873186. URL https://doi.org/10.1109%2Ftii.2018.2873186.

[48] A. Thelen, X. Zhang, O. Fink, Y. Lu, S. Ghosh, B. D. Youn, M. D. Todd, S. Mahadevan, C. Hu, and Z. Hu. A comprehensive review of digital twin — part 1: modeling and twinning enabling technologies. *Structural and Multidisciplinary Optimization*, 65(12), nov 2022. doi: 10.1007/s00158-022-03425-4. URL https://doi.org/10.1007%2Fs00158-022-03425-4.

[49] H. C. A. Tilborg. *Fundamentals of Cryptology*. Kluwer Academic Publishers, 2002. doi: 10.1007/b116810. URL https://doi.org/10.1007%2Fb116810.

[50] Wüst, Karl and Gervais, Arthur. Ethereum eclipse attacks. Technical report, 2016. URL http://hdl.handle.net/20.500.11850/121310.

[51] M. Xiong and H. Wang. Digital twin applications in aviation industry: A review. *The International Journal of Advanced Manufacturing Technology*, 121(9-10):5677–5692, jul 2022. doi: 10.1007/s00170-022-09717-9. URL https://doi.org/10.1007%2Fs00170-022-09717-9.

List of Figures

List of Tables