Journal of Manufacturing Processes 145 (2025) 211–235

Contents lists available at ScienceDirect

Journal of Manufacturing Processes

journal homepage: www.elsevier.com/locate/manpro

Research article

Machine learning-based early detection of malicious G-code manipulations in 3D printing

Hala Ali *, Alberto Cano, Irfan Ahmed

Department of Computer Science, Virginia Commonwealth University, Richmond, 23284, VA, United States

ARTICLE INFO

Keywords:
3D printing
Additive manufacturing
G-code attack
Machine learning

ABSTRACT

The increased adoption of 3D printing across various critical manufacturing sectors has made it a fruitful target for adversaries, particularly through the manipulation of G-code instructions that control the operations of 3D printers. Simple modifications to these instructions could significantly impact the integrity of 3D-printed objects. While side-channel analysis during printing is a common detection method, identifying potential malicious G-code before printing can save time and resources. Existing work relies on primitive encryption and hashing techniques and cannot distinguish between benign and malicious G-code instructions. It assumes that G-code files are benign and uses them as a reference model, focusing only on the integrity checking of G-code during storage and transmission. This paper introduces a novel automated approach to efficiently differentiate between benign and subtly manipulated G-code caused by filament, thermodynamic, and Z-profile attacks without requiring a reference model. As the first study leveraging recent advancements in Machine Learning (ML), we address several challenges in dataset generation, feature engineering, G-code segmenting and labeling, and ML classifier selection. We generate diverse G-code datasets to identify the optimal dataset characteristics and conduct a comprehensive formal analysis to extract the most suitable features. Efficient labeling strategies are employed at both layer and command levels, using the Multiple Instance Learning (MIL) paradigm for the former. We adopt the Bidirectional Long Short-Term Memory (Bi-LSTM) model enhanced by an attention mechanism and focal loss function for layer classification. Meanwhile, the Random Forest (RF) algorithm and Multilayer Perceptron (MLP) neural network model are used for command classification. All classifiers are designed to handle the imbalanced dataset. Experimental evaluation demonstrates the efficacy of our approach. The Bi-LSTM model achieves F1 scores up to 91.3% in detecting filament attacks, while the RF algorithm performs better in detecting nuanced thermodynamic and Z-profile changes at the command level, achieving F1 scores between 81.6% and 99.3%.

1. Introduction

3D printing, also known as additive manufacturing (AM), has been adopted across various critical sectors, such as energy supply, healthcare facilities [1], and firearms production [2,3], due to its ability to build precise and complex shapes [4,5]. However, this increased integration in sensitive manufacturing has made it an attractive target for malicious actors. Adversaries have exploited vulnerabilities at multiple stages of the 3D printing workflow, including modifying design files such as Computer-Aided Design (CAD) and STereoLithography (STL) files [6–8]. They have also manipulated G-code files, which contain structured instructions controlling 3D printer operations [9], and compromised the printer's firmware and bootloader [10,11].

While traditional hash and digital signature methods have been used to improve design file integrity, the 3D printing process remains vulnerable at multiple stages. The slicing software that converts STL files into layered G-code sequences may be susceptible to attacks [12], potentially generating malicious instructions. Additionally, G-code files can be compromised during sharing or utilization, further expanding the attack surface. Malicious modifications to these files involve creating cavities within the 3D object [8], disrupting filament extrusion [13,14], modifying infill structures [15,16], changing fan and printing speeds [10,17], adjusting nozzle and bed temperatures [18], and compromising the layer bonding and thickness [19,20]. These manipulations can significantly compromise the quality of 3D printed objects without visible deformations [21], potentially leading to severe real-world consequences. For instance, creating hidden cavities within a 3D-printed gun could cause dangerous malfunctions. Similarly, changes in the material density of a 3D-printed heart implant could have fatal

* Corresponding author.
E-mail addresses: alih16@vcu.edu (H. Ali), acano@vcu.edu (A. Cano), iahmed3@vcu.edu (I. Ahmed).

https://doi.org/10.1016/j.jmapro.2025.04.012
Received 6 August 2024; Received in revised form 15 March 2025; Accepted 3 April 2025
Available online 25 April 2025

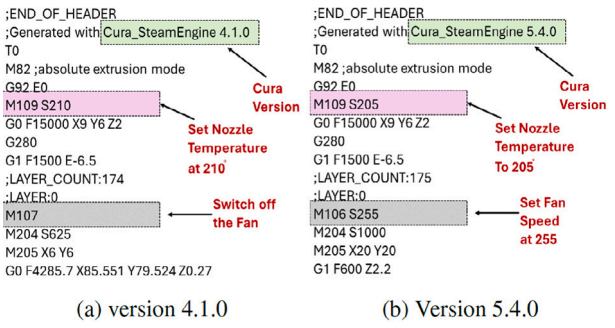

Fig. 1. Temperature and fan speed patterns in G-code generated by different versions of slicing software (Ultimaker Cura) for an identical design.

consequences for a patient. Belikovetsky et al. [14] demonstrated a practical example of real-world G-code attacks by creating cavities near the center of a drone's propeller, causing it to fail shortly after takeoff. Furthermore, Rais et al. [22] showed how subtle changes in filament extrusion and nozzle temperature could significantly impact the physical properties of printed objects.

To this end, several detection approaches focus on analyzing side channels during the printing process, such as surface roughness [23], nozzle temperature [24,25], acoustic emissions [25], infrared imaging [26], and inertia sensors [16]. However, we can save time and eliminate material waste by detecting G-code attacks before printing begins. In this direction, the existing work [27–29] is limited and relies on primitive encryption and hashing techniques that cannot analyze G-code files to distinguish between benign and malicious G-code instructions. These approaches assume that G-code files are benign and use them as a reference model, focusing only on integrity checking of G-code during storage and transmission. Moreover, they lack the ability to identify the specific nature, location, and potential impact of modifications, which is crucial for assessing risk and implementing effective mitigations in complex manufacturing environments. As a result, G-code files must be analyzed to ensure they do not contain potential malicious patterns.

Indeed, detecting malicious manipulations within extensive G-code files is challenging due to the subtle and complicated nature of G-code attacks, making manual analysis impractical and error-prone. For instance, filament attacks span multiple commands to achieve their sabotage goal. They target the filament length along the movement command in various ways, resulting in overlapping footprints within G-code data [22]. Furthermore, the magnitude of these attacks, such as the size of cavities and the extent of density change, varies not only across files but also between layers within the same file. Notably, such modifications can also be intentional design choices rather than attacks, as benign designs might necessitate different material densities in specific parts. Thermodynamic attacks pose another challenge by affecting entire layers within the G-code file through subtle adjustments to temperature and fan speed instructions. However, rule-based methods prove inadequate to detect such changes for several reasons. Firstly, slicing software variability generates unique temperature and fan speed patterns across the G-code file, even for identical designs and slicing parameters, as illustrated in Fig. 1. Secondly, user customization of slicing parameters can be misclassified as attacks by static rules. Thirdly, the validity of a configuration often depends on its position within the G-code file; as the length of layers varies with design size and shape, the locations of instructions shift accordingly. Lastly, validating millions of G-code lines against extensive rule sets is computationally expensive. Given these challenges, recent advancements in machine learning are promising and warrant a comprehensive study to explore the detection of complex and subtle manipulations in G-code files.

This study introduces a novel detection approach that accurately distinguishes between benign and subtle malicious manipulations in G-code files. It provides insights into the nature of these modifications without requiring a reference model, such as the original G-code file or its hash values. Moreover, this approach efficiently recognizes intricate malicious overlapping patterns within G-code commands. It can detect various attacks, including filament attacks, such as cavity and filament density variation; thermodynamic attacks involving nozzle (extruder) temperature, bed (build plate) temperature, and fan speed; and Z-profile attacks that adjust printing bed leveling. It is important to note that the impact of these attacks varies based on their magnitude, type, and target location within the object. Moreover, the same attack can have dramatically different consequences for objects of various sizes. While real-world evaluation is essential to confirm malicious intent, our focus on early detection leads us to flag these manipulations as potentially malicious. This strategy helps identify and mitigate possible attacks at the earliest stage. So, when manipulations are detected in a G-code file, our approach discards this file and prevents it from being sent to the 3D printer for execution, thereby enhancing the overall security of the 3D printing process.

As the first approach leveraging machine learning for detecting G-code attacks, several research challenges arise. A significant challenge lies in the lack of publicly available G-code datasets, particularly those containing documented attack scenarios. Consequently, generating a diverse and representative dataset becomes crucial, considering factors such as size, diversity, compactness, completeness, and realism. Moreover, identifying and extracting informative features to represent G-code commands efficiently is another challenge. This task requires capturing subtle changes, which demands domain-specific knowledge and an understanding of the context. Additionally, it is essential to determine the optimal way to segment and label the G-code files and find the optimal ML model to classify them. This process must consider the extensive length of G-code files, the sequential dependencies between commands, and the various attack patterns.

Therefore, we begin our study by discussing multiple factors while generating the G-code datasets, such as the diversity of 3D designs and slicing parameters and their impact on the effectiveness of detecting G-code attacks. These parameters, utilized by slicing software to convert 3D designs into G-code files, include printing speed, layer thickness, infill patterns, density, and direction. We then conduct a formal analysis of benign and malicious G-code samples to identify the most informative features to represent G-code instructions. These features encompass the command type and number, movement distance and direction angle of the print head, filament amount, thermodynamic settings including the nozzle and bed temperatures and fan speed, Z-parameter values that determine bed leveling, and layer characteristics such as thickness, number, and indicator. To identify the optimal feature set, we employ a wrapper-based model evaluation method [30]. This approach tests various combinations of features by assessing the machine learning model's performance on a test dataset. Through this process, we systematically investigate the impact of different features on detection accuracy, aiming to optimize our model's ability to identify G-code attacks.

Building on this foundation, we introduce two classification strategies, one focusing on entire layers and another on individual commands, considering the attack's behavior. The complex patterns of filament attacks necessitate a flexible approach to capture the full attack trace, leading us to adopt the Multiple Instance Learning (MIL) representation paradigm [31,32]. MIL provides a comprehensive approach widely employed in various fields, including image classification, text categorization, and web mining [33]. In the context of G-code, we consider each layer as a bag, and each command is an instance of that bag. If a sequence of commands within a layer is malicious, the entire layer is labeled as malicious. Conversely, thermodynamic and Z-profile attacks typically affect single commands, making command-level classification more suitable. This approach is particularly effective

for detecting malicious layers influenced by setting changes outside of their commands.

Our layer classification approach employs Bi-LSTM [34], a Recurrent Neural Network (RNN) variant well-suited for handling sequential data like G-code. Bi-LSTM's capacity to process sequences while considering both past and future context enhances its understanding of interdependencies among G-code commands. By incorporating an attention mechanism, we further improve the model's ability to identify malicious patterns within variable-length layers, allowing it to focus on the most relevant features. Furthermore, to handle the dataset imbalance resulting from varying layer counts across files and the number of attack-impacted layers, we use the focal loss function with this model [35]. On the other hand, and for command classification, we adopt the Random Forest (RF) algorithm [36] and Multilayer Perceptron (MLP) neural network model [37], which can effectively handle high-dimensional, non-linear relationships between the extracted features.

Therefore, the contributions of this paper are as follows:

- Propose an efficient machine learning-based early detection approach of malicious manipulations caused by G-code attacks, providing insights into the nature of these modifications without requiring a reference model.
- Generate and analyze diverse G-code datasets with varying 3D designs and slicing parameters to determine the optimal dataset characteristics for enhanced G-code attack detection.
- Conduct a formal analysis to identify the most informative features for G-code attack detection and determine the optimal feature set.
- Introduce efficient strategies to segment, label, and classify G-code instructions, considering the extensive length of G-code files and the nature of attacks.

Our experiments emphasize the critical role of diversifying dataset characteristics, including design, slicing parameters, and feature sets, for accurately detecting subtle malicious manipulations. Moreover, the Bi-LSTM model effectively detects filament attacks, achieving F1 scores up to 91.3%. Meanwhile, the RF algorithm performs better in detecting nuanced thermodynamic and Z-profile changes at the command level, achieving F1 scores between 81.6% and 99.3%. Additionally, the focal loss function significantly mitigates the impact of dataset imbalance, enhancing the detection approach's overall robustness. To support further research, our codes and datasets are available at: https://github.com/HalaAli198/ML-based-G-code-Attacks-Detection.

The rest of the paper is organized as follows. Section 2 covers the background and attack model. Section 3 reviews related work. Sections 4 and 5 address the research challenges and proposed methodology. Section 6 presents the experimental evaluation. Section 7 illustrates the efficacy of our methodology in real-world scenarios, while Section 8 analyzes its scalability. Finally, Section 9 discusses the limitations and Section 10 concludes the paper.

2. Background and attack model

2.1. 3D printing workflow

The 3D printing process consists of four main stages [38], as illustrated in Fig. 2. It starts with creating a digital 3D model using CAD software, which is subsequently saved as an STL file [39]. This file encapsulates the object's surface geometry as a mesh of interconnected triangles. Next, specialized slicing software converts the STL file into layers of precise G-code instructions that control printing parameters, including nozzle movements, temperature settings, fan speeds, and material extrusion rates. Finally, the printer's firmware interprets these G-code instructions to construct the object layer by layer physically [15, 40].

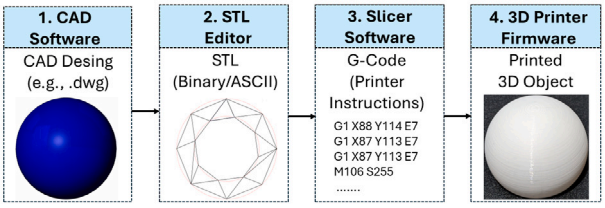

Fig. 2. 3D printing workflow.

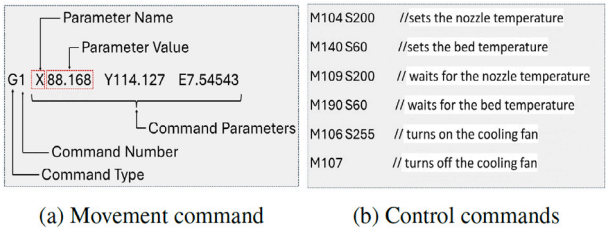

Fig. 3. Examples of G-code commands.

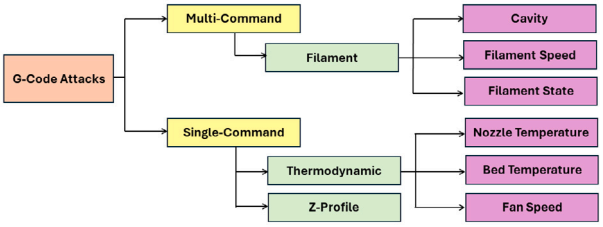

Fig. 4. G-code attacks categories.

2.2. G-code data

G-code files contain critical instructions that control 3D printer operations [41], including print head coordinates, material extrusion amount, printing speed, and temperatures for both the nozzle and build plate (bed). These temperature settings ensure proper material adhesion and flow during the printing process. G-code also controls fan speed, affecting the cooling rate of extruded material and consequently impacting print quality. Moreover, the bed leveling is adjusted according to the values of the Z-axis parameter of movement G-code commands. Proper bed leveling is essential for preventing common issues such as warping, curling, and delamination, enhancing overall print quality [17].

Fig. 3 illustrates examples of these G-code commands. Specifically, Fig. 3(a) demonstrates a movement command that positions the print head at coordinates (X = 88.168 mm, Y = 114.127 mm) and sets the filament extrusion length to 7.54543 mm. Fig. 3(b) shows examples of M-commands used to control temperatures and fan speed during printing.

2.3. G-code attacks

This study classifies the G-code attacks based on the number of manipulated commands into multi-command and single-command attacks, as shown in Fig. 4.

Multi-command attacks

This category of attacks involves manipulating multiple parameters across movement commands within the internal structure of the targeted object, potentially compromising its strength without visible deformations [22,40]. These attacks include:

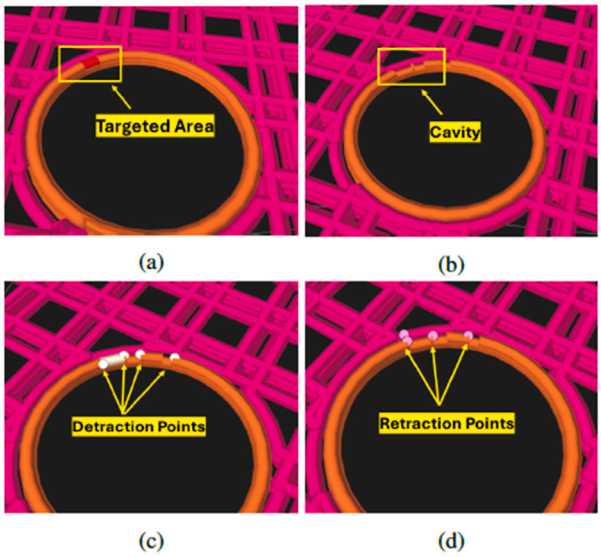

Fig. 5. Impacted G-code sample by filament cavity attack.

- Cavity attack ($\mathcal{CV}$): This attack physically creates cavities within the printed object by dividing the targeted movement distance into shorter segments and selectively disabling material extrusion at some segments. It involves inserting G1 movement commands to create the segments and to retract and push the material before and after the cavity. The total filament allocated to the generated segments and the disabled extrusion amount equals the original filament amount of the targeted command. Fig. 5 shows an example of G-code impacted by this attack, visualized using the *Zupfe GCode Viewer* [42]. This figure demonstrates one pattern of the attack, creating three consecutive cavities. Each cavity is formed by detracting the filament to make a clear cavity and then pushing it back. In the visualization, white circles indicate detraction locations, while the purple circles represent retraction locations.
- Density variation through Filament Speed Attack($\mathcal{FS}$): This attack physically alters the filament motor speed to manipulate the amount of extruded filament, introducing localized weaknesses within the printed object. In the context of G-code, it focuses on modifying the *"E"* parameter of targeted G1 commands to reduce the filament amount extruded at these commands while compensating for it elsewhere within the same layer, maintaining the overall object weight. However, the subtle manipulations caused by this attack, measured in millimeters, cannot be visualized in G-code viewers or detected by the naked eye due to the minimal scale of the changes.
- Density variation through filament state attack($\mathcal{FT}$): This attack physically adjusts the filament motor state (ON/OFF) before reaching the targeted regions. It aims to mute filament extrusion over the entire movement distance by removing the *"E"* parameter of the targeted G1 command or setting identical *"E"* parameters of consecutive G1 commands, followed by G92 to push the material with the following commands. Fig. 6 shows an impacted G-code sample by this attack.

Single-command attacks

This category includes attacks that inject, delete, or modify a single G-code command, such as:

- Thermodynamic Attacks: These attacks physically alter the nozzle temperature ($\mathcal{T}_n$), bed temperature ($\mathcal{T}_b$), and fan speed ($\mathcal{F}_s$) by manipulating the corresponding G-code commands, such as M104/M109 for nozzle temperature, M140/M190 for bed temperature, and M106/M107 for fan speed. These attacks can lead to defects in the printed object, including shrinking and warping [43–45].

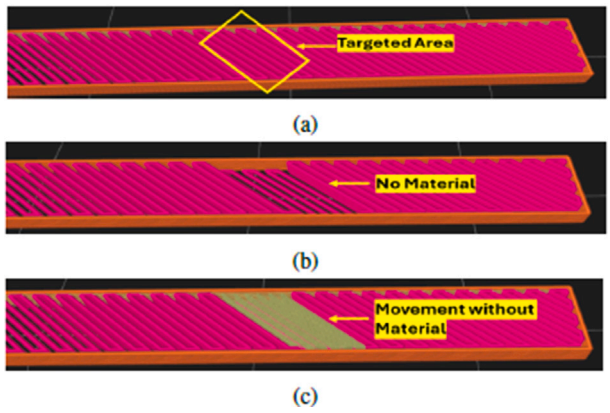

Fig. 6. Impacted G-code sample by filament state attack.

- Z-profile Attack ($\mathcal{Z}_p$): This attack targets the printing bed level by manipulating the *"Z"* parameter of G0 and G1 commands [17,46], consequently affecting the layer thickness of the printed object. Such alterations can compromise the mechanical properties of the printed object, including its strength [15] and surface roughness [47].

2.4. Attack model

We assume that the 3D printer operates in a secure environment and can accurately create an intended object when receiving a benign G-code file. However, G-code files can be compromised beforehand and require analysis to ensure they do not contain any potential malicious instructions. For instance, users can generate and share G-code files for printing, or open-source G-code files can be reused, which may contain malicious instructions. Our approach serves as a primary line of defense, examining G-code files, identifying the attack's type, and forwarding only valid files to the 3D printer.

The attacker aims to sabotage the quality of the printed object by compromising the G-code file before it arrives at the printer, which can be achieved throughout the lifecycle of G-code file generation, sharing, and utilization. Fig. 7 illustrates how an attacker can compromise a G-code file. One method is to target the slicer software and inject malicious instructions during the G-code file generation process. Kurkowski et al. [12] demonstrated this attack by installing a kernel driver on the machine hosting the slicer software, which locates G-code instructions in the slicer process memory and injects malicious code. Additionally, Moore et al. [48] discovered vulnerabilities in the Cura slicer software that attackers can exploit to access G-code data in memory. Another approach for the attacker is to directly access the G-code stored on the host machine's disk. Moreover, when G-code files are transferred over the network to the printer, an adversary can intercept and modify the files during transmission, injecting malicious G-code instructions.

3. Related work

This section discusses various methods proposed to detect G-code attacks during printing by analyzing side-channel data. Additionally, it explores pre-printing detection approaches that utilize cryptographic techniques such as hashing, digital signatures, and encryption.

3.1. Side-channel based detection

Researchers have leveraged several machine-learning techniques to detect malicious G-code manipulations by analyzing side-channel data, including sensor readings and images captured by thermal, infrared (IR), and optical cameras. Si et al. [23] worked on monitoring the fan speed changes during printing and predicting their impact on

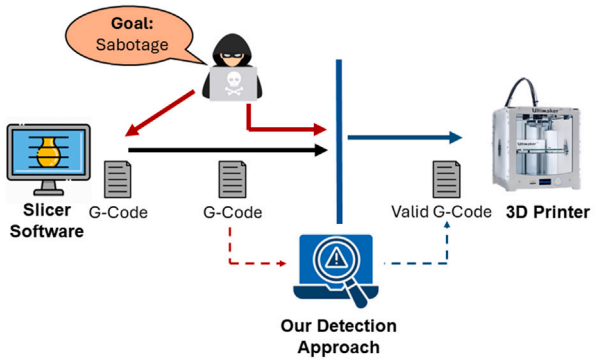

Fig. 7. Sabotage attack model.

print quality to detect any malicious manipulations. Their approach employs K-means clustering to determine optimal printing parameters, including layer thickness, infill density, and fan speed, while neural networks are used to predict surface roughness. Zhou et al. [49] focused on detecting alterations in part design, such as void creation, and G-code parameters, such as layer thickness modifications, using side-channel vibration signals. Their system employs the Echo State Network (ESN) to extract features, which are then used with both machine learning classifiers, including Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP), and Multivariate Exponentially Weighted Moving Average (MEWMA) control charts to identify abnormalities. Zhang et al. [50] worked on predicting the size of internal voids in printed parts, considering the impact of five printing parameters, such as layer thickness, sintering temperature, ramp ratio, nozzle temperature, and printing speed. Their approach adopts Support Vector Regression (SVR) and Neural Networks (NN) to predict void percentages based on these parameters. Whereas Khanzadeh et al. [51] aimed to detect the geometric deviations and directions in the printed part using laser-scanned coordinate data. Their method uses a Self-Organizing Map (SOM) to cluster the geometric deviations into different types based on magnitude and direction, linking them to specific process conditions, particularly infill percentage and extruder temperature.

Beyond machine learning techniques, researchers have explored other methods to detect G-code attacks. Gao et al. [17] proposed a real-time monitoring system that uses multiple sensors, including an accelerometer and magnetometer attached to the printer extruder, an accelerometer on the print bed, and a camera. The system aims to reconstruct and verify four key printing attributes: layer thickness, nozzle speed, internal infill design, and fan speed. It employs various techniques like Kalman filtering, image processing, and audio analysis to estimate these parameters. This system can identify potential firmware-based attacks that manipulate the printing process by comparing the reconstructed values to the expected G-code instructions. Kurkowski et al. [52] relied on analyzing the data collected from three side channels, including the build plate level, the print chamber temperature, and the laser firing time. Their approach involves creating a baseline from the sensor readings without requiring access to the toolpath code to detect anomalies in the powder bed fusion process. Haris et al. [21] introduced a framework that uses external sensors to monitor the nozzle movement, filament extrusion, and temperatures of the 3D printer. It compares the sensor data against expected values derived from the G-code instructions to detect anomalies. The framework analyzes the printing process layer-by-layer in both spatial and temporal domains. It can detect various attacks, including those targeting nozzle kinematics, filament extrusion, and thermodynamics. Moore et al. [53] monitored the current supplied to actuators (i.e., motors of X and Y steppers, nozzle, and build plate) during the printing process. Their system compares the resulting power traces to those from known benign prints. Deviations in the traces that exceed a set threshold indicate potential attacks. Such attacks include insertions, deletions, and reordering of individual G-code movement commands. Chhetri et al. [54] proposed a method to detect kinetic cyber-attacks during 3D printing by monitoring acoustic emissions. Their approach uses machine learning techniques such as Gradient Boosting Regression (GBR) and Logistic Regression (LR) to model the relationship between control parameters (e.g., nozzle speed, axis movement, distance) and observed acoustic emissions. The system continuously compares actual emissions to those expected based on G-code instructions, flagging deviations beyond set thresholds as potential attacks. This method can detect changes in printing parameters like nozzle speed in X and Y directions.

Other detection methods have focused on analyzing image, audio, and video data acquired during printing. Bhandarkar et al. [55] aimed to detect warpage in 3D-printed polymer parts through real-time image capture. They classified these images into warped and unwarped categories using convolutional neural networks (CNN). Wu et al. [56] addressed infill structure attacks by analyzing manipulations in images of printed parts, utilizing different machine learning algorithms such as Random Forest (RF) and k-Nearest Neighbors (kNN). Al-Mamuna et al. [57] detected changes in the internal structure of printed objects by analyzing the captured video of each layer during printing. They applied adaptive image segmentation to extract high-contrast texture regions relevant to the printing path and characterized the distribution of geometric features for each layer. Yang et al. [58] focused on detecting malicious changes in the infill density of printed objects by analyzing both audio and video signals. Their approach compares the Fast Fourier Transform (FFT) of a benign reference audio signal with the test signal using the Wasserstein distance. They also reconstructed the extruder's movement path from video recordings and compared it to the expected G-code using the Hausdorff distance. Belikovetsky et al. [16] explored audio signals from side-channel emanations to verify the integrity of printed objects. Their approach adapts audio fingerprinting techniques, similar to music recognition apps, to classify 3D printer sounds. By recording and segmenting the audio of a known normal print as a reference, their system can detect deviations in subsequent prints that may indicate tampering.

Furthermore, recent studies have focused on improving print quality by detecting anomalies in G-code using various vision-based monitoring techniques. Goh et al. [59] developed an on-site monitoring system using a camera attached to the print head that feeds video to object detection models for identifying defects. The researchers trained various YOLO models to detect and classify printing anomalies, such as under-extrusion and over-extrusion, achieving mean average precision scores of over 80%. They also employed ONNX optimization to improve inference speed to approximately 70 FPS. Moreover, their approach enabled real-time G-code corrections during printing when defects are detected, thus enhancing 3D print quality. Petsiuk and Pearce [60] introduced a method that compares images of printed layers with reference images generated from G-code using Blender software [61]. They applied Histograms of Oriented Gradients (HOG) with twelve similarity metrics, such as Pearson's r, Spearman's rho, cosine, and Dice. However, this approach requires no preliminary training data and can detect errors early enough to pause printing or implement future automated corrections. Unlike vision-based approaches, Kumar et al. [62] developed a vibration-based method for detecting mechanical anomalies using low-cost sensors mounted directly on the printer. Their LSTM implementation achieved 97.17% accuracy when comparing four machine learning algorithms. This approach automatically terminates printing upon detecting anomalies, targeting mechanical issues like component failure and belt slippage that vision-based systems typically miss.

3.2. Pre-printing detection

These approaches aim to maintain the integrity of G-code files and secure them during transmission and storage before printing starts, using cryptographic techniques such as hashing, digital signatures, and encryption. Li et al. [27] grouped G-code commands into blocks, each randomly mapped to a corresponding *mapped block*. Authentication bits, generated from X and Y coordinates using a hash function, are embedded into the least significant bits of these coordinates. Recovery bits, consisting of the original X and Y coordinate bits, are embedded in the mapped block. The 3D printer verifies the authentication bits to detect tampering and restores authentic blocks using the recovery bits. Tampered blocks are partially recovered if their mapped block is authentic. However, this method can only detect changes in X and Y coordinates, requires the original G-code for comparison, and cannot detect injected or removed commands. Oligshclaeger et al. [28] proposed an end-to-end encryption method for G-code files between the user and the printer facilitated by a third-party computing server. This server manages user interactions, encryption, decryption, storage of encrypted G-code files, and key retrieval for the 3D printer. The approach divides the G-code into parts, using unique encryption and decryption keys for each part. However, it suffers from high computational complexity due to generating multiple symmetric encryption keys with the G-code file and extensive communication between the user, 3D printer, and server. Shi et al. [29] introduced a blockchain-based approach to secure the G-code file transmission between the designer and the manufacturer responsible for printing. The G-code file is segmented into layers, each hashed using the SHA256 function and encrypted with the manufacturer's public key using the RSA algorithm. These encrypted layers and their hash values are stored on the cloud. Upon receiving the encrypted G-code file, the manufacturer decrypts the layers, verifies the stored hashes against the original values, and prints the validated G-code file. While this approach offers some security benefits, it may flag legitimate modifications as false alarms without insights into changes, fail to address modifications before hashing or after verification, and cause high computational complexity due to the asymmetric encryption and decryption of each G-code layer.

We compare our work with these proposed approaches as they represent the current state-of-the-art in pre-printing G-code protection. While these methods offer valuable contributions, they have limitations that our approach aims to address. Table 1 presents the critical distinctions between our approach and these proposed methods, including primary focus, employed techniques, manipulation detection capabilities, reference model dependency, and operational efficiency. We leverage machine learning techniques to detect potential malicious manipulations in G-code and provide insights into their nature. Moreover, our method does not require a reference model, third-party involvement, or integration with the 3D printer's firmware. By utilizing pre-trained machine learning models for inference, we can improve the operational efficiency compared to the computationally intensive operations of other methods. However, we conduct both layer-level and command-level analyses, offering a more detailed examination of the G-code. Additionally, the use of machine learning enables our approach to efficiently handle large datasets and adapt to new types of manipulations by frequently updating the datasets and retraining the models. Furthermore, Fig. 8 demonstrates the limitations of hash functions in distinguishing between benign and potentially malicious manipulations, as they do not provide insight into the nature or intent of the changes. For example, replacing the M106 S0 command with the M107 command in a benign G-code file results in different SHA1 hash values despite both commands being functionally equivalent in turning off the fan. This highlights the necessity of our machine learning approach, which can analyze the context and nature of changes more effectively than traditional hash-based methods.

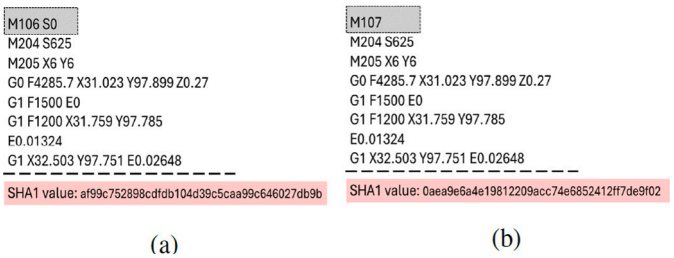

Fig. 8. Comparison of SHA1 hash values of functionally equivalent G-code commands.

4. Challenges addressed

This section discusses the challenges we faced while developing our ML-based detection approach.

C1: Complex nature of G-code attack patterns. The primary challenge in this research lies in identifying intricate malicious manipulations within extensive G-code files. Filament attacks, in particular, exhibit overlapping patterns and varied impacts depending on the targeted object's characteristics. For instance, the filament cavity attack can appear in ways that mimic other filament attack types. It may remove the filament length parameter *"E"* across consecutive commands, resembling the filament state attack, or reduce the value of this parameter similarly to the filament speed attack, but with the intent of creating cavities. Moreover, the magnitude of these attacks can vary across layers and files, with differences in the number of targeted commands, cavity sizes, the ratio of change in filament density, and the ratio of muted filament. Such variations result in various malicious patterns that necessitate context-aware detection methods.

While thermodynamic and Z-profile attacks leave more visible footprints, they present their own challenges. Changes in temperatures, fan speed, and bed leveling settings might be intentional design choices rather than malicious alterations. For example, different object parts may require specific temperatures or bed levels. However, the impact of these changes depends on their location within the G-code file; modifications during non-critical operations (e.g., homing the print head) may not affect object integrity. Additionally, these configurations often follow specific patterns based on slicing software, material, and object geometry, making it difficult to distinguish between intentional and malicious alterations. The lack of a reference model further complicates the detection process, as there is no baseline for comparison. These challenges raise our primary research question (RQ1):

> RQ1: How can we efficiently distinguish between benign and potentially malicious manipulations while providing insights into their subtle nature without requiring a reference model or real-time monitoring of the printing process?

C2: Lack of G-code database availability. One of the main challenges in this research is the lack of publicly accessible G-code datasets, particularly those representing various attack scenarios. G-code is often treated as proprietary, resulting in limited availability of documented attacks and corresponding files. Organizations are reluctant to share G-code samples due to intellectual property concerns, leading to a lack of diverse and representative datasets for training and testing machine learning models. This challenge leads to the second research question (RQ2) in this study:

> RQ2: How can we generate an efficient dataset for G-code analysis, considering factors such as size, completeness, compactness, and realism?

Table 1
Comparison of G-code security approaches.

Aspect	Li et al. [27]	Oligshclaeger et al. [28]	Shi et al. [29]	Our approach
Primary Focus	Integrity protection	Transmission security	Integrity protection	Malicious manipulation detection
Main Technique	Block mapping and embedding	End-to-end encryption	Blockchain	Machine learning
Identify Manipulation Nature	No	No	No	Yes
Analysis Level	Block	Block	Layer	Layer and Command
Reference Model Required	Yes	No	Yes	No
G-code Modification Required	Yes	No	No	No
Encryption Used	No	Yes	Yes	No
Third-party Dependency	No	Yes	Yes	No
Printer's Firmware Involvement	Yes (for verification)	Yes (for decryption)	No	No
Operational Efficiency	Low (per-print verification)	Low (multiple encryptions per file)	Low (blockchain operations)	High (fast ML inference)

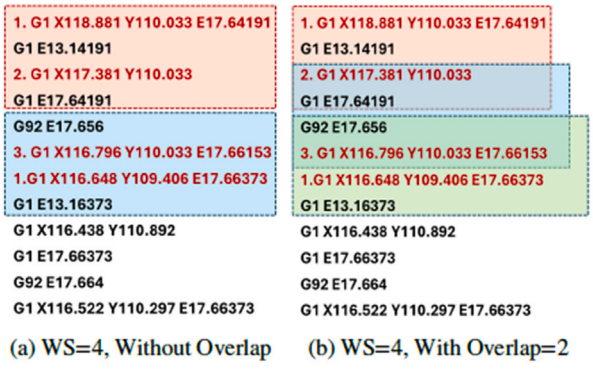

Fig. 9. Window slicing of a cavity attack footprint.

C3: Identifying informative feature set for G-code attack detection. Machine learning models may struggle with extracting features that distinguish subtle patterns, such as those caused by G-code attacks. This difficulty arises due to their lack of deep contextual understanding. However, traditional algorithms relying on predefined features may miss nuanced details hidden within raw data, especially those requiring domain-centric knowledge. While human experts with in-depth 3D printer understanding might identify essential features targeted by malicious attacks, the effectiveness of a machine learning model depends on the quality of its training data, potentially overlooking complex contextual nuances and thus struggling with identifying the context-specific features accurately. This challenge raises the third research question (RQ3):

> RQ3: How can we identify and extract the optimal feature set to efficiently detect the subtle malicious modifications caused by G-code attacks?

C4: Feeding sequential G-code data to machine learning models. G-code can contain hundreds of thousands of instructions, making it impractical to feed the entire sequence of instructions to an ML model. Filament attacks often span multiple commands, and objects of varying sizes result in G-code sequences of different lengths, complicating the determination of an appropriate chunk size that captures the entire attack trace. For example, Fig. 9 demonstrates the limitations of employing a sliding window approach to divide a G-code sample impacted by the cavity attack into smaller chunks. As shown in Fig. 9(a), using a fixed window size (e.g., WS=4) without overlap causes the attack footprint to be split across multiple windows, potentially missing crucial parts of the attack sequence. Conversely, Fig. 9(b) illustrates that while using overlapping windows may capture more information, this can result in intermingled portions of different consecutive attack footprints, leading to improper capture of the attack patterns.

Moreover, entire layers might be impacted by malicious changes outside of their intended commands. In such scenarios, adjustments to settings like temperatures, fan speed, or bed leveling can be introduced early in the G-code file or between the layers. Therefore, the diverse nature of attack categories necessitates appropriate segmenting and labeling strategies for effective classification, raising the following research question (RQ4):

> RQ4: How can we efficiently segment, label, and feed the sequence of G-code commands to an ML model, considering the varying nature of G-code attacks?

C5: Selecting the optimal machine learning model. Manual detection of malicious manipulations within G-code files is cumbersome and error-prone due to the extensive length of these files and the subtle nature of changes. As mentioned earlier, the footprints of filament attacks can overlap and vary across files with different designs, shapes, and infill structures. Additionally, settings such as temperatures, fan speed, and bed leveling follow diverse patterns across layers and files. The validity of instructions controlling these settings often depends on their position within the G-code file, and as layer lengths vary with design geometry, the locations of these instructions shift accordingly. These complexities make rule-based detection methods impractical for identifying potential malicious manipulations within G-code. Furthermore, validating each command or sequence of commands against predefined rules is computationally expensive, given the extensive length of G-code files.

Machine learning presents a robust solution to these challenges. Unlike manual or rule-based detection methods, which are often time-consuming and prone to errors, machine learning algorithms can automatically identify subtle and complex manipulations by analyzing vast amounts of data. Thus, by leveraging machine learning techniques, we can significantly enhance detection accuracy, minimize false positives, and improve operational efficiency. However, the effectiveness of this approach relies on selecting and optimizing the most suitable ML classifier for the task. This classifier must be capable of capturing the sequential interdependence between G-code commands, understanding feature relationships, and recognizing complicated attack footprints. This consideration leads us to the last research question (RQ5):

> RQ5: How can we identify the suitable ML model for efficiently recognizing malicious manipulations, given the sequential nature of G-code data and complex attack footprints?

5. Research methodology

This section provides detailed answers to the aforementioned research questions, outlining our methodology for detecting malicious G-code manipulations. Our process begins with the generation of G-code files, followed by a preprocessing phase. During preprocessing, we remove comments and file headers and filter the commands to focus on movement, filament retraction, and control instructions. We then represent each G-code file as a sequence of K layers, where each layer comprises n commands. Each command is characterized by a vector of m features identified through a comprehensive formal analysis. The G-code files are subsequently segmented into layers and commands,

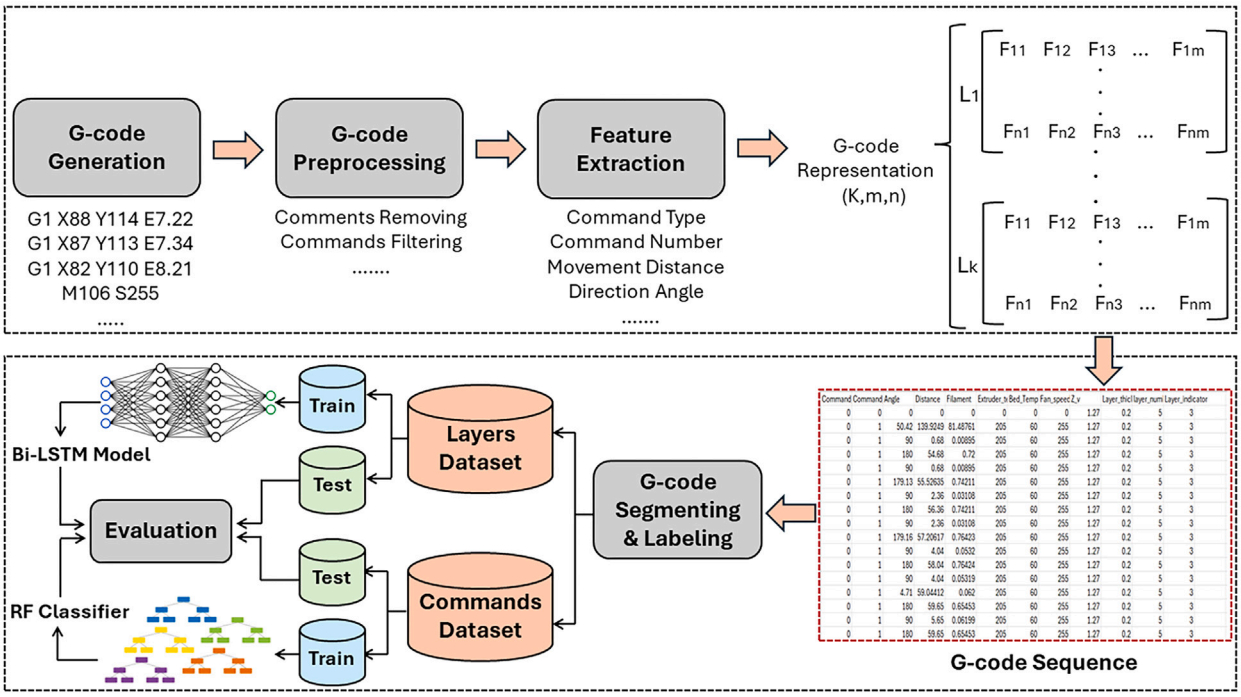

Fig. 10. Proposed framework.

which are labeled and stored in datasets. These prepared datasets are then split into training and testing sets for machine learning model development and evaluation. These steps are illustrated in Fig. 10 and further discussed in this section.

5.1. G-code dataset generation

To create a G-code dataset, there are three critical characteristics that should be considered depending on the research objectives, which are:

- The objects included in the dataset and their geometry complexity.
- The slicing configurations used to slice the objects to generate the G-code.
- The features extracted by analyzing the G-code.

There are two studies so far in the literature that created G-code datasets. Lasluisa et al. [63] focused on G-code parameter optimization using K-means clustering. Their dataset included only one object (the 3DBenchy model), with a specific focus on control commands such as M104, M109, M140, and M190 for temperature control, G1 for speed control, and M221 for flow control. For slicing configurations, they considered a specific set of parameters such as the layer thickness, print speed, perimeter speed, extruder temperature, bed temperature, and flow rate, with a fixed infill density of 10% and infill speed of 80 mm/s. The 3DBenchy model is then sliced with multiple parameter combinations to generate the training dataset. Wu et al. [9] adopted the LSTM model to predict mechanical properties (specifically Axial and Shear forces) directly from G-code. Their dataset was considerably focused, examining only a 40 mm × 40 mm × 1 mm plate structure using a PolyJet 3D printer, with specific slicing parameters including infill pattern (with seven variations), infill density, infill direction, and filament diameter. Their complete dataset comprised 13,048 entries, each representing a G-code file with these four features and corresponding mechanical test values. In contrast, our dataset offers substantially broader coverage and deeper analysis of G-code characteristics, designed for G-code manipulations detection. We consider multiple levels of complexity in design geometry, a diverse set of slicing parameters while slicing the design, and multiple features to represent the generated G-code.

Table 2
Print profiles of Ultimaker3 3D printer.

Print profile	Layer thick. (mm)	Init. nozzle temp. (°C)	Nozzle Temp. (°C)	Init. print speed (mm/s)	Print speed (mm/s)
Extra Fine	0.06	195	200	60	6
Fine	0.1	200	205	70	10
Normal	0.15	200	205	80	22.5
Fast	0.2	205	210	70	30

We focus on generating and analyzing G-code datasets for Fused Deposition Modeling (FDM) 3D printers, using the Ultimaker 3 printer as the primary case study. While we implement and evaluate the detection efficiency of our proposed approach using this specific printer, it still can be adaptable to various G-code datasets and generalizable to other 3D printers. Despite slight variations in G-code formats, the fundamental structure and commands remain primarily consistent across printers [64,65]. The feature engineering process, segmenting, labeling, and classification methods can be easily adapted to accommodate the specific characteristics of different printers. It is important to note that 3D printers have default configurations based on the materials they use. For instance, the Ultimaker 3 printer has four default profiles that specify various printing parameters, including the initial nozzle temperature for the first and subsequent layers, layer thickness, and initial print speed for the first and following layers, as shown in Table 2.

Dataset design objects

Regarding the choice of designs included in the training, we consider a range from simple geometrical structures to highly complicated shapes, as shown in Fig. 11. For instance, some designs exhibit medium complexity, characterized by sharp angles, intricate curves, or irregular outlines, as shown in Figs. 11(e), 11(f), and 11(g). The most complex designs feature highly complex and detailed shapes with irregular

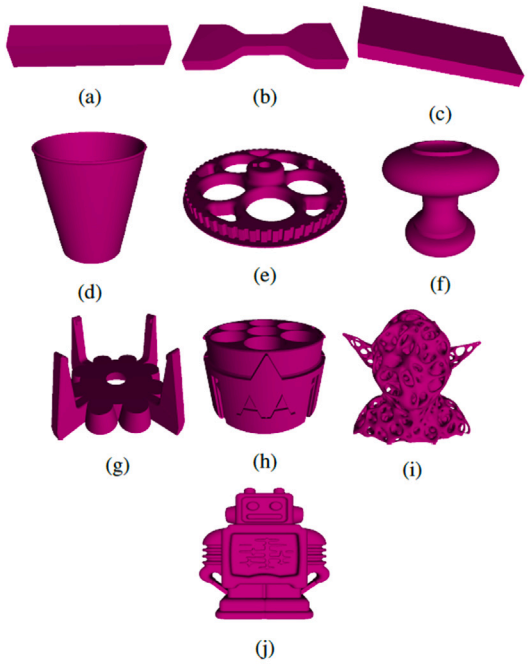

Fig. 11. Example designs used to generate the training dataset.

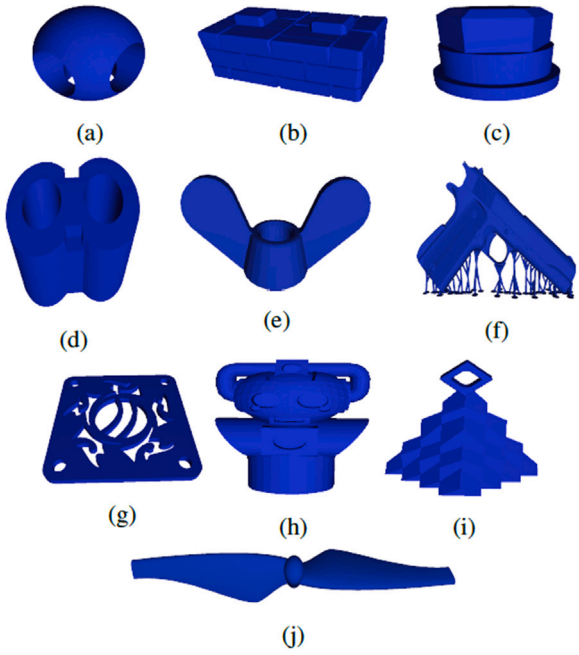

Fig. 12. Example designs used to generate the testing dataset.

patterns or textures, illustrated by Figs. 11(h), 11(i), and 11(j). In contrast, our test dataset includes 180 unique designs for evaluating the ML models. A small subset of these designs is shown in Fig. 12 using the *ViewSTL Online Tool* [66]. This collection includes critical objects such as 3D-printed guns (Fig. 12(f)), drone propellers (Fig. 12(j)), fans, and various manufacturing components. By incorporating such diverse designs, we cover a broad spectrum of additive manufacturing applications, from consumer products to potentially sensitive or regulated items.

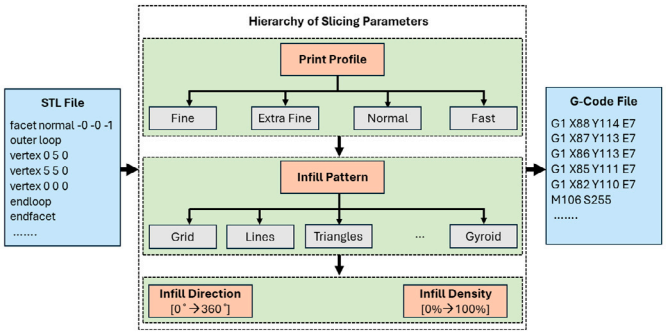

Fig. 13. Hierarchy of slicing parameters.

Table 3
Slicing parameters, corresponding G-code commands, and their Impact on the printed object.

Parameter	G-code	Impact	Ref.
Infill Pattern	(X, Y) of G0, G1, E of G1	Strength, print time, material usage	[69,70]
Infill Density	E of G1	Strength, material usage	[46,71]
Infill Direction	(X, Y) of G0, G1, E of G1	Strength, print time, stiffness	[72–74]
Infill Retraction	E of G92	Stringing, surface quality	[19,75]
Nozzle Temp.	S of M104 , M109	Layer adhesion, warping, material flow	[18,43]
Bed Temp.	S of M140 , M190	Layer adhesion, warping, material properties	[18,76]
Fan Speed	S of M106 , M107	Layer adhesion, warping, material properties	[17,44]
Layer Thick.	Z of G0, G1	Surface finish, resolution	[46,71]

Slicing process

To generate G-code files, we first download trusted STL files (designs) from reputable sources such as *Thingiverse* [67] and slice them using Ultimaker Cura 4.1.0 software [68]. This software uses various parameters while generating G-code, including print profile, infill characteristics, temperature settings, and fan speed. The slicing process of each design begins by selecting a print profile that controls various settings, as shown in Table 2. With each profile, there are various infill patterns to be used, such as Grid, Lines, Triangles, Tri-Hexagon, and others. These patterns have an infill density ranging from 0% to 100% and an infill direction from 0° to 360°. For FDM-based printers that use PLA material, the nozzle temperature is usually set between 195° and 210°. Additionally, the bed temperature is typically maintained at 60° to promote proper adhesion and minimize warping of the printed object. Fig. 13 presents this hierarchy of parameters that guides our feature extraction and analysis processes.

Define malicious G-code

G-code files are generated using various combinations of the above-mentioned slicing parameters. These files exhibit consistent patterns in temperatures, fan speed, and layer thickness settings. For example, the fan speed typically starts at 0° for the first layer, increases to 85° and 170° for the second and third layers, respectively, and remains at 255° for subsequent layers, while the bed temperature usually remains constant at 60° across all layers. Such benign patterns can vary significantly depending on the used slicing software, printer model and profiles, material type, and user customization. However, given the lack of a reference model and prior knowledge of the G-code file under test, we consider any deviation from default configuration patterns within the file as potentially malicious. Table 3 illustrates the relationship between slicing parameters, their corresponding G-code commands, and the impact of manipulating these commands on the printed object, supported by relevant literature.

While simple configuration changes can indicate malicious intent, filament attacks present a more complex challenge. These attacks exploit the intricate relationships between various G-code commands to compromise the integrity of the printed object. The cavity attack, for example, leverages specific command sequences to create voids within the printed object. When these sequences are used in an unexpected manner, they can indicate potential voids, a malicious aspect demonstrated by Haris et al. [22]. However, by modifying parameters such as the filament length *"E"* in movement or filament update commands, this attack produces distinctive patterns, as detailed in Section 2.3. On the other hand, the filament speed attack targets the relationship between movement commands and material extrusion within the infill section. Normally, the infill has a repetitive structure where the amount of extruded filament remains consistent for equal movement distances. Therefore, any changes to the filament density that compromise this consistency are considered potentially malicious. Another type of manipulation, the filament state attack, is characterized by setting consecutive extrusion *"E"* parameters to the same value or removing this parameter from certain movement commands. These patterns serve as indicators of potential filament state attacks, which can disrupt the consistent extrusion of material during printing.

Generated G-code datasets

We aim to identify optimal dataset characteristics for training machine learning models to detect G-code attacks with high accuracy. To achieve this, we focus on design diversity and slicing parameter variability as key factors in dataset construction. We generated seven distinct G-code datasets for training, each comprising 70% of the total 600 generated files. These datasets include G-code files generated with various designs, print profiles, infill patterns, densities, and directions. The remaining 30% (180 files) serves as a common test set across all experiments, acting as a standardized benchmark that allows us to fairly evaluate the impact of different training datasets on the model's performance. By keeping the model architecture constant and varying only the training data, we can determine the level of design and slicing parameter diversity needed to detect new patterns of G-code attacks in unseen real-world objects. Notably, each design in the test set is sliced once with a unique combination of slicing parameters, resulting in a highly diverse test dataset that sufficiently challenges the ML model during evaluation.

Both the training and testing partitions are divided into 80% benign ($\mathcal{B}$) and 20% malicious classes, including cavity ($\mathcal{CV}$), filament speed ($\mathcal{FS}$), filament state ($\mathcal{FT}$), nozzle temperature ($\mathcal{T}_n$), bed temperature ($\mathcal{T}_b$), fan speed ($\mathcal{F}_s$), and Z-profile ($\mathcal{Z}_p$) attacks. This 80/20 split is deliberately chosen to reflect a realistic scenario where benign operations are more common than malicious ones. By training on this skewed distribution, we aim to create a model that can reliably distinguish between normal and potentially malicious files, better preparing it for real-world deployment where malicious G-code files are relatively rare.

It is essential to mention that each G-code file contains a large number of layers, providing sufficient data for training and testing ML models. The number of layers (K) varies based on the object height (Z_{obj}) and layer thickness (L_{th}), as calculated by Eq. (1). This variability in layer count, combined with the fact that attacks may target different numbers of layers within a file, results in the diverse class distribution observed in our training (DS) and testing (Test) datasets, as shown in Table 4.

$$K = \left\lceil \frac{Z_{obj}}{L_{th}} \right\rceil \tag{1}$$

To create a robust and representative dataset, we manually manipulate specific commands in each layer following the strategies discussed in Section 2.3. We introduce attacks of varying magnitudes and types, including creating cavities of different sizes, modifying filament density to various ratios, muting filament with a varying number of commands, and adjusting thermodynamic settings to different degrees deviating from the default configurations. This approach ensures a diverse dataset that captures a wide range of potential attack scenarios. Furthermore, to enhance the reliability of our evaluation process, we maintain complete distinctiveness in combinations of design, print profile, infill pattern, density, direction, and attack magnitude. We also ensure no overlap between training and testing layers across these combinations.

Table 4
Layer-level class distribution for the training and testing datasets.

Dataset	Class distribution							
	$\mathcal{B}$	$\mathcal{CV}$	$\mathcal{FS}$	$\mathcal{FT}$	$\mathcal{T}_n$	$\mathcal{T}_b$	$\mathcal{F}_s$	$\mathcal{Z}_p$
DS_1	9139	247	195	198	245	228	240	240
DS_2	9139	247	195	198	245	228	240	240
DS_3	9139	478	418	408	449	423	449	447
DS_4	13 858	478	418	408	449	423	449	447
DS_5	9139	220	227	182	258	400	294	256
DS_6	13 858	220	227	182	258	400	294	256
DS_{6p}	13 858	220	227	182	258	400	294	256
Test	7374	241	241	337	182	255	171	342

For the test dataset specifically, we incorporate subtle manipulations designed to rigorously evaluate our detection approach. These include a minimum number of infill lines impacted by the filament attacks, created cavities with small sizes, subtle changes in the filament densities, minor thermodynamic adjustments, such as ±12° variations in nozzle temperature, and ±2%, ±3%, and ±4% changes in the fan speed in addition to nuanced adjustments in the layer thickness, such as 0.05 mm. 0.1 mm, 0.12 mm, etc. These subtle changes are particularly challenging to detect as they do not cause visible deformations while significantly impacting the mechanical properties of printed objects, as discussed later in Section 6.1.

The datasets, DS_1, DS_2, and DS_3, are generated using simple geometric designs, such as rectangular bars and standard ASTM D790 and ASTM D638 Type IV specimens, which are widely used in existing research [15,22]. These designs are illustrated in Figs. 11(a), 11(b), and 11(c). While benign G-code samples are generated by varying all slicing parameters across these datasets, the malicious samples differ in their level of slicing diversity. DS_1 employs only the *Fast* profile (Table 2) and *Lines* infill pattern, DS_2 includes all infill patterns with the *Fast* profile, and DS_3 incorporates all profiles and infill patterns. The entire range of infill density and direction is utilized. The objective is to investigate whether using standard designs with specific slicing parameters, as employed in existing studies, is sufficient for detecting attacks on more diverse, unseen designs and to explore how diversifying the slicing parameters impacts the efficacy of attack detection.

DS_4, DS_5, and DS_6 are generated to study the impact of diversifying benign and malicious designs on detection accuracy. A set of ten unique designs, categorized as simple, medium, or complex based on their geometrical structure and detail, is shown in Fig. 11. We use these designs to generate the benign G-code samples of DS_4 while keeping the malicious samples from DS_3. Conversely, we use them to generate the malicious G-code samples of DS_5 while maintaining the benign samples from DS_3. In DS_6, we use these designs to generate both benign and malicious samples. The G-code files are generated by varying all of the slicing parameters. However, DS_{6p} differs from DS_6 by considering only 70% of the total infill patterns, density, and direction in both benign and malicious samples, aiming to understand how the diversity of infill characteristics affects detection performance and dataset completeness.

5.2. Feature engineering

This section conducts a formal analysis to identify and extract features crucial for detecting nuanced malicious manipulations within G-code files.

Formal analysis

G-code attacks target different commands, so both the command type and number are considered essential features. The command type (C_t) has 0 for G-commands and 1 for M-commands, while the command number (C_n) has nine distinct values, which are 0 for G0, 1 for G1, 92 for G92, 104 for M104, 140 for M140, 109 for M109, 190 for M190, 106 for M106, and 107 for M107. Moreover, the *"S"* parameter of the M-command represents either the nozzle temperature (S_n), bed temperature (S_b), or fan speed (S_f), all of which represent crucial features. Since the Z-profile attack targets the *"Z"* parameter of G0 commands, its absolute value (Z_v) is also incorporated. Given the relation between the nozzle temperature and the layer thickness (L_{th}) (Table 2), we include it as a feature. The layer thickness represents the change in Z values (current Z and previous one Z_p) and is calculated as $L_{th} = Z - Z_p$.

Furthermore, we analyzed benign samples and found that the nozzle temperature and fan speed varied across layers. Fan speed typically starts at 0° with the first layer, increases to 85° and 170° with the second and third layers, respectively, remains at 255° with subsequent layers, and then reduces to 0° at the end of the last layer. While the nozzle temperature is set differently with the first layer and remains consistent with subsequent layers (Table 2). Therefore, we add two more features called the layer number (L_n) and layer indicator (L_{in}). However, $L_n \in [0, K]$, while L_{in} has 0, 1, 2 with the first three layers, 3 with intermediate layers, and 4 with the last layer. The number of layers (K) varies across the files and is calculated using Eq. (1).

Since filament attacks primarily affect the extruded filament amount, we add it as an essential feature calculated as $\Delta E = E - E_p$, where E is the current filament length, and E_p is the preceding one. Considering the behavior of cavity and filament density variation attacks (Section 2.3), we include the movement distance (d) and movement direction angle (Θ) as additional features, calculated using Eqs. (2) and (3), respectively.

$$d = \sqrt{(X - X_p)^2 + (Y - Y_p)^2} \tag{2}$$

where (X, Y) is the current location and (X_p, Y_p) is the previous location of the print head.

$$\Theta^\circ = \arctan\left(\frac{Y - Y_p}{X - X_p}\right) \times \frac{180^\circ}{\pi} \tag{3}$$

As a result, each G-code command is represented by a set of features (C_t, C_n, d, Θ, ΔE, S_n, S_b, S_f, L_{th}, Z_v, L_n, and L_{in}). For G-commands, S_n, S_b, and S_f are the most recent M-command values. If a G-command lacks a Z parameter, it uses the last updated Z value. Layer thickness is calculated and updated for each new Z value. M-commands represent different features, with each command having values for C_t, C_n, L_n, L_{in}, and either S_n, S_b, or S_f, while other features are zeros.

To demonstrate the importance of d, Θ, and ΔE features and how they are manipulated by filament attacks, we analyze random samples of filament cavity and filament density attacks. Fig. 14 (left) highlights the original command targeted by the cavity attack in red. Fig. 14 (right) illustrates the additional inserted G1 commands. These commands represent the three sub-segments resulting from dividing the movement distance determined by the original two commands. The cavity is created by removing the *"E"* parameter of the second G1 command, preceded by a filament retraction of $r = 4$ mm and followed by a filament push of $p = 4$ mm. However, $u = 0.025226$ mm represents the muted filament amount at the cavity location. Notably, the values of r, p, and u are calculated as $E - E_p$. Moreover, the attack may segment the target distance into a different number of segments with varying cavity sizes.

Fig. 15 visualizes the attack's impact on the targeted movement distance. The total distance $d = 8.06102$ mm is divided into three equal 2.687006 mm sub-segments. The initial segment has $\Delta E = 0.02532$ mm, the middle segment has muted filament by $u = 0.025226$ mm, while the last segment has $\Delta E = 0.025274$ mm. Summing these values equals

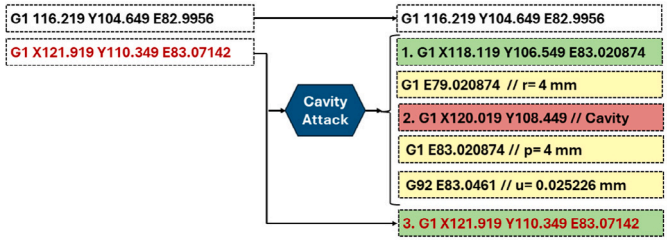

Fig. 14. Cavity attack footprint in G-code.

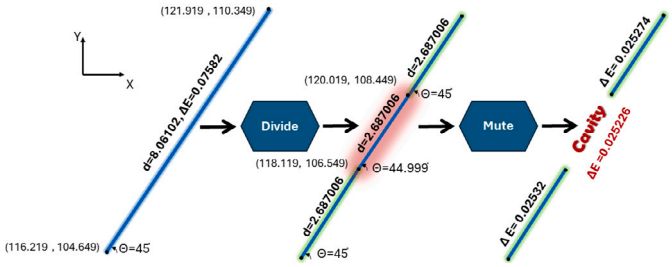

Fig. 15. Cavity attack impact on the movement command.

the original filament amount before the attack: 0.02532 + 0.025226 + 0.025274 = 0.07582 mm. This attack also subtly changes the direction angle by a minimal increment of 0.001° while segmenting the targeted command. We examined various samples and determined that this attack can adjust it to a maximum limit of 0.03°.

The filament speed attack exploits recurring patterns in infill structures. It leverages the relation between d and ΔE features to identify the most frequent filament amount (ΔE^*) for a given d. For instance, Fig. 16 illustrates the impact of this attack on a targeted distance $d = 8.0610173$ with $\Delta E^* = 0.10774$. The left part highlights the targeted command in red and the compensating commands in green. In contrast, the right part shows the new filament amounts assigned to the targeted and compensating commands, calculated using Eq. (4). The variations in filament amount are denoted as $G92_{\Delta E}$, and $G92'_{\Delta E}$, while the manipulated filament amount is denoted as ΔE_m.

$$\Delta E_m = \begin{cases} \Delta E^* - G92_{\Delta E} & ;\text{targeted command} \\ \Delta E^* + |G92'_{\Delta E}| & ;\text{compensating command} \end{cases} \tag{4}$$

The original filament amount for the targeted command is decreased by $G92_{\Delta E}$, from 0.10774 to 0.06063, a reduction of 0.04711. Conversely, it is increased for the compensating commands by $G92'\Delta E$ amounts of 0.002355 and 0.02356. As a result, the new E parameter of the targeted command is smaller than the original, whereas it is larger for the compensating commands. The E value of the $G92$ command matches the actual E of the original commands, leading to a positive $G92\Delta E$ for the target and negative $G92'_{\Delta E}$ for the compensating commands. However, this attack does not alter the object's weight; thus, modifications across each layer must satisfy Eq. (5). It is important to note that the number of targeted and compensating commands may vary across layers with different changes in the target filament amounts.

$$G92_{\Delta E} = \sum_{i=1}^{n'} |G92'_{\Delta E_i}| \tag{5}$$

where n' is the number of compensating commands.

Although the filament state attack focuses on manipulating only the ΔE feature by setting consecutive *"E"* parameters to the same value or removing this parameter from certain movement commands, it may overlap with the filament speed attack's footprint. To demonstrate this,

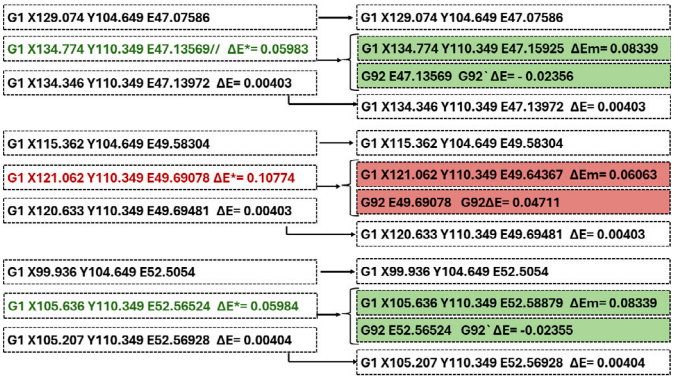

Fig. 16. Filament speed attack footprint in G-code.

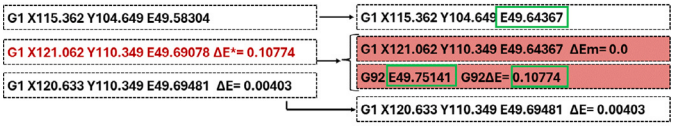

Fig. 17. Filament state attack footprint in G-code.

we consider Eq. (4). The filament state attack results in $G92_{\Delta E} = \Delta E^*$, leading to:

$$\Delta E_m = \begin{cases} \Delta E^* - \Delta E^* = 0 & ;\text{targeted command} \end{cases} \quad (6)$$

In this scenario, the sequence of commands after modifications appear as follows: The *"E"* value of the G92 command is adjusted to make $G92_{\Delta E} = \Delta E^* = 0.10774$, effectively setting ΔE_m to zero. The *"E"* value of the targeted command remains as modified by the filament density attack. The *"E"* value of the preceding command is changed to match the *"E"* value of the targeted command, resulting in $\Delta E_m = 0$. Meanwhile, the compensating commands remain as changed by the filament speed attack.

Fig. 17 illustrates these modifications, highlighted in a green box, demonstrating how the footprints of these attacks can overlap. Consequently, when considering only the red section, the pattern may be classified as a filament state attack. However, when taking into account all sections, including the compensating commands, the pattern aligns with the characteristics of the filament speed attack. This overlap emphasizes the complexity of distinguishing between these attacks and highlights the importance of comprehensive analysis in detecting and classifying such manipulations.

Feature selection

To identify the optimal set of features to represent G-code, we employ a wrapper-based model evaluation approach [30], as illustrated in Fig. 18. This method leverages the performance of the machine learning model itself to guide the feature selection process. In our study, we use the Bi-LSTM model's performance as the induction algorithm. The wrapper method systematically evaluates various combinations of features by assessing the Bi-LSTM model's performance on the test dataset. This iterative process allows us to explore the impact of different feature subsets on detection accuracy. [77].

The feature selection process starts with a core set of features, including C_t, C_n, d, Θ, ΔE, S_n, S_b, and S_f, which are essential for representing G-code commands and are included by default. For other features, such as layer number (L_n), layer indicator (L_{in}), layer thickness (L_{th}), and Z-value (Z_v), their necessity is less certain. Therefore, we iteratively exclude one of these additional features and evaluate the Bi-LSTM's performance on the test dataset. Our investigation particularly focuses on how these features relate to the model's ability

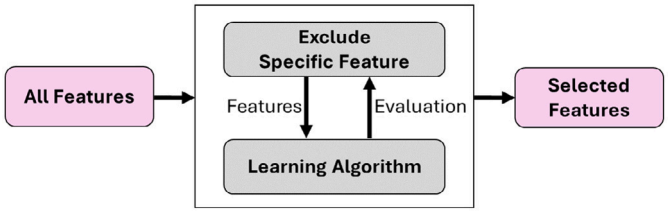

Fig. 18. Wrapper-based Feature Selection.

to recognize thermodynamic and Z-profile attacks. For instance, we explore whether excluding the layer thickness feature, which is related to both nozzle temperature and Z-value, affects the model's ability to detect nozzle temperature and Z-profile attacks. Similarly, we assess the impact of excluding layer number and layer indicator features on detecting attacks that target parameters varying with layer progression, such as nozzle temperature and fan speed attacks. This approach helps us identify the optimal feature subset that enhances the model's ability to detect G-code attacks accurately.

For command-level classification, we observe that commands within the same layer often share identical values for several features, including S_n, S_b, S_f, L_{th}, Z_v, L_n, and L_{in}. This similarity could lead to redundant feature vectors. To address this and ensure dataset diversity, we incorporate all extracted features when representing commands. The positional information of commands is also crucial in our analysis. Temperature adjustments, fan speed changes, and bed leveling typically occur at specific points, such as the beginning or end of a layer. However, the location of these control commands varies depending on the layer length within the G-code file. To capture this important spatial context, we introduce two additional features: Command index (C_{idx}) representing the command's location within the layer, and command indicator (C_{in}), specifying whether the command is the first, an intermediate, or the last in its layer.

5.3. Labeling strategies

Considering challenge (C4) and attack categories, we propose two labeling strategies: layer labeling and command labeling.

Layer labeling

To address attacks spanning multiple commands, such as filament attacks, we adopt a layer-level labeling strategy based on the Multiple Instance Learning (MIL) representation paradigm [31]. In MIL, each sample (named a bag) contains a variable number of unique instances, each characterized by feature vectors. In the context of G-code files, we consider each layer as a bag and each command within that layer as an instance. The layer's label is determined by the presence and nature of malicious commands it contains. If a layer has a control command changing the default temperatures, fan speed, or Z-value, it is considered a malicious layer with a specific class based on the type of change. For layers affected by changes to thermodynamic settings not explicitly present in the commands, we identify default patterns of such settings and label the layer based on the manipulated features. For example, if the fan speed is set to 100% before a layer begins, the entire layer is labeled as an $\mathcal{F}_s$ attack due to the affected S_f feature of movement commands within that layer. However, in practice, layers can be affected by multiple attacks simultaneously, which is beyond the scope of this study.

Command labeling

For command-level classification, we focus on attacks that target individual commands, such as thermodynamic and Z-profile attacks. These attacks typically impact one of the following features: S_n, S_b, S_f, L_{th}, and Z_v. Therefore, each movement command is classified into one

of four specific attack types ($\mathcal{T}_n$, $\mathcal{T}_b$, $\mathcal{F}_s$, $\mathcal{Z}_p$) based on the manipulated feature. However, any deviation from the default configuration patterns is considered potentially malicious, and the specific class is identified based on the targeted feature. For example, consider a scenario where all movement commands within a specific layer of an object are expected to have a nozzle temperature of 210°. If we encounter a command with a different temperature (e.g., 200°), we label it as a nozzle temperature attack ($\mathcal{T}_n$). Conversely, commands that adhere to the expected temperature are labeled as benign ($\mathcal{B}$).

5.4. ML model architecture and configuration

To classify G-code layers, we employ Bi-LSTM, a variant of Recurrent Neural Networks (RNNs), due to its ability to handle sequential dependencies among G-code commands, complex feature relationships, and the nuanced nature of attacks. By leveraging both past and future context [78,79], Bi-LSTM can capture subtle malicious patterns that span multiple commands efficiently. The LSTM unit, which represents the core of this model, comprises three gating components: input gate, forget gate, and output gate, each crucial for updating and preserving information over time. The input gate (i_t) controls the information flow that enters the memory cell. The forget gate (f_t), utilizing a sigmoid activation function, determines which information to retain or discard, significantly impacting the network's memory and controlling the information from the previous cell state c_{t-1}. However, the cell state (c_t) is updated based on the last state cell (c_{t-1}), the effects of the input and forget gates, and the candidate cell state ($\tilde{c}_t$). Finally, the output gate (o_t) controls the information flow from the cell state to the output [34]. The mathematical operations for each gate and the cell state update are:

$$\begin{aligned} i_t &= \sigma(W_i \odot [h_{t-1}, x_t] + b_i) \\ f_t &= \sigma(W_f \odot [h_{t-1}, x_t] + b_f) \\ \tilde{c}_t &= \tanh(W_c \odot [h_{t-1}, x_t] + b_c) \\ c_t &= f_t * c_{t-1} + i_t * \tilde{c}_t \\ o_t &= \sigma(W_o \odot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(c_t) \end{aligned} \tag{7}$$

Here, $\odot$ denotes the element-wise multiplication, W and b are learnable parameters, σ is the sigmoid function, and tanh is the hyperbolic tangent function. The LSTM architecture processes a sequence of inputs $x_1, x_2, \ldots, x_t$ and generates a sequence of hidden states $h_1, h_2, \ldots, h_t$. Each hidden state h_t is computed as a function of the corresponding input x_t, the previous hidden state h_{t-1}, and the current cell state c_t.

In our proposed approach, each G-code layer is represented as a sequence of feature vectors, with each vector representing a single command. Specifically, for a layer with n commands, we define a sequence of feature vectors f_j, where $j \in 1, 2, \ldots, n$. Each feature vector f_j is defined as $f_j = [C_t, C_n, d, \Theta, \Delta E, S_n, S_b, S_f, L_{th}, Z_v, L_n, L_{in}]$. To ensure uniform input dimensions for our model, we pad these sequences to a length of $M = \max(n)$ commands across all layers in the dataset. These padded sequences serve as input to the stacked Bi-LSTM hidden layers, as illustrated in Fig. 19. Each layer comprises multiple LSTM units that manage the information flows through input, output, and forget gates. These Bi-LSTM layers facilitate the hierarchical encoding of instances through an instance-level transformation function $g_\psi(f_j)$. This function maps the command feature vector (f_j) to an instance-level representation (h_j), using learnable parameters ψ.

To enhance the model's capability to handle variable-length input sequences and capture long-term dependencies, we incorporate an attention mechanism into the Bi-LSTM model. The attention mechanism gives a different focus to the information generated from the hidden layers of Bi-LSTM during prediction [80]. It helps mitigate the vanishing gradient problem and provides insights into the contribution of each G-code command to the final layer classification. Consequently, each encoded vector h_j is assigned a weight (β_j), indicating its importance, calculated using learnable attention parameters w and V:

$$\beta_j = \frac{\exp(w^T \tanh(V h_j^T))}{\sum_{j=1}^{M} \exp(w^T \tanh(V h_j^T))} \tag{8}$$

The layer-level representation χ is then obtained as the weighted sum of the instance-level representations:

$$\chi = \sum_{j=1}^{M} \beta_j h_j \tag{9}$$

This representation then serves as input to a classifier function $\phi(\chi)$, implemented using a softmax activation function, to get the predicted class probabilities p_c for a given G-code layer:

$$p_c = \phi(\chi) = \frac{\exp(w_c^T \chi + b_c)}{\sum_{i=1}^{C} \exp(w_i^T \chi + b_i)} \tag{10}$$

where C is the number of classes, while w_c and b_c are the weight vector and bias term for class c, respectively.

During training, the model learns the abovementioned parameters, such as ψ, w, V, and classifier weights by minimizing the log-likelihood loss between the predicted class probabilities p and the actual label y using gradient-based optimization. Finally, and to address the class imbalance in the dataset, we employ the focal loss function, which focuses on hard-to-classify examples while training Bi-LSTM [35]:

$$\mathcal{L}_{focal} = -\sum_{c=1}^{C} \alpha_c (1 - p_c)^\gamma y_c \log(p_c) \tag{11}$$

where γ adjusts the focus on hard-to-classify examples, and α_c is a weighting factor for a class c that is inversely proportional to the class frequency (N_c) and calculated as $\alpha_c = \frac{N}{N_c}$; N is the number of instances in the dataset.

The imbalance in our dataset arises from the varying number of layers across different objects (see Eq. (1)) and the fact that attacks may target different layers. Although various techniques, such as data augmentation, oversampling, and undersampling, can be employed to address this imbalance, our primary objective is to maintain the realism of the G-code dataset. Consequently, we avoid these techniques as they can introduce bias or result in the loss of valuable information [81,82].

For the command-level classification of thermodynamic and Z-profile attacks, we employ Random Forest (RF) and Multilayer Perceptron (MLP) algorithms [36,37]. These techniques are chosen for their ability to process high-dimensional feature spaces and capture non-linear relationships, which is crucial for detecting subtle variations in G-code commands. The RF algorithm's ensemble nature provides robustness against overfitting and offers built-in feature importance measures, enhancing interpretability. Meanwhile, the adaptable neural network architecture of the MLP model can capture nuanced changes within G-code command features. Both classifiers proved their well-established performance in similar classification tasks, such as detecting malicious adjustments to the design and layer thickness, considering the features extracted from the side-channel vibration signals [49].

While other classification algorithms are suitable for this task, our primary objective is to improve the detection of subtle thermodynamic and Z-profile changes. Therefore, this finer-grained classification approach aims to overcome the potential limitations of the attention-based MIL in specific scenarios, complementing our layer-based detection strategy and providing a more comprehensive framework for identifying diverse G-code attacks.

6. Experiments and discussion

This section presents the experimental comparative baseline, setup, evaluation metrics, and optimal model configuration. It also addresses the research questions in this study through empirical analysis, identifying the optimal dataset characteristics, feature set, and classification model for both layer and command-level detection of G-code attacks.

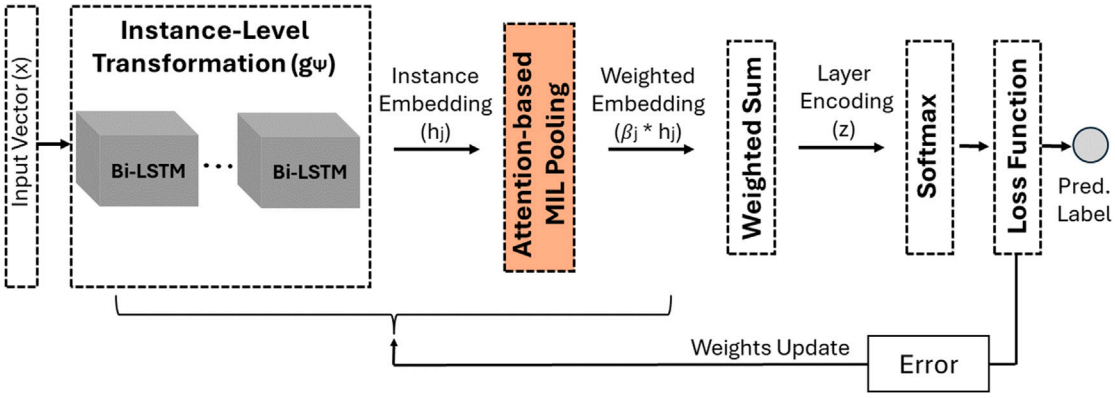

Fig. 19. Bi-LSTM model architecture.

6.1. Comparative baseline

The comparative baseline is established through documented physical impacts and mechanical testing results. The 3D printing workflow inherently spans two distinct domains, cyber and physical. In the cyber domain, where G-code is generated and potentially manipulated before printing, only three previous studies have addressed G-code protection through hashing and encryption methods. These existing approaches aim to prevent modifications rather than detect or classify them. Thus, experimental comparison with them is inappropriate due to the main differences in approach (detection vs. prevention). Therefore, we provided a theoretical comparison, as shown in Table 1, to highlight the key differences and advantages of our approach. This comparison demonstrates that while existing methods focus on integrity checking and require original files or third-party involvement, our approach provides deeper insights by identifying the specific nature and location of modifications, operating without requiring reference models, and providing a classification of attack types.

In the physical domain, numerous methodologies have been proposed for G-code attack detection, encompassing side-channel analysis, real-time print monitoring, and post-production mechanical testing. As the attacks examined in this study were identified through visual deformation analysis and mechanical property deviations, we establish our validation baseline by leveraging the intrinsic relationships between G-code features and both physical observations and mechanical properties documented in the existing research findings. Our test dataset comprises 180 unique designs, specifically chosen to represent a range of geometric complexities from simple shapes to complex structures with varying layer characteristics. 80% of the testing files were generated without any manipulations, and 20% of them incorporated carefully implemented manipulations based on documented attack patterns from previous research. For example, we manipulated the default nozzle temperature with varying values that could be small, impacting the mechanical strain and stress of the object, or large, causing observable defects such as material oozing. Additionally, we implemented filament attacks with multiple variations that demonstrate measurable impacts on the mechanical properties of the object through tensile and three-point bending tests [17,23].

For high-magnitude attacks that produce visible defects, we establish our baseline through physical printing verification and visual validation using the *Zupfe GCode Viewer* to verify attack implementations. Such impacts include the gaps created by the filament state attack, as demonstrated later in Fig. 30 with the printed face shield headband and in Fig. 31 with the printed drone propeller [14]. Further experimental validation is presented in Fig. 32, which shows the material melting and oozing that result from disabling the cooling fan during the printing of the drone propeller. Moreover, Fig. 20(a) shows visible impacts of filament cavity attacks across multiple infill lines

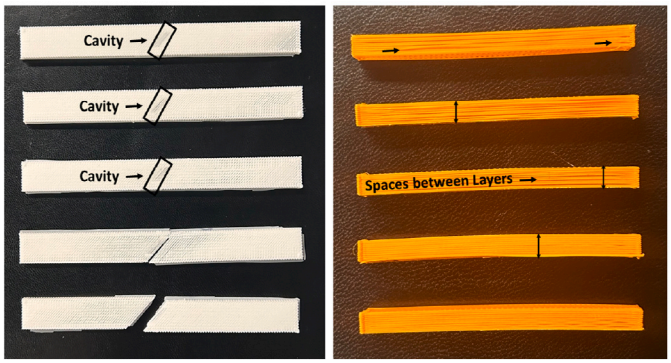

Fig. 20. Visual impacts of filament cavity and Z-Profile attacks.

with various cavity sizes (e.g., 0.2 mm, 0.3 mm, 0.4 mm), significantly compromising structural integrity, causing specimens to fracture at their central region. Fig. 20(b) shows the impact of the Z-Profile attack on the space between the layers. Samples impacted by filament attacks were also visualized earlier in Figs. 5 and 6 using the Zupfe GCode Viewer. In addition to the defective objects presented in this study, previous research by Gao et al. [17] illustrated physical deformations caused by changes in fan speed, print speed, and filament extrusion, while Si et al. [23] demonstrated how fan speed adjustments affect the surface roughness of the objects.

For subtle manipulations that do not cause visible deformations, our baseline relies on analyzing the mechanical testing results. Using the MTS Insight 30 machine, we conducted tensile tests on PLA-printed regular bars to demonstrate how minor parameter changes affect the mechanical properties of the object. We tested specimens with modifications in nozzle temperature (±12°), fan speed (±4%), and layer thickness (±0.2 mm). The benign specimens established baseline properties with an average peak stress of 23.63 MPa and strain of 0.026 mm/mm. Higher nozzle temperature increased the peak stress to 28.2 MPa due to enhanced layer fusion while maintaining a similar strain (0.0256 mm/mm). Conversely, lower temperature reduced peak stress to 20.633 MPa with comparable strain (0.025 mm/mm). Fan speed modifications also produced notable changes. Higher fan speed resulted in 24.033 MPa peak stress with 0.029 mm/mm strain, while lower fan speed led to increased stress of 28.1 MPa, attributable to slower cooling, enabling better layer fusion. The Z-profile attack modifying layer thickness produced the most distinctive mechanical behavior. While peak stress decreased slightly to 22.3 MPa, strain increased substantially to 0.0506 mm/mm. As illustrated in Fig. 21, specimens exhibited clear layer separation during testing, with the

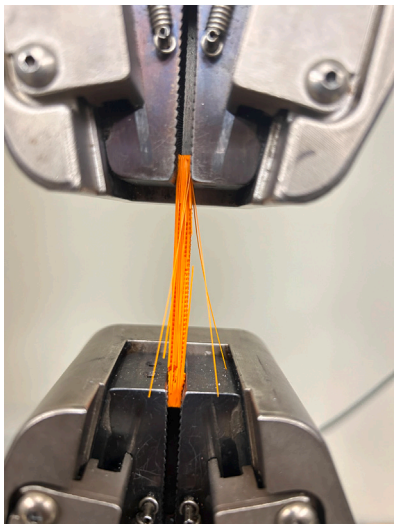

Fig. 21. Tensile test for a sample impacted by Z-profile attack.

layers delaminating cleanly rather than breaking, indicating weakened interlayer bonding while maintaining structural integrity within each layer.

Existing work [22] also provided a comprehensive analysis of the mechanical properties of objects impacted by filament and thermodynamic attacks. The authors demonstrated how creating cavities with various sizes, changing the density of the filament, and modifying the filament extrusion resulted in premature structural failure at attack locations while impacting the strain and stress characteristics of the object. Their work further illustrated the impact of subtle changes in the nozzle temperature on the residual thermal stress and strain. In a subsequent study [15], the same authors expanded their findings through mechanical testing of samples impacted by Z-profile attacks that modified the layer thickness with various magnitudes ranging from 0.05 mm to 0.2 mm.

This comprehensive validation approach, combining visual verification, simulator analysis, and mechanical testing results, provides a robust baseline against which we can evaluate our detection method. The correlation between our detection results and these established physical impacts demonstrates the practical reliability of our approach in identifying both visible and subtle manipulations before printing occurs, preventing material waste and potential safety issues.

6.2. Experimental setup

We conducted our experiments on a GNU/Linux cluster equipped with 26 nodes of NVIDIA V100, A100, and H100 GPUs and 38 TB of RAM. Our environment was built using TensorFlow 2.15.0, CUDA 12.2, CUDNN 8.9.2, and Python 3.9.18. Although we leveraged this high-performance setup, a detailed analysis of the model's time complexity and computational efficiency is beyond the scope of this study.

6.3. Evaluation metrics

In the context of multi-class imbalanced datasets, overall accuracy is an insufficient metric for assessing classifier performance [83]. In such scenarios, the majority class (benign) impacts the model's performance more than the minority classes (malicious), skewing the accuracy metric. To address this, we focus on evaluating the classifier's performance for each class within the dataset, using precision ($Prec$), recall (Rec), and F1-score ($F1$) metrics [84]. We primarily use the F1 measure for result discussion and comparison. These metrics are defined as follows:

- **Precision**: The ratio of correctly predicted positive instances to the total predicted positives.
- **Recall** (or Sensitivity): The ratio of correctly predicted positive instances to all actual positive instances.
- **F1-score**: The harmonic mean of precision and recall, offering a balanced measure beneficial for uneven class distributions.

To facilitate comprehensive comparisons and gain insights into overall model performance, we also calculate these metrics across all classes [84]. This approach enables us to evaluate Bi-LSTM's overall performance across different configurations to determine the optimal setup and with various feature sets to identify the optimal one. Furthermore, it provides a clearer basis for comparing Bi-LSTM performance against other RNN models (i.e., LSTM and GRU) and against RF for thermodynamic and Z-profile attack detection. However, we employ macro-averaging for these metrics. Each metric is calculated independently for each class before averaging them, ensuring that each class contributes equally to the overall metric. The formulas for the macro-averaged metrics are [85]:

$$Prec_M = \frac{\sum_i \frac{TP_i}{TP_i+FP_i}}{C} \; Rec_M = \frac{\sum_i \frac{TP_i}{TP_i+FN_i}}{C}$$

$$F1_M = \frac{2}{1/Prec_M + 1/Rec_M}$$

Here, TP_i, TN_i, and FP_i represent the True Positives, True Negatives, and False Positives for the ith class, while C indicates the total number of classes.

6.4. Identify the model configuration

Our experimental analysis begins with determining the optimal Bi-LSTM architecture to be used in the subsequent experiments. However, we faced two main challenges which are the extensive layer lengths (up to 9865 commands in DS_1, DS_2, DS_3, and 9799 in other datasets) and extreme class imbalance. To address these challenges, we explored various architectural modifications beyond the attention mechanism, including changing the numbers of hidden layers and neurons, testing various optimizers, adjusting the number of epochs, and fine-tuning the γ value in the focal loss function. Our goal is to mitigate large gradients and instability while balancing model performance and complexity. The key results of these experiments are illustrated in Fig. 22.

To ensure a fair comparison, we conducted a set of experiments using fixed training and testing datasets while systematically varying Bi-LSTM model configurations. We selected the most diverse training dataset (DS_6) as our benchmark, evaluating each generated model against the same testing dataset that has been discussed in Section 5.1. It is crucial to highlight that the overall F1 score of the Bi-LSTM model is 0.759, which is primarily due to its limited effectiveness in detecting the Z-profile attack, as will be discussed later in this section. This specific attack type poses significant detection challenges that are effectively addressed by employing finer-grained classification using the RF algorithm.

Fig. 22(a) illustrates the overall performance of the model with different optimizers, namely Adam, RMSprop, and Nadam. Our findings demonstrate that the Nadam optimizer consistently outperforms its counterparts across all evaluation metrics. Furthermore, we investigated the impact of gradient clipping on these optimizers. However, we observed that without clipping, gradient values escalate dramatically during training, potentially destabilizing the learning process. This observation highlights the critical role of gradient management in maintaining model stability and convergence. Moreover, Fig. 22(b) illustrates that training the model for 200 epochs maximizes model performance while mitigating the risk of overfitting. This empirically determined epoch count ensures robust learning without compromising generalization capabilities. We also conducted an in-depth analysis of the focal loss function, exploring its behavior with (γ = 2, 3,

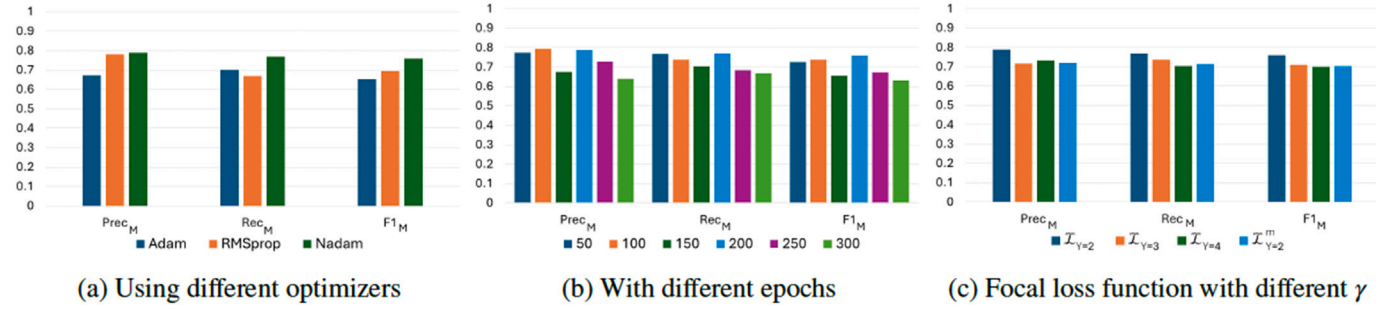

Fig. 22. Overall performance of Bi-LSTM with different configurations.

Table 5
Bi-LSTM model configuration.

Parameter	Value
Input Layer	Bidirectional LSTM (256 units)
Hidden Layers	3× Bidirectional LSTM (128 units each)
Output Layer	Dense (8 units) with Softmax activation function
Dropout Layer	30%
Optimizer	Nadam (Learning Rate: 0.001, Gradient Clip: 1.0)
Loss Function	Focal with γ=2
Batch Size	16
Epochs	200

4), as shown in Fig. 22(c). Additionally, we examined the impact of considering only the α values of the malicious classes ($\mathcal{L}^{m}_{focal}$) while maintaining the original contribution of the benign class. Our results show that reducing the benign's contribution while focusing more on the minor classes with ($\gamma = 2$) leads to the best overall performance across the metrics. However, we observed that using ($\gamma > 4$) leads to excessively large gradients, potentially destabilizing the optimization process. Based on these results, Table 5 shows the Bi-LSTM model configuration adopted in our experiments.

6.5. Identify the optimal dataset characteristics

Upon determining the optimal configuration for the Bi-LSTM model, we keep the model architecture constant while varying only the training data. By consistently evaluating the model against the same test dataset across all experiments, we establish a standardized benchmark to assess the impact of different training datasets on the model's performance fairly. This approach enables us to determine the level of design and slicing parameter diversity required to detect new patterns of G-code attacks in unseen real-world objects. Table 6 demonstrates that diversifying the slicing parameters with simple designs negatively impacts the detection of benign layers and reduces the distinction between filament attacks, as shown with DS_1, DS_2, and DS_3. It also illustrates that increasing the diversity of benign designs with $DS4$ improves the detection of the benign layers besides the thermodynamic attack samples compared to $DS5$. Furthermore, including malicious samples with simple designs (DS_4) reduces the misclassification of filament state attacks ($\mathcal{FT}$) as filament speed attacks ($\mathcal{FS}$), resulting in a higher F1 score compared to DS_6.

To better understand the complexity and diversity of designs, we utilize the *3DPEA G-code Simulator* [86] to illustrate sectional views of layers from different designs (Figs. 23 and 24). G-code files generated by slicing designs may have infill sections characterized by specific patterns. For instance, a *Cross* infill pattern is used with a rectangular bar (Fig. 23(a)), while a *Cubic* pattern is employed with an ASTM D638 Type IV specimen (Fig. 23(b)). More complex shapes are shown in Figs. 24(a) and 24(b), with Figs. 24(c) and 24(d) demonstrating how the same object can have different layer shapes.

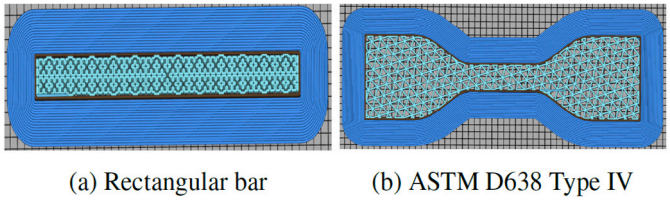

Fig. 23. Layer-level sectional view of the rectangular bar and ASTM D638 Type IV specimen.

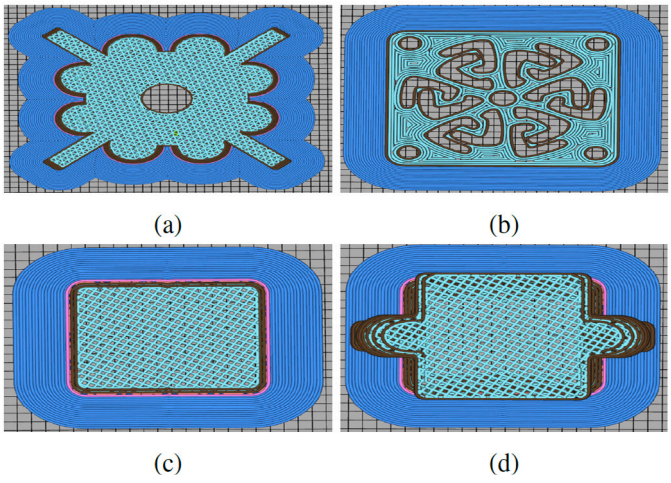

Fig. 24. Layer-level sectional view of different objects.

Typically, all the layers within the same G-code file share the same infill patterns, density, and direction. However, more intricate designs with detailed features or asymmetrical elements may introduce varied infill characteristics across the layers. This diversity in designs contributes to improved detection accuracy of the model. Our findings indicate that including diverse benign and malicious designs (DS_6) helps maintain a balance in detecting both benign and malicious layers, leading to improved performance across all classes. However, both DS_4 and DS_6 demonstrate strong performance, with DS_4 outperforming DS_6 by an average of 4.97%, excluding the $\mathcal{Z}_p$ class. On the other hand, limiting the value ranges of infill characteristics to 70% of the total in both benign and malicious samples reduces the model performance, as observed in DS_6p. This observation emphasizes the importance of considering all possible values of infill patterns, density, and direction for a comprehensive training dataset.

Table 6
Performance of Bi-LSTM with different training dataset characteristics.

Class	DS_1			DS_2			DS_3			DS_4			DS_5			DS_{6p}			DS_6		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
$\mathcal{B}$	0.957	0.955	0.956	0.949	0.872	0.909	0.943	0.564	0.706	0.947	0.980	**0.963**	0.960	0.403	0.567	0.948	0.829	0.884	0.954	0.837	0.891
$\mathcal{CV}$	0.968	1.000	**0.984**	0.764	0.928	0.838	0.183	0.978	0.309	0.960	0.934	0.947	0.937	0.978	0.957	0.909	0.994	0.949	0.848	0.989	0.913
$\mathcal{FS}$	0.548	0.937	0.691	0.848	0.296	0.439	0.738	0.328	0.454	1.000	0.550	0.709	0.533	0.989	0.693	0.526	0.841	0.648	0.859	0.746	**0.799**
$\mathcal{FT}$	1.000	0.387	0.558	0.650	0.664	0.657	0.589	0.840	0.693	0.739	0.954	**0.833**	0.811	0.181	0.296	0.621	0.227	0.332	0.992	0.521	0.683
$\mathcal{T}_n$	0.855	0.865	0.860	0.688	0.889	0.776	0.875	0.859	0.867	0.987	0.900	**0.942**	0.857	0.842	0.849	0.993	0.778	0.872	0.706	0.942	0.807
$\mathcal{T}_b$	0.979	0.957	0.968	0.845	0.937	0.888	0.761	0.973	0.854	0.949	0.945	0.947	0.948	1.000	**0.973**	0.891	0.992	0.939	0.914	1.000	0.955
$\mathcal{F}_s$	0.579	0.923	0.712	0.447	0.819	0.579	0.189	0.846	0.310	0.955	0.819	0.882	0.399	0.951	0.562	0.852	0.857	0.855	0.975	0.863	**0.915**
$\mathcal{Z}_p$	0.038	0.029	0.033	0.029	0.067	0.04	0.029	0.167	0.051	0.000	0.000	0.000	0.048	0.632	0.089	0.03	0.119	0.049	0.068	0.260	**0.107**

Table 7
Performance of Bi-LSTM with different feature sets.

Class	DS_{in}			DS_n			DS_{th}			DS_z			DS_{all}		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
$\mathcal{B}$	0.943	0.818	0.876	0.919	0.608	0.732	0.939	0.833	0.883	0.947	0.778	0.854	0.954	0.837	**0.891**
$\mathcal{CV}$	0.702	1.000	0.825	0.227	0.580	0.327	0.947	0.983	**0.965**	0.869	0.917	0.892	0.848	0.989	0.913
$\mathcal{FS}$	0.636	0.444	0.523	0.402	0.493	0.443	0.592	0.972	0.736	0.549	0.739	0.631	0.859	0.746	**0.799**
$\mathcal{FT}$	0.619	0.546	0.580	0.130	0.408	0.197	0.825	0.277	0.415	0.521	0.471	0.494	0.992	0.521	**0.683**
$\mathcal{T}_n$	0.773	0.737	0.754	0.958	0.661	0.782	0.903	0.819	0.859	0.937	0.871	**0.903**	0.706	0.942	0.807
$\mathcal{T}_b$	0.964	0.953	0.959	0.355	0.992	0.523	0.930	0.992	0.960	0.951	0.988	**0.969**	0.914	1.000	0.955
$\mathcal{F}_s$	0.919	0.874	0.896	0.762	0.846	0.802	0.935	0.868	0.900	0.929	0.934	**0.932**	0.975	0.863	0.915
$\mathcal{Z}_p$	0.036	0.146	0.058	0.036	0.178	0.060	0.028	0.105	0.044	0.035	0.169	0.058	0.068	0.260	**0.107**

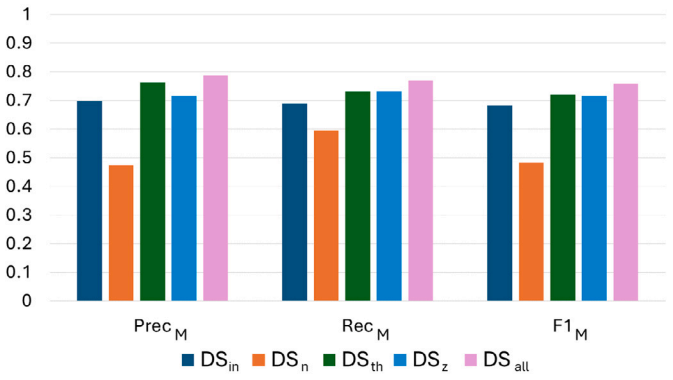

Fig. 25. Overall performance of Bi-LSTM with different feature sets.

6.6. Identify the optimal feature set

We conducted this experiment using the DS_6 dataset and five variations of it, each excluding a specific feature: DS_{in} (excluding layer indicator), DS_n (excluding layer number), DS_{th} (excluding layer thickness), DS_z (excluding Z-value), while DS_{all} considers all the features. Table 7 shows that incorporating all features significantly enhances the differentiation between benign and Z-profile attack samples. However, excluding the Z-value feature reduces this distinction but notably improves the detection of thermodynamic attacks. In contrast, excluding the layer thickness negatively impacts the detection of nozzle temperature and Z-profile attacks due to the strong correlation between layer thickness and these manipulated features, as shown in Table 2. Furthermore, the layer indicator and layer number features are essential for better detecting the nozzle temperature and fan speed attacks.

The overall performance of the Bi-LSTM model with the different feature sets is illustrated in Fig. 25. It demonstrates the highest impact of excluding the layer number on the overall performance of the model, followed by excluding the layer indicator. Moreover, it shows that excluding either the layer thickness or the Z-value has a comparable impact on the overall performance. Notably, the model achieves optimal performance across all metrics when no features are excluded, highlighting the importance of considering all the extracted features.

6.7. Evaluation of layer classification

This section validates the choice of the Bi-LSTM model for detecting G-code attacks at the layer level. Given the sequential dependencies between G-code commands, complex feature relationships, and intricate attack footprints, RNN models are well-suited for classifying the layers. The LSTM model effectively captures long-term dependencies between G-code commands while processing sequences in one direction, considering only the past and present data. However, it does not take into account future data points when making predictions. In contrast, the Bidirectional LSTM (Bi-LSTM) model processes G-code data in both forward and backward directions, providing a more comprehensive context for each point in the sequence. The Gated Recurrent Unit (GRU) model, introduced in 2014 [87], offers a simplified gated structure compared to LSTM. However, this model lacks separate memory cells and output gates, leading to full exposure of the unit's content [88]. While this simplification can be advantageous in some applications, it may limit GRU's capacity to capture the complex patterns present in G-code attacks.

Table 8 presents the performance comparison of LSTM, Bidirectional GRU (Bi-GRU) [89], and Bi-LSTM models across all classes. These models were trained on the $DS6$ dataset with identical configurations (Table 5), differing only in their recurrent unit types (i.e., LSTM, GRU). We observe that the cavity attack ($\mathcal{CV}$) is recognized effectively by all the models due to its distinctive footprint across G-code commands, as shown in Fig. 14. In contrast, the models struggle with distinguishing between filament speed ($\mathcal{FS}$) and filament state ($\mathcal{FT}$) attacks because of their overlapping patterns. An example of this overlap is formally demonstrated in Section 5.2, which illustrates how considering only the red section in Fig. 17 may cause the model to classify it as a filament state attack. However, incorporating the preceding and following commands (green sections) leads the model to capture the entire trace of the filament speed attack and distinguish it from the filament state attack. Therefore, the Bi-LSTM model gives a better F1 score for detecting these attacks compared to other models.

As demonstrated by Table 8, the Bi-LSTM model also exhibits superior performance in recognizing the thermodynamic attacks, including nozzle temperature ($\mathcal{T}_n$), bed temperature ($\mathcal{T}_b$), and fan speed ($\mathcal{F}_s$) attacks. These attacks alter multiple features such as C_t, C_n, S_n, S_b, and S_f, which are related to layer characteristics like L_{th}, L_n, and L_{in}. The widespread impact on various features results in a more distinct

Table 8
Performance comparison of the LSTM, Bi-GRU, and Bi-LSTM models.

Class	LSTM			Bi-GRU			Bi-LSTM		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
$\mathcal{B}$	0.953	0.651	0.774	0.939	0.624	0.749	0.954	0.837	**0.891**
$\mathcal{CV}$	0.884	0.884	0.884	0.929	0.807	0.864	0.848	0.989	**0.913**
$\mathcal{FS}$	0.457	0.979	0.623	0.447	0.624	0.521	0.859	0.746	**0.799**
$\mathcal{FT}$	0.515	0.071	0.125	0.168	0.101	0.126	0.992	0.521	**0.683**
$\mathcal{T}_n$	0.879	0.889	0.884	0.927	0.813	0.866	0.706	0.942	0.807
$\mathcal{T}_b$	0.962	0.996	0.979	0.947	0.988	0.967	0.914	1.000	0.955
$\mathcal{F}_s$	0.667	0.901	0.766	0.623	0.879	0.729	0.975	0.863	**0.915**
$\mathcal{Z}_p$	0.042	0.325	0.074	0.045	0.380	0.081	0.068	0.260	**0.107**

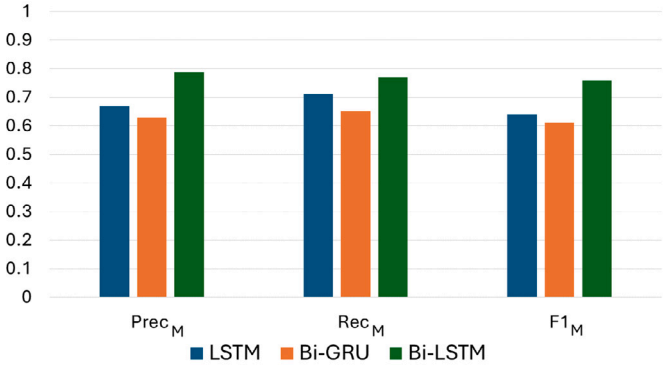

Fig. 26. Overall Performance Comparison of the LSTM, Bi-GRU, and Bi-LSTM Models.

footprint across the layer, enhancing the detection capability of the model for these attacks. Conversely, the Z-profile attack ($\mathcal{Z}_p$) changes only the Z parameter of movement commands, impacting the Z-value (Z_v) and, consequently, the layer thickness (L_{th}). However, with only two features affected, the Z-profile attack creates a minor footprint across the feature vectors within a layer, making it difficult for the model to recognize this attack accurately. These subtle changes require finer-grained classification to be detected using the RF algorithm.

Fig. 26 shows that while the Bi-GRU model also handles long-term dependencies in G-code sequences in both directions, it still underperforms compared to other models due to its simpler unit structure, which lacks the separate memory cells found in LSTM units. The memory cell enables the model to maintain information over longer sequences, a crucial feature for G-code attack detection, where attack patterns spread across multiple commands or different areas within a layer. As a result, LSTM slightly outperforms the GRU model, showing a 4.18% improvement in F1 score. More significantly, Bi-LSTM demonstrates a substantial performance gain over Bi-GRU, with a 23.81% increase in F1 score. This improvement highlights Bi-LSTM's superior ability to manage the complex interrelationships between G-code commands across entire layers.

To validate our choice of the focal loss function, we evaluate the Bi-LSTM model's performance using various techniques designed to address dataset imbalance. We compared the focal loss function ($\mathcal{L}_{focal}$) against the weighted cross-entropy loss function ($\mathcal{L}_{wce}$) [90] and the cost-sensitive learning (*Cost*) approach [84]. However, the standard categorical cross-entropy loss, typically used for multi-class classification problems, often performs poorly on imbalanced datasets. To address this, the weighted version of this function ($\mathcal{L}_{wce}$) assigns higher weights to minority classes, thereby increasing their importance during training as follows:

$$\mathcal{L}_{wce} = -\sum_{c=1}^{C} w_c y_c \log(p_c) \tag{12}$$

Here, w_c represents the weight of class c, typically inversely proportional to its frequency as illustrated in Table 4.

Table 9
Performance evaluation of Bi-LSTM with different class imbalance techniques.

Class	$\mathcal{L}_{wce}$			*Cost*			$\mathcal{L}_{focal}$		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
$\mathcal{B}$	0.957	0.639	0.767	0.948	0.974	**0.961**	0.954	0.837	0.891
$\mathcal{CV}$	0.871	0.934	0.901	0.923	1.000	**0.960**	0.848	0.989	0.913
$\mathcal{FS}$	0.372	0.677	0.480	0.671	0.757	0.711	0.859	0.746	**0.799**
$\mathcal{FT}$	0.386	0.092	0.149	0.734	0.546	0.627	0.992	0.521	**0.683**
$\mathcal{T}_n$	0.880	0.859	0.869	0.987	0.871	**0.925**	0.706	0.942	0.807
$\mathcal{T}_b$	0.996	0.988	0.992	0.981	0.992	**0.986**	0.914	1.000	0.955
$\mathcal{F}_s$	0.871	0.967	0.917	0.994	0.874	**0.929**	0.975	0.863	0.915
$\mathcal{Z}_p$	0.044	0.371	0.079	0.000	0.000	0.000	0.068	0.260	**0.107**

On the other hand, the *Cost* approach integrates class-dependent cost items into Bi-LSTM backpropagation learning, with each class's cost inversely proportional to its frequency (N_c) raised to a power λ. After exploring λ values from 0.1 to 0.9, we found that $\lambda = 0.3$ yields the best overall performance.

$$Cost(c) = \left(\frac{1}{N_c}\right)^{\lambda} \tag{13}$$

Table 9 illustrates that adopting the $\mathcal{L}_{wce}$ function enhances the detection of thermodynamic attacks but at the expense of recognizing the benign samples, which represent the majority class. Despite assigning high weights to the $\mathcal{FS}$, $\mathcal{FT}$, and $\mathcal{Z}p$ classes based on their frequencies in Table 4, this function does not significantly improve their detection. This limitation arises from $\mathcal{L}_{wce}$'s focus on class frequency rather than the classification complexity of such attacks, resulting in poor detection performance. In contrast, the *cost* approach improves the recognition of more frequent classes, including benign samples and thermodynamic attacks, at the expense of rarer ones, such as filament speed, filament state, and Z-profile attacks. However, this method addresses class imbalance without considering the classification difficulty of each example. Consequently, it performs poorly with complex attack types with fewer samples in the dataset, such as filament speed and filament state attacks. Compared to these approaches, the focal loss function ($\mathcal{L}_{focal}$) exhibits superior performance across all classes by simultaneously addressing class imbalance and classification difficulty. It focuses on hard-to-classify examples and dynamically adjusts the loss for each instance based on its classification complexity, which is particularly beneficial for subtle and intricate filament attacks.

Finally, to further validate the robustness of our chosen Bi-LSTM approach and address potential overfitting concerns, we conduct a comprehensive cross-validation analysis. We precisely divide the DS_6 dataset into five equal folds, and in each validation round, four of them are utilized for training while one fold is reserved for validation, ensuring comprehensive coverage of all data patterns. We ensure that G-code layers from the same file appear exclusively in either the training or validation folds. The results, as shown in Table 10, provide strong evidence against overfitting. Across different folds, the model maintains macro F1 scores ranging from 0.728 to 0.858, with corresponding precision (0.783–0.844) and recall (0.779–0.878) showing balanced performance. This consistency across different data splits is particularly noteworthy given the challenges of our multi-class classification task with imbalanced data distribution. Moreover, the variation between folds indicates that the model is learning generalizable patterns rather than memorizing specific instances. It is important to note that the relatively lower F1 scores result from the misclassification between benign and Z-profile attack samples, aligning with our previous findings in Table 8. This performance is expected given the subtle nature of Z-profile manipulations detected better by the command-level classification using the RF algorithm.

Table 10
5-Fold cross-validation results for the Bi-LSTM Model.

CV round	Overall $Prec_M$	Overall Rec_M	Overall $F1_M$
$Round_1$	0.833	0.840	0.818
$Round_2$	0.844	0.878	0.858
$Round_3$	0.843	0.855	0.841
$Round_4$	0.783	0.807	0.728
$Round_5$	0.787	0.779	0.732
$Round_{mean}$	0.818	0.83.2	0.795

6.8. Evaluation of command classification

These experiments aim to demonstrate the superiority of command-level classification over layer-level classification using Bi-LSTM in detecting subtle changes caused by thermodynamic and Z-profile attacks. In such scenarios, certain layers are affected without obvious malicious patterns within their commands (Challenge C4). To this end, we prepare an efficient dataset of commands and identify the suitable feature set and classifier for optimal results to be compared with the Bi-LSTM model, focusing primarily on thermodynamic and Z-profile attacks. For experimental settings, we employed the RF algorithm with its default configuration of 100 trees. To mitigate class imbalance, we assigned weights to classes inversely proportional to their frequencies in the training set (see Table 4), ensuring that underrepresented classes receive higher weights, thereby balancing their influence during training. For the MLP architecture, we conducted extensive experiments to optimize its parameters. The final configuration consists of three hidden layers, with 256 neurons in the first layer and 128 neurons in each of the following layers. It uses ReLU activation, focal loss function, and Adam optimizer, with a batch size of 32, and is trained for 100 epochs. This architecture was chosen based on its superior performance in our experiments.

Dataset preparation

We used the dataset $DS6$ represented by all the extracted features for training. We also created a new test dataset ($Test_2$) for evaluation. $Test_2$ is a variant of the original test dataset with control commands (M-commands) strategically removed from certain layers. For further clarification, we typically adjust the temperatures and fan speed of movement commands within a layer by injecting the corresponding M-commands into that layer or by manipulating the parameters of existing ones. However, in the $Test_2$ dataset, we removed such injected commands from specific layers. Instead, these layers are influenced by external M-commands that lie outside their own commands. For the Z-profile attack samples, we reduced the number of bed level alterations by removing the added Z parameter from movement commands and implementing a single change to the actual Z-value at the start of each of these layers. This approach creates a more challenging scenario to test both the Bi-LSTM model and the RF algorithm.

To create a labeled command dataset (DS_{part}), we select a representative group of 20 commands from each set of commands sharing the same values of S_n, S_b, S_f, L_{th}, Z_v, L_n, and L_{in} across all layer variations (discussed in Section 5.2). The resulting DS_{part} consists of 319687 commands, 270491 of them are benign ($\mathcal{B}$), 11418 are nozzle temperature ($\mathcal{T}_n$), 16562 are bed temperature ($\mathcal{T}_b$), 9403 are fan speed ($\mathcal{F}_s$), and 11813 are Z-profile ($\mathcal{Z}_p$). While $Test_2$ comprises 148139, 4952, 7765, 5658, and 9800 commands from each class, respectively.

Experimental results

Our experiments start by discussing the impact of training dataset size on the detection performance of the RF algorithm. Table 11 compares the performance of the RF model trained on DS_{part} and DS_{total}, which includes all the commands from all the layers. However, both datasets are represented using the complete set of 12 extracted features. While DS_{total} slightly improves detection for certain classes such as $\mathcal{Z}p$, $DSpart$ achieves comparable results with reduced training overhead. The efficiency of DS_{part}, despite being a subset, is attributed to the sampling strategy that effectively captures the variability of G-code commands, considering their repetitive nature within layers. For example, all commands within a layer might have the same nozzle temperature, resulting in a consistent feature (S_n) throughout the layer.

Table 11
Performance of RF with different training datasets.

Class	DS_{part}			DS_{total}		
	Prec	Rec	F1	Prec	Rec	F1
$\mathcal{B}$	0.974	0.994	0.984	0.986	0.992	**0.989**
$\mathcal{T}_n$	0.903	0.837	**0.869**	0.744	0.797	0.769
$\mathcal{T}_b$	0.990	0.997	0.993	0.999	0.996	**0.997**
$\mathcal{F}_s$	0.957	0.915	0.935	0.999	0.931	**0.964**
$\mathcal{Z}_p$	0.956	0.711	0.816	0.913	0.774	**0.838**

Table 12
Performance of RF with different feature sets.

Class	12-Features			14-Features		
	Prec	Rec	F1	Prec	Rec	F1
$\mathcal{B}$	0.974	0.994	**0.984**	0.969	0.997	0.983
$\mathcal{T}_n$	0.903	0.837	0.869	0.923	0.827	**0.872**
$\mathcal{T}_b$	0.990	0.997	0.993	0.999	0.998	**0.998**
$\mathcal{F}_s$	0.957	0.915	0.935	0.964	0.908	0.935
$\mathcal{Z}_p$	0.956	0.711	**0.816**	0.966	0.632	0.764

After identifying DS_{part} as the appropriate training dataset, we investigate the impact of incorporating additional features while representing the G-code commands on the RF's performance. Table 12 illustrates the effect of including C_{idx} and C_{in} features alongside the primary set of 12 features on RF's performance trained on DS_{part}. While including these features slightly improves the detection of $\mathcal{T}_n$ and $\mathcal{T}_b$ attacks, it negatively impacts detecting the $\mathcal{Z}_p$ attack. However, these features provide positional information within the layer, which is not relevant for Z-profile attacks that typically affect the Z-value consistently across the entire layer. Therefore, including these features leads the RF algorithm to assign less importance to the Z-value feature, which is essential for identifying Z-profile attacks.

With these 12 identified features, Fig. 27 illustrates their relative importance in detecting thermodynamic and Z-profile attacks. The RF classifier assigns the highest weights to features most relevant for attack detection, such as 0.2599 to bed temperature (S_b), 0.1921 to fan speed (S_f), 0.1608 to Z-value (Z_v), and 0.1494 to nozzle temperature (S_n) features. Our analysis reveals that the feature importance rankings strongly align with theoretical expectations of G-code attack characteristics. Moreover, for class-specific feature importance, the bed temperature feature (S_b) demonstrates dominant importance (0.7989) for bed temperature attack detection, while nozzle temperature (S_n) shows significant weighting (0.5568) for nozzle temperature attacks. Similarly, Z-value (Z_v) exhibits substantial importance (0.5462) for Z-profile attacks, and fan speed (S_f) maintains meaningful contribution (0.3843) for fan speed attack detection. This differentiated feature importance distribution provides evidence that the model has learned meaningful command patterns rather than overfitting to noise.

The final experiment further provides strong evidence against overfitting the RF algorithm while showing a robust pattern of generalization. First, we applied hyperparameter tuning at the model level, setting the maximum tree depth to 20, minimum samples for split to 10, and minimum samples per leaf to 4. However, the classifier exhibits slight deviations from its performance using the default configurations. Overall, RF gives an F1 score of 0.9069, showing strong generalization capabilities. The performance varies logically across attack types, with high performance on distinct attack patterns (bed temperature attacks: 0.9949 F1) and relatively lower but still robust performance on subtle modifications (Z-profile attacks: 0.8105 F1). This variation reflects the

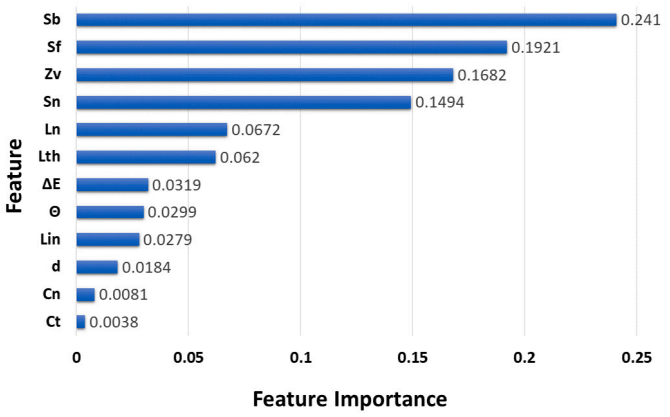

Fig. 27. Overall feature importance assigned by RF.

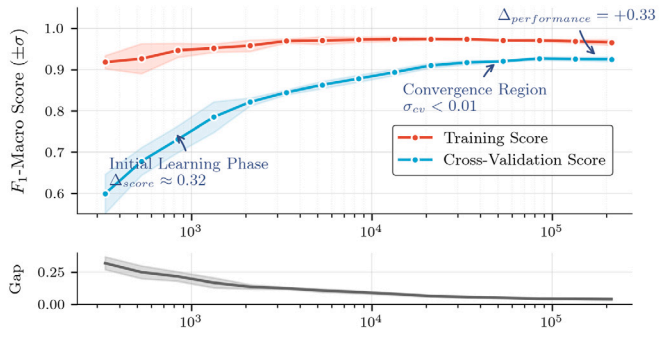

Fig. 28. Learning curves showing the RF performance in terms of F1-macro training and cross-validation scores with varying training dataset size.

inherent characteristics of different attack types rather than indicating overfitting artifacts. Moreover, Fig. 28 shows the learning curve utilizing Cross-Validation (CV) with K=5 and 319668 command entries, where sampling sizes followed a logarithmic scale. The F1 macro scores represent the performance of the RF classifier at different training sample sizes. Starting with a training score of 0.9182 (±0.0160) and CV score of 0.5993 (±0.0473), the model steadily improves to a training score of 0.9656 (±0.0071) and CV score of 0.9253 (±0.0059). The significant reduction in the training-CV gap (from 0.3189 to 0.0403) and the decreasing standard deviations indicate proper learning rather than memorization. Notably, the CV scores show consistent improvement up to 0.33 without plateauing early, indicating effective learning of underlying patterns. This combination of the high F1 scores, theoretically aligned feature importance, and stable learning curves demonstrates that our model has achieved genuine learning of G-code attack patterns rather than overfitting to training.

With the optimal dataset and feature set determined, our final step is identifying the best classifier between RF and MLP. Table 13 highlights the RF's superior performance across all classes, particularly in detecting $\mathcal{T}_n$ and $\mathcal{Z}_p$) attacks with an F1 score of 0.869 and 0.816, respectively. The RF's ensemble approach enables it to capture complex patterns in G-code data more effectively than the MLP's single network structure, especially for the nuanced changes caused by these attack types. Moreover, RF inherently performs feature selection, focusing on the most relevant attributes for each attack type. This is crucial for distinguishing between different types of thermodynamic attacks and identifying subtle Z-profile modifications.

Building on the best results obtained with the RF classifier trained on DS_{part} using the main 12 features, we compare these outcomes with the performance of the Bi-LSTM model on $Test_2$, as shown in Table 14.

Table 13
Performance comparison of MLP and RF, trained on DS_{part}.

Class	MLP			RF		
	Prec	Rec	F1	Prec	Rec	F1
$\mathcal{B}$	0.959	0.886	0.921	0.974	0.994	**0.984**
$\mathcal{T}_n$	0.216	0.822	0.342	0.903	0.837	**0.869**
$\mathcal{T}_b$	1.000	0.917	0.957	0.990	0.997	**0.993**
$\mathcal{F}_s$	0.868	0.900	0.884	0.957	0.915	**0.935**
$\mathcal{Z}_p$	0.493	0.385	0.432	0.956	0.711	**0.816**

Table 14
Performance comparison of Bi-LSTM (layer classification) and RF (command classification).

Class	Bi-LTSM			RF		
	Prec	Rec	F1	Prec	Rec	F1
$\mathcal{B}$	0.923	0.661	0.771	0.974	0.994	**0.984**
$\mathcal{T}_n$	0.365	0.316	0.339	0.903	0.837	**0.869**
$\mathcal{T}_b$	0.881	0.349	0.500	0.990	0.997	**0.993**
$\mathcal{F}_s$	0.399	0.390	0.394	0.957	0.915	**0.935**
$\mathcal{Z}_p$	0.049	0.374	0.087	0.956	0.711	**0.816**

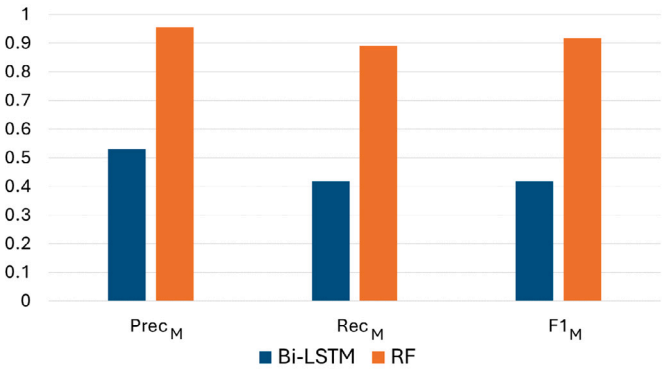

Fig. 29. Overall performance comparison of Bi-LSTM (layer classification) and RF (command classification).

The RF model demonstrates superior command-level classification, particularly in detecting Z-profile attacks, with an F1 score of 81.6%. This highlights the effectiveness of our finer-grained approach in addressing the limitations of the Bi-LSTM model. When entire layers are impacted by malicious changes outside of their commands, there are no injected malicious control commands to trigger Bi-LSTM's attention mechanism. Thus, the consistent values of the impacted feature across all layer commands make it difficult for attention-based Bi-LSTM to recognize such cases, which are included in $Test_2$. Furthermore, these external modifications impact fewer features across the layer. For instance, injecting an M-command affects both the C_t, C_n, and one of the S_n, S_b, and S_f features. In contrast, an external command impacts only one feature among S_n, S_b, and S_f, resulting in a more subtle footprint for the Bi-LSTM model to detect.

Finally, Fig. 29 illustrates how applying finer-grained classification at the command level using the RF algorithm gives better overall detection of subtle thermodynamic and Z-profile manipulations compared to the Bi-LSTM model.

7. Real-world case studies

This section evaluates the efficacy of our approach in real-world scenarios previously considered in the literature, such as printing face shield headbands [29] and drone propellers [14]. The code and datasets used in these case studies are available in our GitHub repository[1]

[1] https://github.com/HalaAli198/ML-based-G-code-Attacks-Detection

Table 15
Slicing parameters used to generate the G-code file for the face shield headband design.

Parameter	Value
Material	Generic PLA
Nozzle Diameter	0.4 mm
Print Profile	fast
Layer Thickness	0.2 mm
Infill Pattern	Lines
Infill Density	100%
Infill Direction	45°
Print Speed	50 mm/s
Initial Print Temp.	210°
Print Temp.	205°
Fan Speed	100%

7.1. Case study 1 (face shield headband):

During the COVID-19 pandemic, 3D printing technology has been widely used for producing personal protective equipment (PPE), specifically face shield headbands [91]. This global crisis necessitated unprecedented levels of digital file exchange between designers and manufacturers all over the world, introducing significant security risks to the design and G-code files. Malicious manipulation of these files could compromise PPE functionality, endangering healthcare professionals in critical environments.

To validate the detection capabilities of our approach against such malicious manipulations, we conducted an experiment using a face shield headband design. The STL file was downloaded from the *Thingiverse* repository [67] and sliced by Ultimaker Cura 5.9.0 using the parameters detailed in Table 15 to generate a G-code file comprising 215 distinct layers. We then introduced localized manipulations to the generated G-code, targeting the shield's critical joint points across layers 1–48, as illustrated in Fig. 30(a). These manipulations included creating structural discontinuities at crucial junctions through a filament state attack, as shown in Fig. 30(b). Physical verification of the attack's impact, demonstrated in Fig. 30(c), revealed significant structural compromises in the manufactured product, with the introduction of critical failure points rendering the headband unsuitable for use, as evidenced in Fig. 30(d).

For evaluation, we created a test dataset by compromising the generated G-code using the attack methods described earlier in the study (see Section 2.3). This dataset comprises eight G-code files, including one benign file and one file for each type of attack. After processing, the dataset contains 628 benign layers derived from the original file and unaffected layers within the compromised files. The cavity attack impacts 141 layers, introducing cavities with a size of 0.6 mm. The filament density attack reduces filament density by 40%, affecting all layers except the bottom two and top two, with five lines impacted per layer. The filament state attack disables material deposition for three lines across the first 48 layers, targeting joint points, while the remaining attacks affect all of the 215 layers with various adjustments.

Table 16 presents the testing results. As all attacks target the same file, and due to the overlapping patterns of filament attacks, it is expected for the model to exhibit some misclassification between classes. Furthermore, with the thermodynamic attacks, the layer sequences remain consistent across copies, with only attack-specific modifications differing. Such changes in consistent layers effectively trigger the Bi-LSTM's attention mechanism, resulting in enhanced attack type recognition compared to the testing results against $Test_1$ in the previous experiments. However, the lower F1 score with the Z-profile attack ($\mathcal{Z}_p$), as discussed earlier, arises from its subtle footprint, which is detected better by command-level classification using the RF algorithm. Overall, the Bi-LSTM model achieves a macro F1-score of 0.856, highlighting its efficacy in detecting manipulations in real-world scenarios before printing starts, thereby saving time, material, and effort.

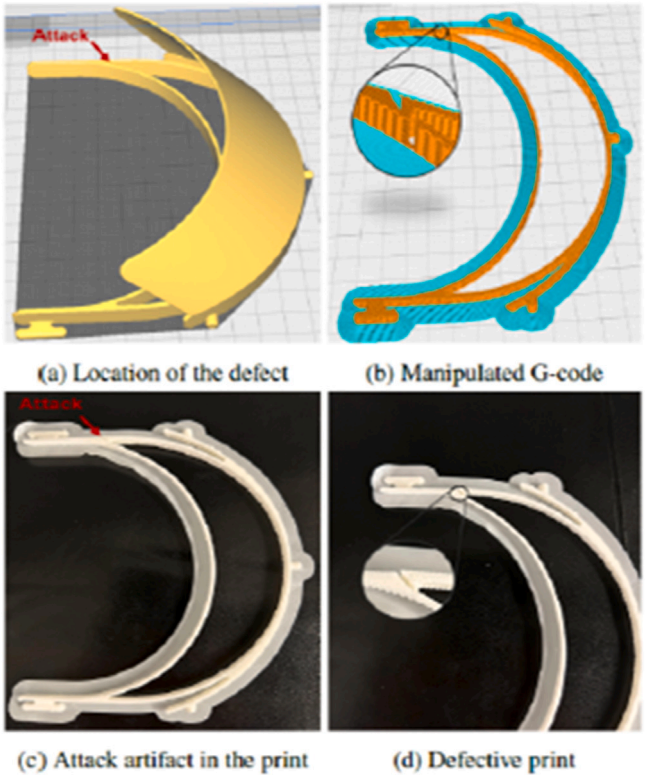

Fig. 30. Face shield headband sabotaged by the filament state attack.

Table 16
Performance of Bi-LSTM with compromised variants of face shield headband G-code.

Class	Prec	Rec	F1
$\mathcal{B}$	0.807	0.949	0.872
$\mathcal{CV}$	0.903	0.794	0.845
$\mathcal{FS}$	0.989	0.834	0.905
$\mathcal{FT}$	0.894	0.875	0.884
$\mathcal{T}_n$	0.973	0.851	0.908
$\mathcal{T}_b$	0.901	0.930	0.915
$\mathcal{T}_s$	0.936	0.744	0.829
$\mathcal{Z}_p$	0.680	0.693	0.687

7.2. Case study 2 (drone propeller):

In addition to face shield headbands, 3D printing technology has also been used to produce drone propellers. Belikovetsky et al. [14] demonstrated a real-world G-code attack targeting drone propellers by introducing gaps at the joint points, which caused the drone to fail shortly after takeoff, as illustrated in Fig. 31. The gaps were created by mimicking the behavior of the filament state attack, setting identical *"E"* parameters for consecutive G1 commands, as detailed in Table 2 of their paper. The study also highlighted the potential risks of compromising G-code files through vulnerable slicing software. Building on this observation, we identified a new attack vector targeting plaintext G-code during its transmission via Inter-Process Communication (IPC) between the processes of the Ultimaker Cura application. This vulnerability, registered as CVE-2024-51330,[2] demonstrates the critical need for robust security measures to protect G-code in 3D printing workflows.

[2] https://nvd.nist.gov/vuln/detail/CVE-2024-51330

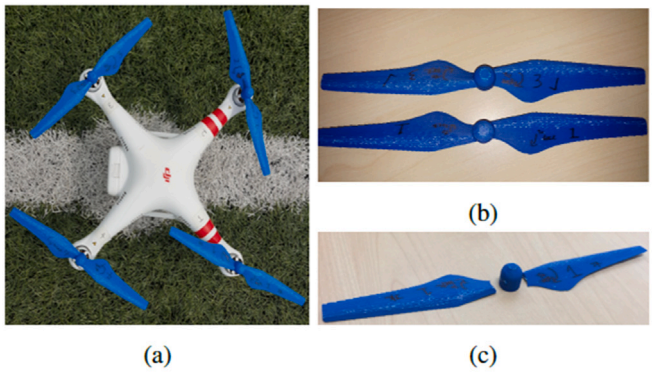

Fig. 31. Drone propellers sabotaged by G-code attack [14].

Table 17
Performance of RF with compromised variants of drone propeller G-code.

Class	Prec	Rec	F1
$\mathcal{B}$	0.6979	0.9205	0.7939
$\mathcal{T}_n$	0.8570	0.8890	0.8727
$\mathcal{T}_b$	0.9975	0.9981	0.9978
$\mathcal{F}_s$	0.9939	0.7789	0.8733
$\mathcal{Z}_p$	0.8852	0.7597	0.8177

Our first case study demonstrated the filament state attack. Therefore, in this case study, we illustrate the effects of a different attack, disabling the cooling fan, during the fabrication of the drone propeller. To this end, we downloaded the STL file for the propeller from the *Thingiverse* repository, sliced by Ultimaker Cura 5.9 using the same parameters detailed in Table 15, and then injected an M107 command at the beginning of the G-code file by exploiting our discovered vulnerability to disable the cooling fan. This attack caused material melting as shown in Fig. 32(a). Furthermore, Figs. 32(b) and 32(c) illustrate the front and back sides of the defective propeller. The back side, which faced the support structures during printing, exhibits severe stringing or oozing - thin, hair-like plastic strands due to filament oozing from the hot nozzle without proper cooling. These defects impact the print quality, making it unsuitable for use.

The defects observed in the propeller print are typically caused not only by adjusting the fan speed but also by changing the nozzle temperature, bed temperature, and layer thickness. Large gaps between layers reduce surface area contact, thus weakening layer adhesion. To demonstrate the RF algorithm's capability in detecting such subtle changes, we created a test dataset by compromising the G-code with thermodynamic and Z-profile attacks. The dataset comprises five 87-layer G-code files, one benign and one for each attack type. Processing these files yields 54260 benign commands ($\mathcal{B}$), 54346 extruder temperature attack commands ($\mathcal{T}_n$), 54347 bed temperature attack commands ($\mathcal{T}_b$), 54258 fan speed attack commands ($\mathcal{F}_s$), and 54260 Z-profile attack commands ($\mathcal{Z}_p$).

Given the finer-grained classification at the command level, and due to targeting the same commands with various attacks, the chance of misclassifying the malicious commands as benign increased, leading to an F1 score of 0.7939 with the bending class as demonstrated in Table 17. However, in our previous experiments, the RF algorithm showed improved performance with fewer misclassifications while testing on a smaller dataset comprising commands from various G-code files ($Test_2$). This improvement can be attributed to the greater diversity in base patterns and command contexts across different files, which helps the classifier better distinguish between normal variations and attack-induced changes. Moreover, we observe that bed temperature attack ($\mathcal{T}_b$) consistently achieves the highest F1 scores (0.997), aligning with the high importance assigned to its corresponding bed temperature feature (S_b), as discussed in detail in Section 6.8, providing evidence that the model has learned meaningful command patterns rather than overfitting to noise.

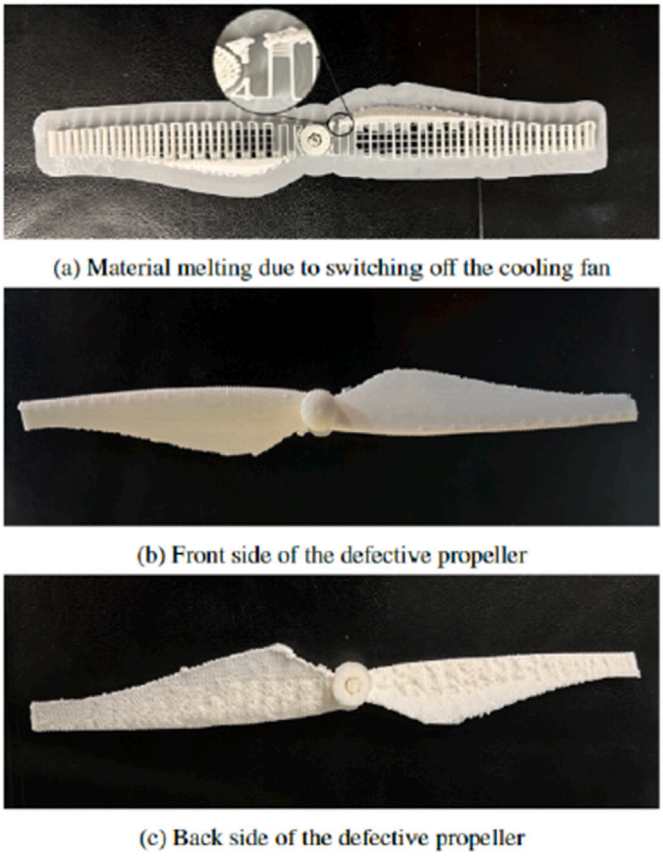

Fig. 32. Drone propeller sabotaged by switching off the cooling fan.

8. Scalability and industrial applicability

This section analyzes the scalability of our Bi-LSTM-based approach for G-code attack detection in larger-scale applications. It is essential to highlight that 3D printing typically follows a "slice-once, print-later" pipeline. G-code generation and analysis occur before the actual printing process begins. Given that our approach is designed as offline (pre-print) detection of G-code attacks, it eliminates real-time processing constraints and makes computational time less critical than accuracy. This early-stage detection of G-code manipulations also offers the significant benefit of preventing material waste and machine time loss.

Furthermore, at the data level, G-code files are systematically divided into layers, each treated as an independent bag within the MIL context. Our methodology is a layer-wise classification, allowing incremental analysis of G-code files by processing the layers independently rather than processing the entire file at once. This approach enhances computational efficiency and aligns well with both small-scale and industrial settings. At the model level, Bi-LSTM employs a selective attention mechanism that focuses on the most relevant segments within the G-code layers while maintaining a minimal three-layer architecture that balances performance with computational complexity. This design also enables efficient batch processing for layer-level analysis, maximizing computational resource utilization through parallel processing capabilities.

To illustrate the practical scalability of our methodology, we conducted an empirical performance analysis utilizing a face shield headband manufacturing case study, encompassing a dataset of 1935 layers. Our experimental measurements demonstrate efficient processing metrics. Utilizing GPU acceleration (NVIDIA A100), the system achieved an average inference latency of 0.0434 s per layer, requiring approximately 9.331 s for processing an entire G-code file comprising 215 layers while maintaining an average memory consumption of 0.59 MB. In contrast, CPU-based processing (Intel Xeon had an average inference latency of 1.5949 s per layer, requiring approximately 5.715 min to process a complete file.

The industrial applicability of our approach is further demonstrated through a scaled manufacturing scenario involving 100 Ultimaker 3 printers producing 1000 headbands. In this context, each printer requires approximately 23.3 h as a printing time of 10 files. Detection of malicious modifications post-printing initiation could result in significant resource wastage, including material costs for 1000 headbands and operational expenses for 100 printers. Conversely, before printing starts, our approach can process all 1000 files in approximately 3 h (9.331 s ×1000 files) on a single GPU (NVIDIA A100) node, demonstrating efficient resource utilization relative to potential losses from compromised design printing.

This comparison demonstrates that the processing overhead of our approach is reasonable and practically manageable in industrial settings, providing an efficient solution for pre-print detection of G-code attacks in real-world manufacturing scenarios. Future research directions could explore additional optimization strategies to further enhance performance in large-scale deployments. Given our layer-independent processing approach, distributed computing architectures could enable parallel analysis of multiple G-code files simultaneously across GPU nodes, potentially reducing the total processing time.

9. Limitations

Our proposed approach may face challenges with detecting minor changes in the geometry of the design represented by X and Y coordinates [10,15]. The varying X and Y parameters within and across layers complicate the establishment of an anomaly baseline, making it difficult to detect such changes. Addressing these issues may require comparing G-code with ground truth and real-time monitoring to reconstruct G-code and identify discrepancies. Furthermore, for simplicity, we assumed that each layer or command is impacted by a single attack. However, in practice, layers can be affected by multiple attacks simultaneously. Different labeling strategies can be employed to address this. *Composite Labeling* represents specific combinations of attacks, with each label corresponding to a unique set of two or more attack types. *Multi-Labeling*, on the other hand, assigns multiple labels to each layer or command to reflect various attack types. Additionally, the ML model needs to be continuously updated as new types of G-code attacks emerge. While our method is effective in detecting known attacks, it may not be able to identify novel attack patterns that were not present in the training data.

10. Conclusion

This paper presents a novel approach for early detection of G-code attacks in 3D printing, thereby saving both time and resources. Our method leverages machine learning techniques to distinguish accurately between benign and potentially malicious manipulations caused by these attacks. It provides insights into the nature of such modifications without requiring a reference model. Several research challenges were addressed, including generating an efficient G-code dataset with optimal characteristics, extracting informative features, optimally segmenting and labeling G-code files, and determining the appropriate ML classification technique. Our approach involved generating and analyzing diverse G-code datasets with varying 3D designs and slicing parameters. A formal analysis identified the most informative features, including command type and number, movement distance and direction angle of the print head, filament amount, thermodynamic settings, Z-parameter value, and layer characteristics. The Bi-LSTM model, enhanced by an attention mechanism and focal loss function, was employed for layer classification, while the RF algorithm was used for command classification. Both classifiers effectively handled imbalanced datasets. Experimental results demonstrated the effectiveness of this approach. The Bi-lSTM model achieved F1 scores up to 91.3%, while the RF algorithm demonstrated superior performance in detecting minor thermodynamic and Z-profile changes at the command level, achieving F1 scores between 81.6% and 99.3%. Based on our findings, we provided guidelines for analyzing G-code data to facilitate future research in this domain.

Future work will focus on building a more comprehensive dataset, exploring transformer-based learning techniques to improve performance, and real-world deployment of our approach into existing 3D printing systems. Additionally, we plan to develop adaptive response mechanisms when G-code manipulations are detected. Beyond the current approach that discards compromised G-code files, we aim to develop automated repair capabilities that could restore manipulated G-code to safe configurations based on detected attack types. We also plan to investigate selective printing methods that could bypass only the compromised layers while executing the unaffected portions of the G-code file. Furthermore, developing risk assessment frameworks that evaluate the severity and impact of detected manipulations would guide appropriate response strategies based on the security implications of specific attack types. These enhancements would transform our detection system into a comprehensive security framework for additive manufacturing workflows, providing not only early warning but also actionable remediation strategies.

CRediT authorship contribution statement

Hala Ali: Writing – original draft, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Alberto Cano:** Writing – original draft, Supervision, Methodology, Conceptualization. **Irfan Ahmed:** Writing – original draft, Supervision, Methodology, Conceptualization.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

High Performance Computing resources provided by the High Performance Research Computing (HPRC) core facility at Virginia Commonwealth University (https://hprc.vcu.edu) were used for conducting the research reported in this work.

Data availability

Code and data are shared on Github and the link is provided in the manuscript.

References

[1] Budzik G, Tomaszewski K, Soboń A. Opportunities for the application of 3D printing in the critical infrastructure system. Energies 2022;15(5):1656.

[2] Ma X, Li Z, Li H, An Q, Qiu Q, Xu W, et al. Database and benchmark for early-stage malicious activity detection in 3D printing. In: 2020 25th Asia and south Pacific design automation conference. IEEE; 2020, p. 494–9.

[3] Jacobs JB, Haberman A. 3D-printed firearms, do-it-yourself guns, & the second amendment. Law Contemp Probl 2017;80(2):129–47.

[4] Yang J, Liu Y. Hybrid modelling method for the prediction and experimental validation of 3D printing resource consumption. J Manuf Process 2023;101:1275–300.

[5] Hasanov S, Alkunte S, Rajeshirke M, Gupta A, Huseynov O, Fidan I, et al. Review on additive manufacturing of multi-material parts: progress and challenges. J Manuf Mater Process 2021;6(1):4.

[6] Yu S-Y, Malawade AV, Chhetri SR, Al Faruque MA. Sabotage attack detection for additive manufacturing systems. IEEE Access 2020;8:27218–31.

[7] Yampolskiy M, Graves L, Gatlin J, Skjellum A, Yung M. What did you add to my additive manufacturing data?: steganographic attacks on 3D printing files. In: Proceedings of the 24th international symposium on research in attacks, intrusions and defenses. 2021, p. 266–81.

[8] Sturm LD, Williams CB, Camelio JA, White J, Parker R. Cyber-physical vulnerabilities in additive manufacturing systems: A case study attack on the. STL file with human subjects. J Manuf Syst 2017;44:154–64.

[9] Wu X, Xue T, Mao S. G-code Net: Learning-based rational design and optimization for additively manufactured structures. MRS Commun 2024;1–9.

[10] Moore SB, Glisson WB, Yampolskiy M. Implications of malicious 3D printer firmware. In: Proceedings of the 50th Hawaii international conference on system sciences. 2017.

[11] Pearce H, Yanamandra K, Gupta N, Karri R. FLAW3D: A Trojan-based cyber attack on the physical outcomes of additive manufacturing. IEEE/ASME Trans Mechatronics 2022;27(6):5361–70.

[12] Kurkowski E, Van Stockum A, Dawson J, Taylor C, Schulz T, Shenoi S. Manipulation of G-code toolpath files in 3D printers: Attacks and mitigations. In: Critical infrastructure protection XVI: 16th IFIP WG 11.10 international conference, ICCIP 2022, virtual event, March 14–15, 2022, revised selected papers. Springer; 2022, p. 155–74.

[13] Mishra PK, Senthil P, Adarsh S, Anoop M. An investigation to study the combined effect of different infill pattern and infill density on the impact strength of 3D printed polylactic acid parts. Compos Commun 2021;24:100605.

[14] Belikovetsky S, Yampolskiy M, Toh J, Gatlin J, Elovici Y. Dr0wned-cyber-physical attack with additive manufacturing. In: WOOT. 2017.

[15] Rais M, Ahsan M, Sharma V, Barua R, Prins R, Ahmed I. Low-magnitude infill structure manipulation attacks on FFF-based 3D printersâ. In: Proceedings of the 16th IFIP working group 11.10 international conference on critical infrastructure protection (March 2022). 2022, https://scholar.google.com/citations.

[16] Bayens C, Le T, Garcia L, Beyah R, Javanmard M, Zonouz S. See no evil, hear no evil, feel no evil, print no evil? Malicious fill patterns detection in additive manufacturing. In: 26th USeNIX security symposium. 2017, p. 1181–98.

[17] Gao Y, Li B, Wang W, Xu W, Zhou C, Jin Z. Watching and safeguarding your 3D printer: Online process monitoring against cyber-physical attacks. Proc ACM Interact Mob Wearable Ubiquitous Technol 2018;2(3):1–27.

[18] Cho EE, Hein HH, Lynn Z, Hla SJ, Tran T. Investigation on influence of infill pattern and layer thickness on mechanical strength of PLA material in 3D printing technology. J Eng Sci Res 2019;3(2):27–37.

[19] Patil S, Deshpande Y, Parle D. Extracting slicer parameters from STL file in 3D printing. Int J Intell Syst Appl Eng 2024;12(14s):192–204.

[20] Hou J-U, Kim D-G, Lee H-K. Blind 3D mesh watermarking for 3D printed model by analyzing layering artifact. IEEE Trans Inf Forensics Secur 2017;12(11):2712–25.

[21] Rais MH, Li Y, Ahmed I. Spatiotemporal G-code modeling for secure FDM-based 3D printing. In: Proceedings of the ACM/IEEE 12th international conference on cyber-physical systems. 2021, p. 177–86.

[22] Rais MH, Li Y, Ahmed I. Dynamic-thermal and localized filament-kinetic attacks on fused filament fabrication based 3D printing process. Addit Manuf 2021;46:102200.

[23] Si H, Zhang Z, Huseynov O, Fidan I, Hasan SR, Mahmoud M. Machine learning-based investigation of the 3D printer cooling effect on print quality in fused filament fabrication: A cybersecurity perspective. Inventions 2023;8(1):24.

[24] Lewis J, Moore AL, et al. In situ infrared temperature sensing for real-time defect detection in additive manufacturing. Addit Manuf 2021;47:102328.

[25] Li F, Yu Z-H, Li H, Yang Z-S, Kong Q-S, Tang J. Real-time monitoring of raster temperature distribution and width anomalies in fused filament fabrication process. Adv Manuf 2022;10(4):571–82.

[26] Nam J, Jo N, Kim JS, Lee SW. Development of a health monitoring and diagnosis framework for fused deposition modeling process based on a machine learning algorithm. Proc Inst Mech Eng Part B: J Eng Manuf 2020;234(1–2):324–32.

[27] Li Z, Gong D, Tan L, Luo X, Liu F, Bors AG. Self-embedding watermarking method for G-code used in 3D printing. In: 2021 IEEE international workshop on information forensics and security. WIFS, IEEE; 2021, p. 1–6.

[28] Oligschlaeger Z, Baltes B, Chin J. Secure 3D printing. 2020, URL https://www.freepatentsonline.com/y2020/0326683.htm. [Accessed 23 April 2024].

[29] Shi Z, Kan C, Tian W, Liu C. A blockchain-based G-code protection approach for cyber-physical security in additive manufacturing. J Comput Inf Sci Eng 2021;21(4).

[30] Kohavi R, John GH. Wrappers for feature subset selection. Artificial Intelligence 1997;97(1–2):273–324.

[31] Cano A. An ensemble approach to multi-view multi-instance learning. Knowl-Based Syst 2017;136:46–57.

[32] Cano A, Zafra A, Ventura S. Speeding up multiple instance learning classification rules on GPUs. Knowl Inf Syst 2015;44(1):127–45.

[33] Zhou Z-H, Jiang K, Li M. Multi-instance learning based web mining. Appl Intell 2005;22:135–47.

[34] Liu G, Guo J. Bidirectional LSTM with attention mechanism and convolutional layer for text classification. Neurocomputing 2019;337:325–38.

[35] Lin T-Y, Goyal P, Girshick R, He K, Dollár P. Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision. 2017, p. 2980–8.

[36] Breiman L. Random forests. Mach Learn 2001;45:5–32.

[37] Tang J, Deng C, Huang G-B. Extreme learning machine for multilayer perceptron. IEEE Trans Neural Netw Learn Syst 2015;27(4):809–21.

[38] Jignasu A, Rurup JD, Secor EB, Krishnamurthy A. NURBS-based path planning for aerosol jet printing of conformal electronics. J Manuf Process 2024;118:187–94.

[39] Dong B, Wang Y, Lu Y. A slicing and path generation method for 3D printing of periodic surface structure. J Manuf Process 2024;120:694–702.

[40] Zeltmann SE, Gupta N, Tsoutsos NG, Maniatakos M, Rajendran J, Karri R. Manufacturing and security challenges in 3D printing. JOM 2016;68(7):1872–81.

[41] Schmidt A-M, Kyosev Y. Particle based simulation of the polymer penetration into porous structures during the fused deposition modelling. J Manuf Process 2023;101:1205–13.

[42] Zupfe gcode viewer. 2024, https://zupfe.velor.ca/. [Accessed 25 May 2024].

[43] Leite M, Fernandes J, Deus AM, Reis L, Vaz MF. Study of the influence of 3D printing parameters on the mechanical properties of PLA. 2018.

[44] Ahsan M, Rais MH, Ahmed I. SOK: Side channel monitoring for additive manufacturing - bridging cybersecurity and quality assurance communities. In: 2023 IEEE 8th European symposium on security and privacy. 2023, p. 1160–78. http://dx.doi.org/10.1109/EuroSP57164.2023.00071.

[45] Xiao C. Security attack to 3D printing. In: XFocus information security conference. 2013.

[46] Delli U, Chang S. Automated process monitoring in 3D printing using supervised machine learning. Procedia Manuf 2018;26:865–70.

[47] Kumar S, Kannan VN, Sankaranarayanan G. Parameter optimization of ABS-M30i parts produced by fused deposition modeling for minimum surface roughness. Int J Curr Eng Technol 2014;3(3):93–7.

[48] Moore S, Armstrong P, McDonald T, Yampolskiy M. Vulnerability analysis of desktop 3D printer software. In: 2016 resilience week. RWS, IEEE; 2016, p. 46–51.

[49] Zhou H, Liu C, Tian W, Kan C. Echo state network learning for the detection of cyber attacks in additive manufacturing. In: 2021 IEEE 17th international conference on automation science and engineering. CASE, IEEE; 2021, p. 177–82.

[50] Zhang Z, Fidan I. Machine learning-based void percentage analysis of components fabricated with the low-cost metal material extrusion process. Materials 2022;15(12):4292.

[51] Khanzadeh M, Rao P, Jafari-Marandi R, Smith BK, Tschopp MA, Bian L. Quantifying geometric accuracy with unsupervised machine learning: Using self-organizing map on fused filament fabrication additive manufacturing parts. J Manuf Sci Eng 2018;140(3).

[52] Kurkowski E, Rice M, Shenoi S. Detecting part anomalies induced by cyber attacks on a powder bed fusion additive manufacturing system. In: Critical infrastructure protection XVI: 16th IFIP WG 11.10 international conference, ICCIP 2022, virtual event, March 14–15, 2022, revised selected papers. Springer; 2022, p. 175–203.

[53] Moore SB, Gatlin J, Belikovetsky S, Yampolskiy M, King WE, Elovici Y. Power consumption-based detection of sabotage attacks in additive manufacturing. 2017, arXiv preprint arXiv:1709.01822.

[54] Chhetri SR, Canedo A, Al Faruque MA. Kcad: kinetic cyber-attack detection method for cyber-physical additive manufacturing systems. In: 2016 IEEE/ACM international conference on computer-aided design. ICCAD, IEEE; 2016, p. 1–8.

[55] Bhandarkar VV, Kumar A, Tandon P. Warpage detection in 3D printing of polymer parts: a deep learning approach. J Intell Manuf 2024;1–13.

[56] Wu M, Zhou H, Lin LL, Silva B, Song Z, Cheung J, Moon Y. Detecting attacks in cybermanufacturing systems: Additive manufacturing example. In: MATEC web of conferences, vol. 108. EDP Sciences; 2017, p. 06005.

[57] Al Mamun A, Liu C, Kan C, Tian W. Real-time process authentication for additive manufacturing processes based on in-situ video analysis. Procedia Manuf 2021;53:697–704.

[58] Yang W, Chen J, Zhang C, Paynabar K. Online detection of cyber-incidents in additive manufacturing systems via analyzing multimedia signals. Qual Reliab Eng Int 2022;38(3):1340–56.

[59] Goh GD, Hamzah NMB, Yeong WY. Anomaly detection in fused filament fabrication using machine learning. 3D Print Addit Manuf 2023;10(3):428–37.
[60] Petsiuk A, Pearce JM. Towards smart monitored AM: Open source in-situ layer-wise 3D printing image anomaly detection using histograms of oriented gradients and a physics-based rendering engine. Addit Manuf 2022;52:102690.
[61] Blender Foundation. Blender: The free and open source 3D creation suite. Blender Foundation; 2025, https://www.blender.org/. [Accessed 14 March 2025].
[62] Kumar R, Sangwan KS, Herrmann C, Ghosh R, Sangwan M. Development and comparison of machine-learning algorithms for anomaly detection in 3D printing using vibration data. Prog Addit Manuf 2024;9(2):529–41.
[63] Lasluisa-Naranjo H, Rivas-Lalaleo D, Vaquero-López J, Cruz-Moposita C. Machine learning G-code optimization. Eng Proc 2024;77(1):32.
[64] G-code — RepRap. 2019, URL https://reprap.org/wiki/G-code. [Accessed 10 July 2019].
[65] Voříšek J, Patzák B. GPAMS: A G-code processor for advanced additive manufacturing simulations. Addit Manuf 2023;65:103279.
[66] Viewstl online tool. 2024, https://www.viewstl.com/. [Accessed 15 May 2024].
[67] Thingiverse. 2024, URL https://www.thingiverse.com/. [Accessed 20 December 2023].
[68] Ultimaker cura software. 2024, URL https://ultimaker.com/software/ultimaker-cura. [Accessed 16 May 2024].
[69] Al Mamun A, Liu C, Kan C, Tian W. Securing cyber-physical additive manufacturing systems by in-situ process authentication using streamline video analysis. J Manuf Syst 2022;62:429–40.
[70] Rismalia M, Hidajat S, Permana I, Hadisujoto B, Muslimin M, Triawan F. Infill pattern and density effects on the tensile properties of 3D printed PLA material. J Phys Conf Ser 2019;1402(4):044041.
[71] Dey A, Yodo N. A systematic survey of FDM process parameter optimization and their influence on part characteristics. J Manuf Mater Process 2019;3(3):64.
[72] Qattawi A, Alrawi B, Guzman A, et al. Experimental optimization of fused deposition modelling processing parameters: a design-for-manufacturing approach. Procedia Manuf 2017;10:791–803.
[73] Peng A, Xiao X, Yue R. Process parameter optimization for fused deposition modeling using response surface methodology combined with fuzzy inference system. Int J Adv Manuf Technol 2014;73:87–100.
[74] Carneiro OS, Silva A, Gomes R. Fused deposition modeling with polypropylene. Mater Des 2015;83:768–76.
[75] Jackson B, Fouladi K, Eslami B. Multi-parameter optimization of 3D printing condition for enhanced quality and strength. Polymers 2022;14(8):1586.
[76] Alabi MO, Nixon K, Botef I. A survey on recent applications of machine learning with big data in additive manufacturing industry. Am J Eng Appl Sci 2018;11(3):1114–24.
[77] Hossain MR, Maung Than Oo A, Ali A. The combined effect of applying feature selection and parameter optimization on machine learning techniques for solar power prediction. CQUniversity; 2013.
[78] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput 1997;9(8):1735–80.
[79] Graves A, Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw 2005;18(5–6):602–10.
[80] Chen Y, Peng L, Wang Y, Zhou Y, Li C. Prediction of tandem cold-rolled strip flatness based on attention-LSTM model. J Manuf Process 2023;91:110–21.
[81] Zhou Z-H, Liu X-Y. On multi-class cost-sensitive learning. Comput Intell 2010;26(3):232–57.
[82] Liu X-Y, Wu J, Zhou Z-H. Exploratory undersampling for class-imbalance learning. IEEE Trans Syst Man Cybern B 2008;39(2):539–50.
[83] Sun Y, Kamel MS, Wong AK, Wang Y. Cost-sensitive boosting for classification of imbalanced data. Pattern Recognit 2007;40(12):3358–78.
[84] Tran D, Mac H, Tong V, Tran HA, Nguyen LG. A LSTM based framework for handling multiclass imbalance in DGA botnet detection. Neurocomputing 2018;275:2401–13.
[85] Forman G. A pitfall and solution in multi-class feature selection for text classification. In: Proceedings of the twenty-first international conference on machine learning. 2004, p. 38.
[86] 3DPEA GCode simulator. 2024, https://www.3dpea.com/en/gcode-simulator. [Accessed 15 May 2024].
[87] Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014, arXiv preprint arXiv:1412.3555.
[88] Shewalkar A, Nyavanandi D, Ludwig SA. Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU. J Artif Intell Soft Comput Res 2019;9(4):235–45.
[89] Ravi V, Alazab M, Srinivasan S, Arunachalam A, Soman K. Adversarial defense: DGA-based botnets and DNS homographs detection through integrated deep learning. IEEE Trans Eng Manage 2021;70(1):249–66.
[90] Özdemir Ö, Sönmez EB. Weighted cross-entropy for unbalanced data with application on covid x-ray images. In: 2020 innovations in intelligent systems and applications conference. ASYU, IEEE; 2020, p. 1–6.
[91] Manero A, Smith P, Koontz A, Dombrowski M, Sparkman J, Courbin D, et al. Leveraging 3D printing capacity in times of crisis: recommendations for COVID-19 distributed manufacturing for medical equipment rapid response. Int J Environ Res Public Heal 2020;17(13):4634.