People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Institute of Aeronautics and Space Studies – Saad Dahlab University of Blida 1

Saad Dahleb University

Faculty of Aeronautics

Department of Aeronautical Structures

Graduation Project Thesis

Title:

Predictive Fatigue Analysis Using Synthetic Data and Deep Learning Models

Student: Loubna Menia

Supervisor: Amale Mahi

Co-Supervisor: Faycal Ykhlef

Academic Year: 2024 – 2025

Contents

List of Figures

List of Tables

Nomenclature and Symbol Definitions

Symbol / Code	Description
	Mechanical / Physical Parameters
u, v	Displacement field components in COMSOL (horizontal/vertical)
$\epsilon_{G1}, \epsilon_{G2}, \epsilon_{G3}$	Strain at gauges G1, G2, G3 (in $\mu\varepsilon$)
$\Delta\sigma$	Applied stress range
ΔK	Stress intensity factor range
E	Young's modulus (material stiffness)
ν	Poisson's ratio (material lateral contraction coefficient)
K_{IC}	Fracture toughness (critical stress intensity factor)
$\sigma_{11}, \sigma_{22}, \tau_{12}$	Stress tensor components
$\epsilon_{11}, \epsilon_{22}, \epsilon_{33}$	Strain tensor components
	Fracture Mechanics and Crack Growth
a	Crack length (mm)
a_0	Initial crack length (mm)
a_{crit}	Critical crack length (e.g., failure threshold)
C	Paris' law material constant
m	Paris' law exponent
$\frac{da}{dN}$	Crack growth rate per cycle
	Simulation and Sensor Setup
$g1$, $g2$, $g3$	Virtual strain gauges placed near the crack

u-$g1$	Displacement at gauge 1 location
s-$g2$	Strain at gauge 2 location
`crack_tip_start`	Probe point at the beginning of the crack
`crack_tip_end`	Probe point at the end of the crack
Cycle	Number of loading cycles (fatigue life progression)
t^*	Observable lifetime fraction in truncated test sequences

Machine Learning and Evaluation

RUL	Remaining Useful Life (cycles before failure)
MAPE	Mean Absolute Percentage Error (performance metric)
TCN	Temporal Convolutional Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
XGBoost	Extreme Gradient Boosting (tree-based ML model)
1D-CNN	One-Dimensional Convolutional Neural Network
MinMaxScaler	Feature normalization method

Acknowledgment

First and foremost, I would like to express my sincere gratitude to Allah, whose guidance and blessings have given me the strength and perseverance to reach this significant milestone in my academic journey.

I extend my deepest appreciation to my supervisors: M.Amel Mahi , whose insights and support guided me throughout the development of this graduation project. Your expertise in the field of fatigue analysis and structural engineering has been invaluable.

To my family—especially my parents—thank you for your unconditional love, prayers, and endless encouragement. Your belief in me has been the foundation of my ambition and motivation.

A special thanks to my friends and colleagues who stood by me during the challenging phases of this project, offering help, laughter, and moral support when it was most needed.

I also want to acknowledge the authors and contributors of the FrameworkFDPP research project, which inspired my thesis and gave me the foundation to explore predictive fatigue analysis. This work has not only shaped my understanding of sequence models like LSTM and GRU but also motivated me to bridge the gap between academic research and real-world engineering applications.

To my professors at the university, thank you for your dedication and for sparking my passion for aeronautics and structural dynamics.

Finally, this work is also dedicated to my future aspirations. Whether in the skies or on the roads of automotive innovation, I aim to continue pushing boundaries, learning continuously, and contributing meaningfully—especially with the hope of one day bringing such advancements home to Algeria.

Abstract

This thesis explores a hybrid methodology combining finite element simulation and machine learning techniques to predict fatigue crack growth in metallic structures. Inspired by the work presented in *A Framework for Generating Large Data Sets for Fatigue Damage Prognostic Problems* (FDPP), this research investigates the credibility and effectiveness of synthetic data by comparing it with results obtained from high-fidelity simulations in COMSOL Multiphysics. A 2D fatigue crack model was simulated under cyclic loading, with strain gauges placed in accordance with the FDPP setup. The resulting strain and displacement data were processed using custom Python scripts to derive meaningful comparisons between the synthetic and simulated datasets. The primary contributions include validating the synthetic dataset's realism via numerical comparison, and introducing alternative machine learning models—such as XGBoost, LightGBM, and Autoencoders—for Remaining Useful Life (RUL) estimation. The findings demonstrate a promising correlation between real and synthetic data trends, paving the way for more credible fatigue prognostics using virtual datasets in conjunction with physics-based simulations.

Chapter 1

Introduction

Fatigue damage is a critical phenomenon affecting the safety and longevity of aeronautical structures. It refers to the progressive and localized structural deterioration that occurs when a material is subjected to cyclic loading over time [1]. In aircraft, components such as fuselage skin, riveted joints, and wing spars are highly susceptible to fatigue due to the repetitive stresses induced during takeoff, cruising, and landing phases.

1.1 Predictive Maintenance in Aerospace

Traditional maintenance strategies in aerospace include preventive and corrective approaches, both of which often incur significant downtime and cost. Predictive maintenance (PdM), on the other hand, uses data-driven models to forecast failures and plan interventions before a component fails [2]. This approach enhances operational efficiency, lowers maintenance costs, and improves flight safety.

A core component of predictive maintenance is the estimation of Remaining Useful Life (RUL), which refers to the amount of time or cycles a component can continue to function safely before it needs repair or replacement [3]. However, developing accurate RUL estimation models is challenging due to the complexity of fatigue behavior and the scarcity of failure data.

1.2 Key Concepts of the Study

This research investigates the integration of finite element simulations and machine learning techniques to predict fatigue damage in aeronautical structures. Below, we define and explain the central concepts and tools used in this study.

1.2.1 Fatigue Damage in Metallic Structures

Fatigue damage typically develops in three stages: crack initiation, crack propagation, and final fracture [4]. Each stage is influenced by factors such as load magnitude, load frequency, and material properties. Understanding and modeling these stages is crucial for accurate fatigue prognosis in aerospace engineering.

1.2.2 Finite Element Analysis (FEA) and COMSOL Multiphysics

Finite Element Analysis (FEA) is a powerful computational technique used to simulate how structures respond to physical effects such as force, heat, and vibration. In this study, we use COMSOL Multiphysics—an advanced FEA tool—to simulate crack behavior under tensile and cyclic loading [5]. These simulations help generate reliable synthetic data that mimics real-world fatigue scenarios.

1.2.3 Synthetic Data Generation Using FrameworkFDPP

Obtaining real fatigue datasets is often expensive and time-consuming. Therefore, synthetic data generated from simulations provides a practical alternative. This study is based on the FrameworkFDPP approach, which proposes a pipeline for creating large and diverse datasets for fatigue damage prognosis using virtual sensors and simulated strain measurements [6]. The synthetic data mimics the signals collected by strain gauges installed on aircraft structures.

1.3 Machine Learning Models for RUL Estimation

The original FrameworkFDPP paper explored several deep learning models for time-series prediction, including:

- Recurrent Neural Networks (RNN) [7]
- Long Short-Term Memory (LSTM) [8]
- Gated Recurrent Units (GRU) [9]
- 1D Convolutional Neural Networks (1D-CNN) [10]
- Temporal Convolutional Networks (TCN) [11]

In this study, the goal is to **explore alternative machine learning models** for the same RUL estimation task using the synthetic data generated via FrameworkFDPP. Specifically, we propose and evaluate the following:

- **XGBoost (Extreme Gradient Boosting)** – a scalable tree boosting system that has proven effective in many structured data problems [12].
- **LightGBM (Light Gradient Boosting Machine)** – a gradient boosting framework that uses tree-based learning and is optimized for speed and memory efficiency [13].
- **Autoencoders** – neural networks used for unsupervised learning of efficient codings and anomaly detection in time-series signals [14].

These models are trained and validated using the same dataset as the original models. Their performance will be compared against the baseline deep learning models presented in the original paper to evaluate whether simpler or more efficient models can achieve comparable results in fatigue RUL estimation.

1.4 Purpose of the Introduction

This chapter has provided the context and motivation for the research, defined key terms such as fatigue, RUL, and predictive maintenance, and introduced both the existing and proposed machine learning models. The next chapter will present a detailed literature review of similar work in fatigue prognosis and machine learning for structural health monitoring.

Chapter 2

Literature Review

This chapter reviews key literature related to fatigue damage prognosis in aeronautical structures, focusing on synthetic data generation methods and machine learning models for Remaining Useful Life (RUL) estimation. It also discusses the FrameworkFDPP paper, which serves as a foundation for this study, and outlines the distinctions between its approach and the methodology adopted in this research.

2.1 Fatigue Damage in Aeronautical Structures

Fatigue damage is a critical concern in aerospace engineering, as aircraft components are subjected to cyclic loading, leading to crack initiation and propagation over time. Understanding and predicting fatigue behavior are essential for ensuring structural integrity and safety. Traditional methods rely on empirical models and extensive testing, which can be time-consuming and costly.

2.2 Synthetic Data Generation for Fatigue Prognosis

Due to the challenges in obtaining large-scale, high-quality fatigue data from physical experiments, researchers have explored synthetic data generation methods. These approaches aim to create realistic datasets that can be used to train and validate prognostic models.

2.2.1 FrameworkFDPP: A Code-Based Synthetic Data Generation Approach

The FrameworkFDPP (Fatigue Damage Prognostics Problem) [15] presents a methodology for generating large synthetic datasets for fatigue damage prognosis. The framework utilizes a code-based approach to simulate strain gauge data, incorporating virtual sensor placement and crack growth modeling based on Paris' law. Key features include:

- **Virtual Strain Gauges**: Simulated placement of strain gauges on a virtual structure to collect strain data.
- **Crack Growth Simulation**: Implementation of Paris' law to model crack propagation over time.
- **Data Generation Pipeline**: Automated generation of training, validation, and test datasets for machine learning models.

This approach allows for the creation of extensive datasets without the need for physical experiments or finite element simulations.

2.2.2 Machine Learning Models for RUL Estimation

The synthetic datasets generated by FrameworkFDPP have been used to train various deep learning models for RUL estimation, including:

- **Recurrent Neural Networks (RNNs)**: Suitable for sequential data analysis.
- **Long Short-Term Memory (LSTM) Networks**: Address the vanishing gradient problem in RNNs.
- **Gated Recurrent Units (GRUs)**: Simplified version of LSTMs with comparable performance.
- **1D Convolutional Neural Networks (1D-CNNs)**: Effective for capturing local patterns in time-series data.

- **Temporal Convolutional Networks (TCNs)**: Utilize causal convolutions for sequence modeling.

These models have demonstrated promising results in predicting the RUL of components based on synthetic strain data.

2.3 Limitations of Code-Based Synthetic Data

While the FrameworkFDPP provides a scalable method for data generation, it relies on simplified assumptions and lacks the detailed physics captured by finite element simulations. The absence of high-fidelity modeling may limit the realism and applicability of the synthetic data to real-world scenarios.

2.4 Finite Element Analysis (FEA) for High-Fidelity Data Generation

Finite Element Analysis (FEA) offers a physics-based approach to simulate the structural behavior of materials under various loading conditions. Tools like COMSOL Multiphysics enable detailed modeling of crack initiation and propagation, providing more accurate strain data for prognostic modeling.

2.5 Proposed Methodology: Integrating FEA and Machine Learning

This research aims to enhance the credibility of synthetic data by:

- **Conducting FEA Simulations**: Using COMSOL Multiphysics to simulate crack growth and generate strain data.

- **Comparative Analysis**: Evaluating the consistency between FEA-generated data and FrameworkFDPP synthetic data.

- **Model Training**: Employing machine learning models such as XGBoost, LightGBM, and Autoencoders to predict RUL based on FEA data.
- **Performance Comparison**: Assessing the predictive accuracy of new models against those used in FrameworkFDPP.

By integrating high-fidelity simulations with advanced machine learning techniques, this study seeks to improve the reliability and applicability of fatigue damage prognostic models in aerospace engineering.

Chapter 3

Problem Statement and Objectives

3.1 Problem Statement

In recent years, the aerospace industry has increasingly emphasized the need for predictive maintenance strategies to ensure the structural integrity and safety of aircraft components subjected to cyclic loading. Fatigue-induced cracks are a major contributor to structural failures, and accurate prediction of the Remaining Useful Life (RUL) of these components is essential to prevent unexpected breakdowns.

The study by Akrim et al. (2022) introduced the *FrameworkFDPP*, which provides a synthetic dataset generation approach based on a virtual model that simulates strain data using placed virtual gauges and Paris' Law to model crack propagation. While this framework is valuable for generating large volumes of training data without physical testing or finite element simulations, the synthetic data is not physically validated against high-fidelity structural simulations or real-world experiments [15].

This lack of physical grounding raises questions about the realism and credibility of the synthetic data when applied to real-world fatigue damage scenarios. Additionally, the deep learning models used in the original framework—such as RNN, LSTM, GRU, 1D-CNN, and TCN—were not compared to newer machine learning algorithms that might offer improved accuracy and interpretability.

3.2 Research Gap

The following gaps have been identified from the literature:

- Lack of comparison between synthetic data from FrameworkFDPP and high-fidelity finite element simulation data.
- No validation of virtual strain data against physically simulated strain results.
- Limited exploration of non-deep learning models such as gradient boosting or unsupervised approaches like Autoencoders in the context of fatigue RUL estimation.

3.3 Research Objectives

This thesis aims to bridge the gap between code-based synthetic data and physics-based simulation in fatigue damage prognosis. The specific objectives are:

1. **Simulate crack propagation using COMSOL Multiphysics:** Generate strain data from a 2D finite element model under fatigue loading to mimic realistic crack behavior.
2. **Compare synthetic vs. simulated strain data:** Evaluate the credibility of the FrameworkFDPP's virtual strain gauge data by comparing it to data from COMSOL simulations.
3. **Develop and test new models:** Implement and train alternative machine learning models (e.g., XGBoost, LightGBM, Autoencoder) on dataset.
4. **Benchmark model performance:** Assess and compare the performance of these models using RUL prediction accuracy metrics such as RMSE, MAE, and R^2, and compare them to results from the original FrameworkFDPP study.

3.4 Scope of the Study

This study will focus on metallic aircraft structures subjected to tensile and fatigue loading. The COMSOL simulations will model crack growth and strain accumulation, while the data from FrameworkFDPP will be used to train and evaluate predictive models.

Chapter 4

Methodology

The methodological workflow shown in Figure 4.1 consists of the following key stages:

- **Start:** The project begins by defining the objectives—predicting crack growth and estimating Remaining Useful Life (RUL) using simulated data and machine learning techniques.

- **COMSOL Simulation:** A 2D fatigue crack growth model is built in COMSOL Multiphysics. Strain gauges are positioned around the crack tip, and cyclic loading is applied to simulate the physical environment.

- **Generate Synthetic Data:** Inspired by the FDPP framework, synthetic datasets are generated to mimic strain and displacement responses under fatigue loading. These datasets help in training models without relying solely on costly physical experiments.

- **Preprocess and Clean Data:** The raw simulation outputs are cleaned, structured, and filtered. This includes removing outliers, handling missing values, and ensuring consistent formatting of features like strain and crack length.

- **Normalize and Scale Features:** All features are normalized using MinMax scaling or similar techniques. This improves the training stability of neural networks and prevents bias toward larger magnitude features.

- **Train ML/DL Models:** Various machine learning and deep learning models—including GRU, XGBoost, and Transformers—are trained to predict either crack length or RUL. Performance is tracked using error metrics like MAPE.
- **Evaluate Performance (MAPE):** The trained models are evaluated using validation and test sets. Mean Absolute Percentage Error (MAPE) is computed to compare accuracy across models and training sizes.
- **Build User Interface (GUI):** A graphical interface is designed (using Tkinter) to allow maintenance technicians to input current cycle or strain values and receive predictions on crack length or RUL instantly.
- **Integrate Model and GUI:** The best-performing trained model is embedded into the GUI, enabling real-time inference and user interaction.
- **Deployment:** The integrated solution is prepared for practical use. This may include packaging, documentation, and testing the system under realistic use scenarios.

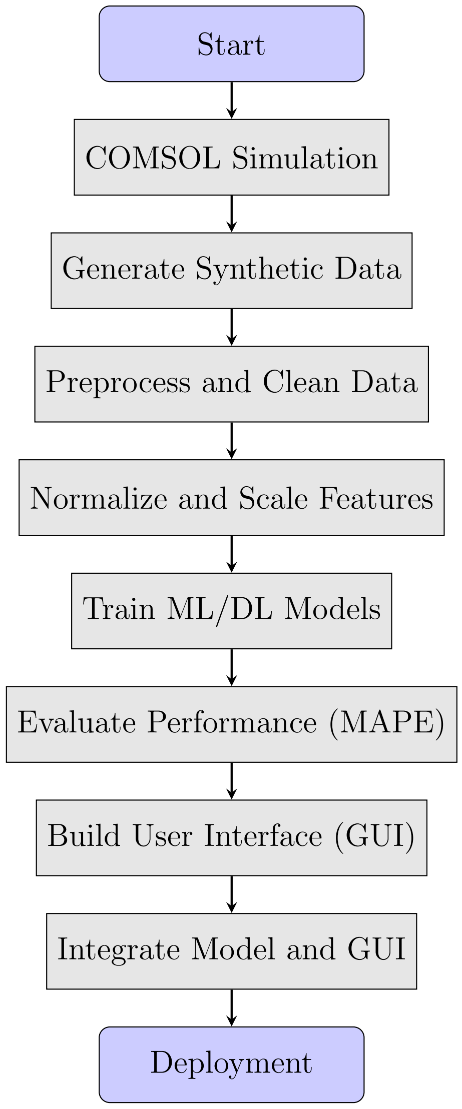

Start
COMSOL Simulation
Generate Synthetic Data
Preprocess and Clean Data
Normalize and Scale Features
Train ML/DL Models
Evaluate Performance (MAPE)
Build User Interface (GUI)
Integrate Model and GUI
Deployment

Figure 4.1: Overall methodological workflow of the project

4.1 Crack Simulation in COMSOL

In this section, we describe the numerical simulation approach used to replicate crack propagation behavior in a 2D structural model using COMSOL Multiphysics®. The goal is to generate strain data that can later be compared to synthetically generated data and used to train predictive models.

4.1.1 Simulation and study assumptions :

- the fuselage panels are representative of short range aircraft such as A320 or B737.
- one cycle is one landing and one take off which takes an average of 1 hour = 3600 second .

4.1.2 Model Geometry and Boundary Conditions

A 2D rectangular plate with dimensions 200 mm (width) × 150 mm (height) was created in COMSOL. A pre-existing elliptical crack of length 2 mm was introduced at the center of the geometry to simulate an initial flaw.

The following boundary conditions were applied:

- Left and Right Edges (x = 0 and x = L) These simulate panel being riveted or attached to structure.

  Use: Prescribed Displacement

  ux = 0

  uy = free (not constrained)

  This means it can expand/contract vertically but can't move left/right.

- A cyclic load in the form of $-78.6 \times 10^6 \cdot \sin\left(\frac{2\pi t}{3600}\right)$ was applied on the top and bottom edges in opposite directions, simulating a realistic alternating fatigue loading over time.
- 3 probe points that represents the strain gauges (sensors) where : strain gauge 1 (3 , 14) , strain gauge 1 (14 , 14) , strain gauge 1 (25 , 14).
- 2 probe points on the edge and in the tip of the crack .
- Aluminum alloy 7075-T6 is used as material for the geometry .

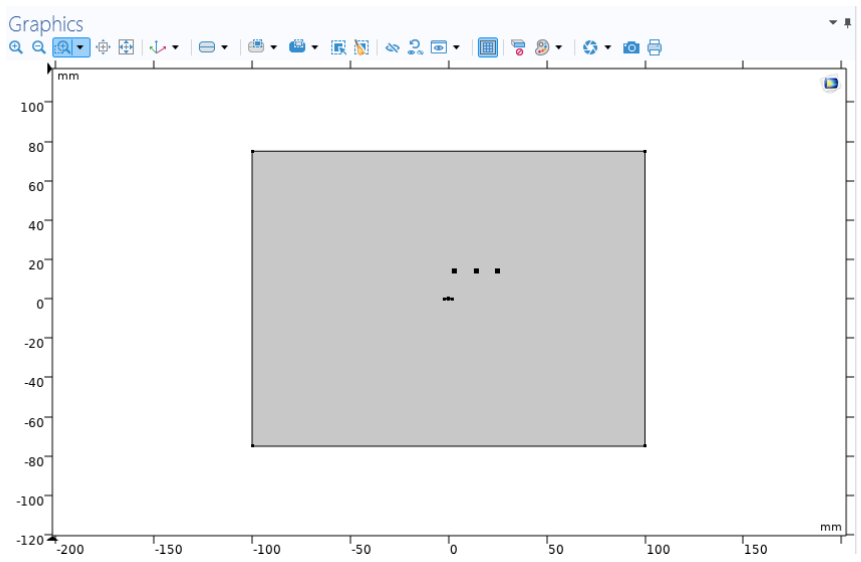


Figure 4.2: 2D geometry of the cracked plate in COMSOL with probe point locations

4.1.3 Simulation Setup and Parameters

The simulation was configured to run for 100 loading cycles, with each cycle lasting 3600 seconds, totaling 360,000 seconds of simulation time. Due to computational limitations and the high-resolution nature of the FEM analysis, this simulation was very time-consuming. then we ran another simulation starting from 100 cycles to 1000 cycle and it took around 24h to export the results .

Virtual Strain Gauges and Data Extraction

Virtual strain gauges were placed at specific probe points, as outlined in the FDPP paper [16], to monitor strain evolution over time. Two additional probes were placed: one at the start and one at the end of the crack, to track crack length growth.

The output from the COMSOL simulation was exported as a CSV file containing the strain and displacement values over time at each probe location.

Post-Processing of COMSOL Data

To analyze the results of the COMSOL simulations, the strain evolution data were extracted and plotted against the number of cycles using Python. Two separate simulations

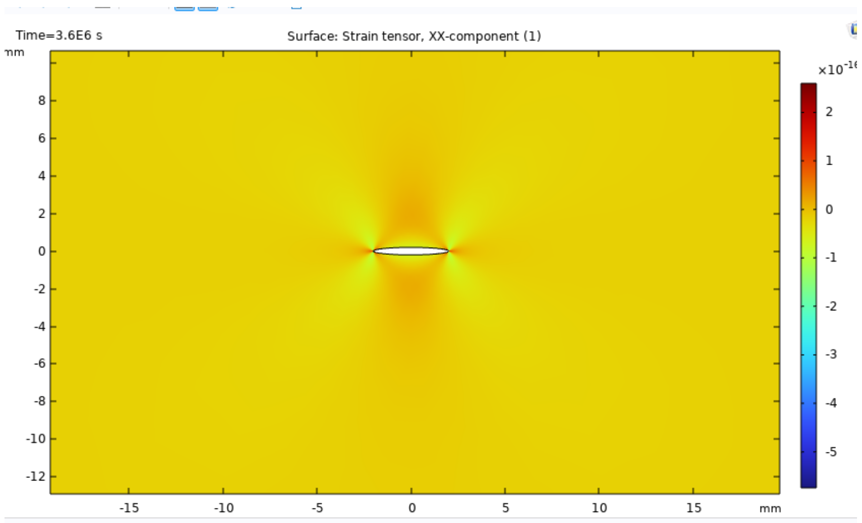


Figure 4.3: normal strains state for a crack of length 2mm

were conducted: one for 100 cycles and another for 1000 cycles, both using identical boundary conditions and virtual strain gauge configurations.

The Python scripts used for this processing — `comsol100.py` and `comsol1000.py` — are available in the GitHub repository linked in Appendix .1. These scripts were responsible for:

- Parsing and processing the COMSOL output CSV files (strain and displacement data).
- Extracting cycle-wise strain readings at all probe points, including those located at the crack tips and virtual strain gauges.
- Plotting critical graphs such as strain vs. cycle and crack length vs. cycle number.

Results from 100-Cycle Simulation: in The 100-cycle simulation meaning after 100 cycles assuming that each cycle takes 1 hour , theres no significant change in the crack length also the starin so that's why there's a need to lanch another simulation till 1000 cylces .

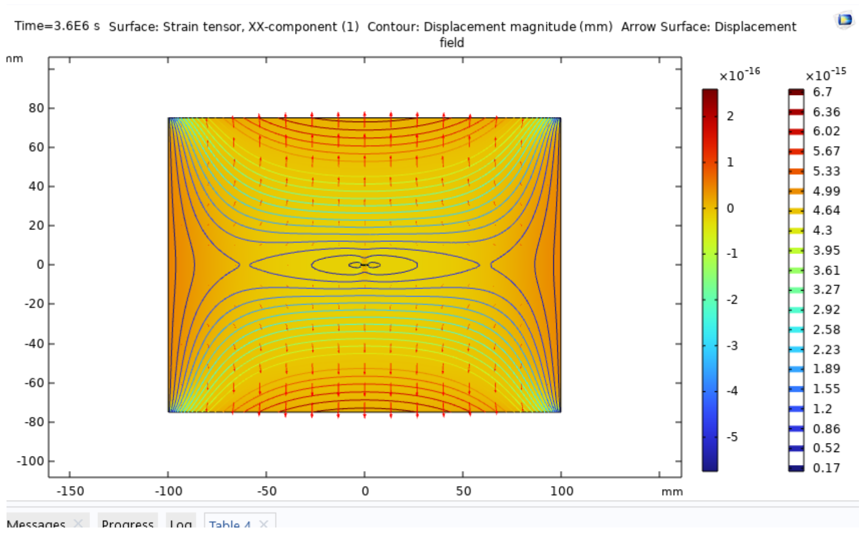


Figure 4.4: cyclic load applied

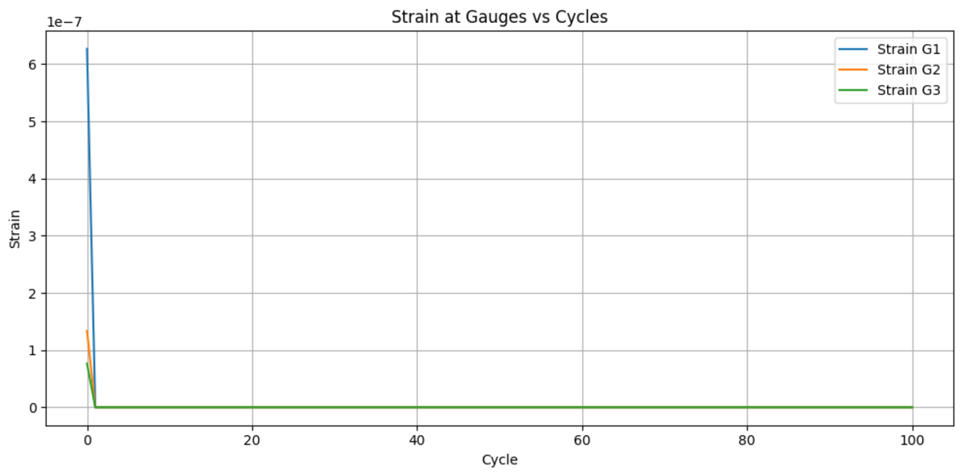
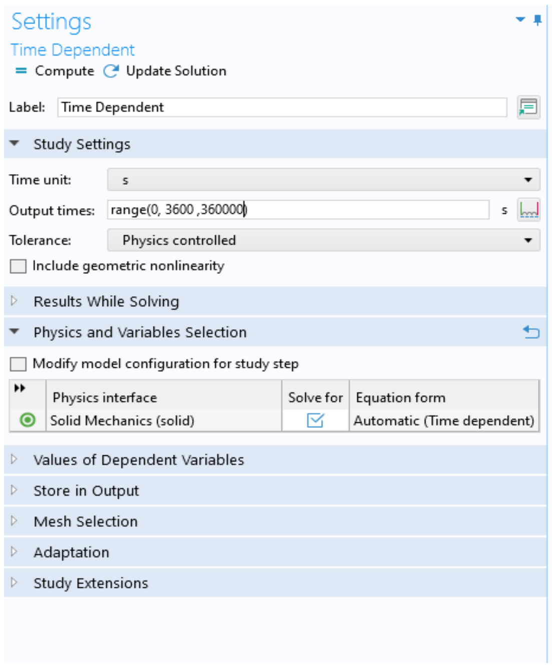


Figure 4.8: Strain vs. Cycle Number for 3 Virtual Gauges (100-cycle COMSOL simulation)


Settings
Time Dependent
= Compute ⟳ Update Solution

Label: Time Dependent

Study Settings

Time unit: s

Output times: range(0, 3600 ,360000) s

Tolerance: Physics controlled

☐ Include geometric nonlinearity

Results While Solving

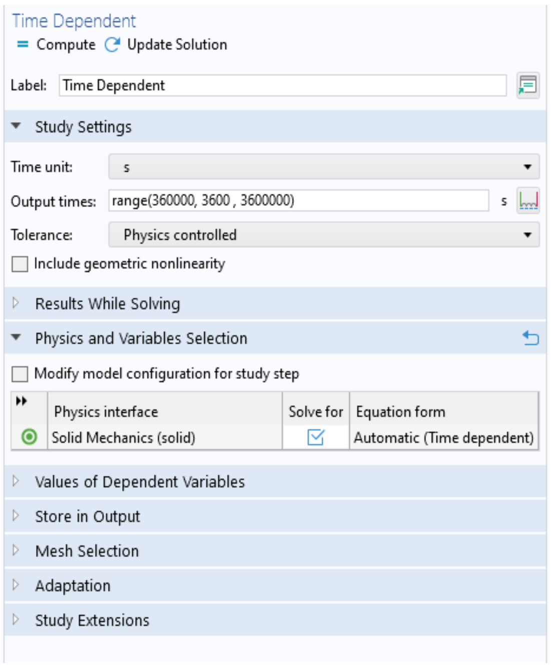
Physics and Variables Selection

☐ Modify model configuration for study step

Physics interface	Solve for	Equation form
Solid Mechanics (solid)	✓	Automatic (Time dependent)

Values of Dependent Variables
Store in Output
Mesh Selection
Adaptation
Study Extensions

(a) Simulating from cycle 1 to cycle 100


Time Dependent
= Compute ⟳ Update Solution

Label: Time Dependent

Study Settings

Time unit: s

Output times: range(360000, 3600 , 3600000) s

Tolerance: Physics controlled

☐ Include geometric nonlinearity

Results While Solving

Physics and Variables Selection

☐ Modify model configuration for study step

Physics interface	Solve for	Equation form
Solid Mechanics (solid)	✓	Automatic (Time dependent)

Values of Dependent Variables
Store in Output
Mesh Selection
Adaptation
Study Extensions

(b) Simulating from cycle 100 to cycle 1000

Figure 4.5: Simulation comparisons across cycles

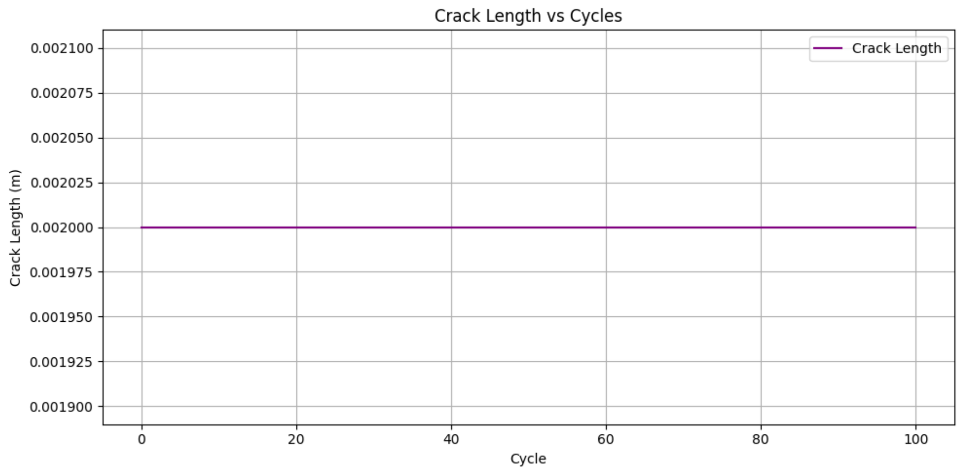
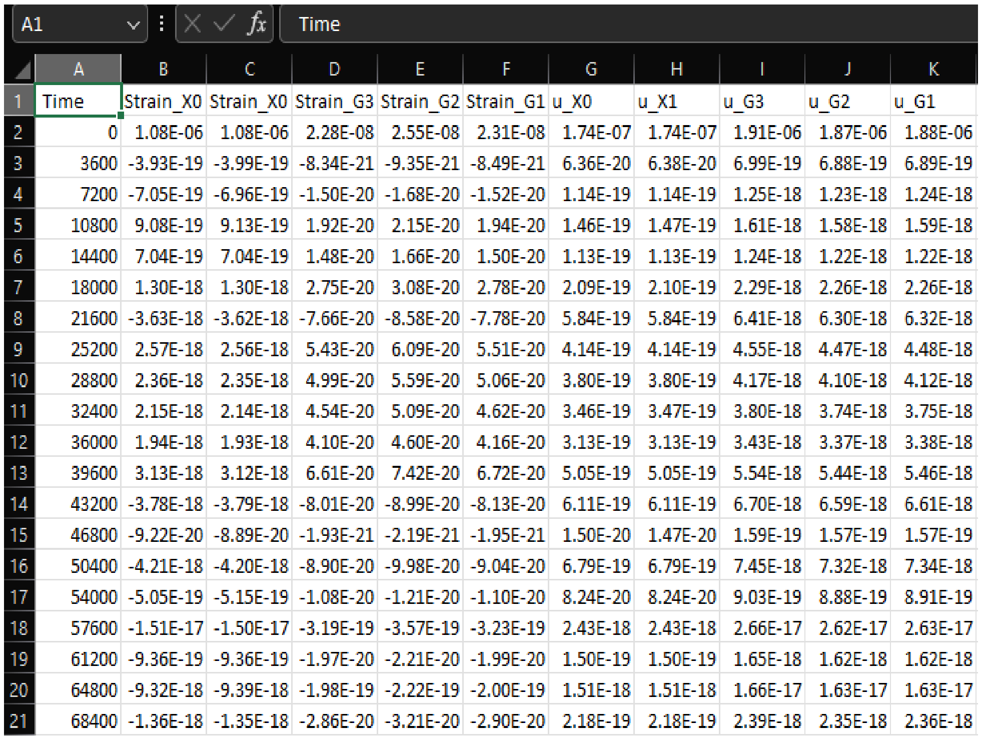


Figure 4.9: Crack Length vs. Cycle Number (100-cycle COMSOL simulation)

Results from 1000-Cycle Simulation: The 1000-cycle simulation took more time but still there's no significant change in the crack length .



A1 Time

	A	B	C	D	E	F	G	H	I	J	K
1	Time	Strain_X0	Strain_X0	Strain_G3	Strain_G2	Strain_G1	u_X0	u_X1	u_G3	u_G2	u_G1
2	0	1.08E-06	1.08E-06	2.28E-08	2.55E-08	2.31E-08	1.74E-07	1.74E-07	1.91E-06	1.87E-06	1.88E-06
3	3600	-3.93E-19	-3.99E-19	-8.34E-21	-9.35E-21	-8.49E-21	6.36E-20	6.38E-20	6.99E-19	6.88E-19	6.89E-19
4	7200	-7.05E-19	-6.96E-19	-1.50E-20	-1.68E-20	-1.52E-20	1.14E-19	1.14E-19	1.25E-18	1.23E-18	1.24E-18
5	10800	9.08E-19	9.13E-19	1.92E-20	2.15E-20	1.94E-20	1.46E-19	1.47E-19	1.61E-18	1.58E-18	1.59E-18
6	14400	7.04E-19	7.04E-19	1.48E-20	1.66E-20	1.50E-20	1.13E-19	1.13E-19	1.24E-18	1.22E-18	1.22E-18
7	18000	1.30E-18	1.30E-18	2.75E-20	3.08E-20	2.78E-20	2.09E-19	2.10E-19	2.29E-18	2.26E-18	2.26E-18
8	21600	-3.63E-18	-3.62E-18	-7.66E-20	-8.58E-20	-7.78E-20	5.84E-19	5.84E-19	6.41E-18	6.30E-18	6.32E-18
9	25200	2.57E-18	2.56E-18	5.43E-20	6.09E-20	5.51E-20	4.14E-19	4.14E-19	4.55E-18	4.47E-18	4.48E-18
10	28800	2.36E-18	2.35E-18	4.99E-20	5.59E-20	5.06E-20	3.80E-19	3.80E-19	4.17E-18	4.10E-18	4.12E-18
11	32400	2.15E-18	2.14E-18	4.54E-20	5.09E-20	4.62E-20	3.46E-19	3.47E-19	3.80E-18	3.74E-18	3.75E-18
12	36000	1.94E-18	1.93E-18	4.10E-20	4.60E-20	4.16E-20	3.13E-19	3.13E-19	3.43E-18	3.37E-18	3.38E-18
13	39600	3.13E-18	3.12E-18	6.61E-20	7.42E-20	6.72E-20	5.05E-19	5.05E-19	5.54E-18	5.44E-18	5.46E-18
14	43200	-3.78E-18	-3.79E-18	-8.01E-20	-8.99E-20	-8.13E-20	6.11E-19	6.11E-19	6.70E-18	6.59E-18	6.61E-18
15	46800	-9.22E-20	-8.89E-20	-1.93E-21	-2.19E-21	-1.95E-21	1.50E-20	1.47E-20	1.59E-19	1.57E-19	1.57E-19
16	50400	-4.21E-18	-4.20E-18	-8.90E-20	-9.98E-20	-9.04E-20	6.79E-19	6.79E-19	7.45E-18	7.32E-18	7.34E-18
17	54000	-5.05E-19	-5.15E-19	-1.08E-20	-1.21E-20	-1.10E-20	8.24E-20	8.24E-20	9.03E-19	8.88E-19	8.91E-19
18	57600	-1.51E-17	-1.50E-17	-3.19E-19	-3.57E-19	-3.23E-19	2.43E-18	2.43E-18	2.66E-17	2.62E-17	2.63E-17
19	61200	-9.36E-19	-9.36E-19	-1.97E-20	-2.21E-20	-1.99E-20	1.50E-19	1.50E-19	1.65E-18	1.62E-18	1.62E-18
20	64800	-9.32E-18	-9.39E-18	-1.98E-19	-2.22E-19	-2.00E-19	1.51E-18	1.51E-18	1.66E-17	1.63E-17	1.63E-17
21	68400	-1.36E-18	-1.35E-18	-2.86E-20	-3.21E-20	-2.90E-20	2.18E-19	2.18E-19	2.39E-18	2.35E-18	2.36E-18

Figure 4.6: CSV file from comsol simulation of 100 cycle

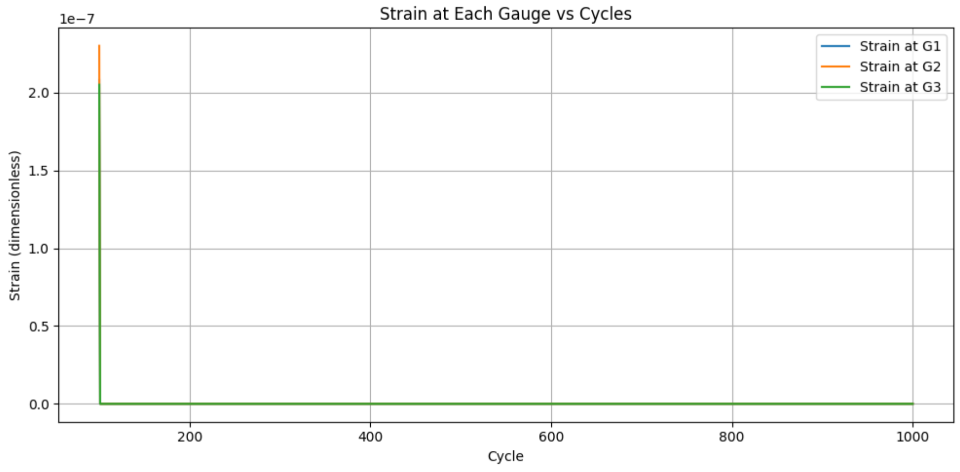
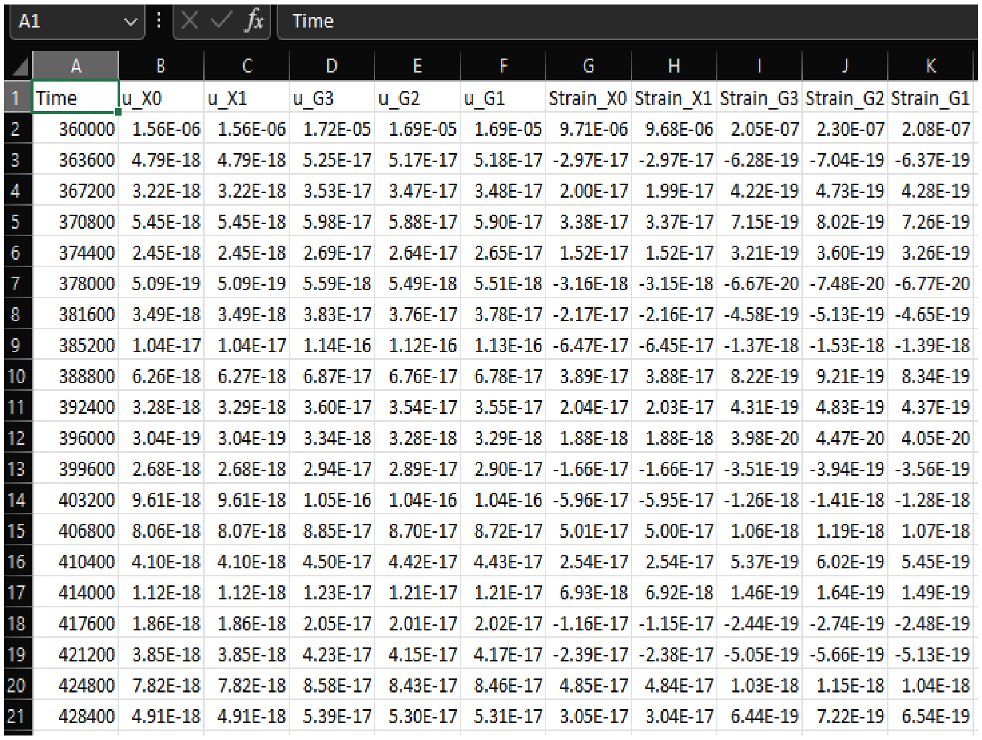


Figure 4.10: Strain vs. Cycle Number for 3 Virtual Gauges (1000-cycle COMSOL simulation)



Time	u_X0	u_X1	u_G3	u_G2	u_G1	Strain_X0	Strain_X1	Strain_G3	Strain_G2	Strain_G1
360000	1.56E-06	1.56E-06	1.72E-05	1.69E-05	1.69E-05	9.71E-06	9.68E-06	2.05E-07	2.30E-07	2.08E-07
363600	4.79E-18	4.79E-18	5.25E-17	5.17E-17	5.18E-17	-2.97E-17	-2.97E-17	-6.28E-19	-7.04E-19	-6.37E-19
367200	3.22E-18	3.22E-18	3.53E-17	3.47E-17	3.48E-17	2.00E-17	1.99E-17	4.22E-19	4.73E-19	4.28E-19
370800	5.45E-18	5.45E-18	5.98E-17	5.88E-17	5.90E-17	3.38E-17	3.37E-17	7.15E-19	8.02E-19	7.26E-19
374400	2.45E-18	2.45E-18	2.69E-17	2.64E-17	2.65E-17	1.52E-17	1.52E-17	3.21E-19	3.60E-19	3.26E-19
378000	5.09E-19	5.09E-19	5.59E-18	5.49E-18	5.51E-18	-3.16E-18	-3.15E-18	-6.67E-20	-7.48E-20	-6.77E-20
381600	3.49E-18	3.49E-18	3.83E-17	3.76E-17	3.78E-17	-2.17E-17	-2.16E-17	-4.58E-19	-5.13E-19	-4.65E-19
385200	1.04E-17	1.04E-17	1.14E-16	1.12E-16	1.13E-16	-6.47E-17	-6.45E-17	-1.37E-18	-1.53E-18	-1.39E-18
388800	6.26E-18	6.27E-18	6.87E-17	6.76E-17	6.78E-17	3.89E-17	3.88E-17	8.22E-19	9.21E-19	8.34E-19
392400	3.28E-18	3.29E-18	3.60E-17	3.54E-17	3.55E-17	2.04E-17	2.03E-17	4.31E-19	4.83E-19	4.37E-19
396000	3.04E-19	3.04E-19	3.34E-18	3.28E-18	3.29E-18	1.88E-18	1.88E-18	3.98E-20	4.47E-20	4.05E-20
399600	2.68E-18	2.68E-18	2.94E-17	2.89E-17	2.90E-17	-1.66E-17	-1.66E-17	-3.51E-19	-3.94E-19	-3.56E-19
403200	9.61E-18	9.61E-18	1.05E-16	1.04E-16	1.04E-16	-5.96E-17	-5.95E-17	-1.26E-18	-1.41E-18	-1.28E-18
406800	8.06E-18	8.07E-18	8.85E-17	8.70E-17	8.72E-17	5.01E-17	5.00E-17	1.06E-18	1.19E-18	1.07E-18
410400	4.10E-18	4.10E-18	4.50E-17	4.42E-17	4.43E-17	2.54E-17	2.54E-17	5.37E-19	6.02E-19	5.45E-19
414000	1.12E-18	1.12E-18	1.23E-17	1.21E-17	1.21E-17	6.93E-18	6.92E-18	1.46E-19	1.64E-19	1.49E-19
417600	1.86E-18	1.86E-18	2.05E-17	2.01E-17	2.02E-17	-1.16E-17	-1.15E-17	-2.44E-19	-2.74E-19	-2.48E-19
421200	3.85E-18	3.85E-18	4.23E-17	4.15E-17	4.17E-17	-2.39E-17	-2.38E-17	-5.05E-19	-5.66E-19	-5.13E-19
424800	7.82E-18	7.82E-18	8.58E-17	8.43E-17	8.46E-17	4.85E-17	4.84E-17	1.03E-18	1.15E-18	1.04E-18
428400	4.91E-18	4.91E-18	5.39E-17	5.30E-17	5.31E-17	3.05E-17	3.04E-17	6.44E-19	7.22E-19	6.54E-19

Figure 4.7: CSV file from comsol simulation of 1000 cycle

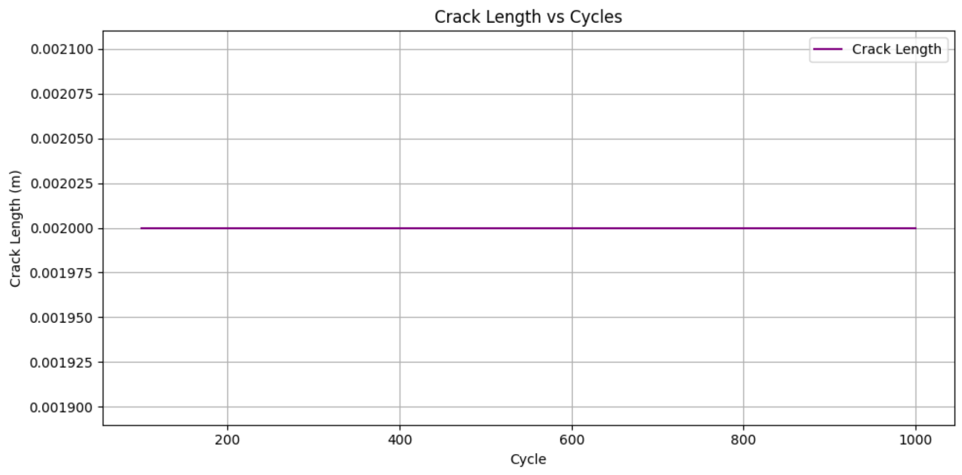


Figure 4.11: Crack Length vs. Cycle Number (1000-cycle COMSOL simulation)

Strain Stabilization Observed in Early Cycles

The strain response obtained from the COMSOL simulation over the first 100 cycles shows a sharp initial decrease in strain measured at the three strain gauges, followed by a stabilization at very small but non-zero values (approximately on the order of 10^{-10}). This reflects the typical elastic adjustment of the structure under cyclic loading. Initially, the material experiences transient effects as it accommodates the applied sinusoidal stress. Once equilibrium is reached, the strain levels settle into a steady-state regime. Since no damage or crack propagation was allowed in this simulation phase, the strain remains stable for the rest of the cycles. This result provides a meaningful validation baseline, indicating that the structure behaves elastically over the early life of loading — contrasting with the synthetic dataset, where strain continues to evolve due to modeled fatigue progression.

4.1.4 Parameter Sensitivity Study

A sensitivity study was conducted to determine how different simulation settings affect the accuracy and stability of the results:

- **Mesh Refinement:** A fine mesh was applied around the crack tip to capture local stress concentration.
- **Time Step:** Various cycle step values (1, 10, 50 cycles) were tested to observe their influence on the results.
- **Material Properties:** Sensitivity to variations in E and ν was analyzed.

This analysis allowed us to identify the optimal setup to balance simulation accuracy and computational cost.

4.1.5 Stress and Strain Field Analysis

The simulation results show that the stress field is highly concentrated near the crack tip, consistent with fracture mechanics theory. After 100 cycles, the crack length remains mostly static, while at 1,000 cycles, the crack shows noticeable propagation.

The strain field also reflects this behavior with a progressive increase near the crack tip over time.

4.2 Crack Simulation with Synthetic Data

To overcome the computational cost and time constraints of simulating fatigue crack growth through finite element software such as COMSOL, this project adopts a data-driven approach to generate large-scale synthetic datasets using the open-source **FrameworkFDPP** available on GitHub .1

4.2.1 Overview of the Framework

FrameworkFDPP is a Python-based framework designed to simulate the progression of fatigue cracks in metallic structures by generating large datasets under varying physical and structural conditions. It leverages fracture mechanics theory, particularly the **Paris Law**, to model crack length evolution over time and across various structural parameters.

4.2.2 The Paris Law

The Paris Law is an empirical relationship used in fracture mechanics to describe the rate of crack growth per load cycle in materials subjected to cyclic loading. It is mathematically expressed as:

$$\frac{da}{dN} = C \cdot (\Delta K)^m$$

Where:

- $\frac{da}{dN}$ is the crack growth rate per cycle.
- ΔK is the stress intensity factor range.
- C and m are material constants (experimentally determined).

This law is embedded in the synthetic data generation process, allowing the simulation of realistic crack evolution.

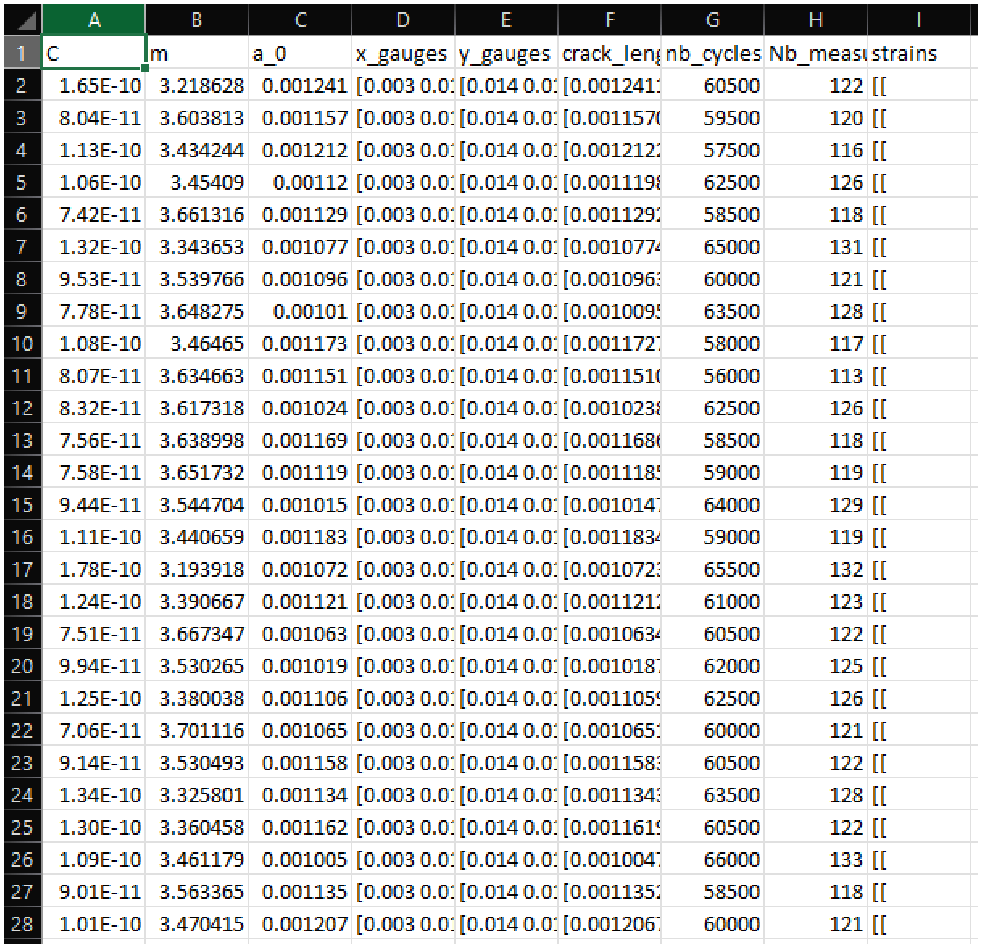
4.2.3 How the Synthetic Data Is Generated

The main Python script responsible for generating the synthetic dataset is `generate_dataset.py`. This script performs the following tasks:

1. Defines the geometry of the plate and locations of the strain gauges.
2. Samples key parameters such as initial crack length a_0, Paris constants C and m, and loading conditions.
3. Simulates crack propagation over time using the Paris Law.
4. Computes the strain fields at predefined gauge positions using an analytical approximation (not FEA).
5. Records the crack length, number of cycles, and gauge strains at each time step.
6. Outputs the data in structured CSV files for training, validation, and testing purposes.

4.2.4 Synthetic Training Set Parameters

After generateing the data a csv file is saved that look like this :



	A	B	C	D	E	F	G	H	I
1	C	m	a_0	x_gauges	y_gauges	crack_len	nb_cycles	Nb_meas	strains
2	1.65E-10	3.218628	0.001241	[0.003 0.0	[0.014 0.0	[0.001241	60500	122	[[
3	8.04E-11	3.603813	0.001157	[0.003 0.0	[0.014 0.0	[0.001157	59500	120	[[
4	1.13E-10	3.434244	0.001212	[0.003 0.0	[0.014 0.0	[0.001212	57500	116	[[
5	1.06E-10	3.45409	0.00112	[0.003 0.0	[0.014 0.0	[0.001119	62500	126	[[
6	7.42E-11	3.661316	0.001129	[0.003 0.0	[0.014 0.0	[0.001129	58500	118	[[
7	1.32E-10	3.343653	0.001077	[0.003 0.0	[0.014 0.0	[0.001077	65000	131	[[
8	9.53E-11	3.539766	0.001096	[0.003 0.0	[0.014 0.0	[0.001096	60000	121	[[
9	7.78E-11	3.648275	0.00101	[0.003 0.0	[0.014 0.0	[0.001009	63500	128	[[
10	1.08E-10	3.46465	0.001173	[0.003 0.0	[0.014 0.0	[0.001172	58000	117	[[
11	8.07E-11	3.634663	0.001151	[0.003 0.0	[0.014 0.0	[0.001151	56000	113	[[
12	8.32E-11	3.617318	0.001024	[0.003 0.0	[0.014 0.0	[0.001023	62500	126	[[
13	7.56E-11	3.638998	0.001169	[0.003 0.0	[0.014 0.0	[0.001168	58500	118	[[
14	7.58E-11	3.651732	0.001119	[0.003 0.0	[0.014 0.0	[0.001118	59000	119	[[
15	9.44E-11	3.544704	0.001015	[0.003 0.0	[0.014 0.0	[0.001014	64000	129	[[
16	1.11E-10	3.440659	0.001183	[0.003 0.0	[0.014 0.0	[0.001183	59000	119	[[
17	1.78E-10	3.193918	0.001072	[0.003 0.0	[0.014 0.0	[0.001072	65500	132	[[
18	1.24E-10	3.390667	0.001121	[0.003 0.0	[0.014 0.0	[0.001121	61000	123	[[
19	7.51E-11	3.667347	0.001063	[0.003 0.0	[0.014 0.0	[0.001063	60500	122	[[
20	9.94E-11	3.530265	0.001019	[0.003 0.0	[0.014 0.0	[0.001018	62000	125	[[
21	1.25E-10	3.380038	0.001106	[0.003 0.0	[0.014 0.0	[0.001105	62500	126	[[
22	7.06E-11	3.701116	0.001065	[0.003 0.0	[0.014 0.0	[0.001065	60000	121	[[
23	9.14E-11	3.530493	0.001158	[0.003 0.0	[0.014 0.0	[0.001158	60500	122	[[
24	1.34E-10	3.325801	0.001134	[0.003 0.0	[0.014 0.0	[0.001134	63500	128	[[
25	1.30E-10	3.360458	0.001162	[0.003 0.0	[0.014 0.0	[0.001161	60500	122	[[
26	1.09E-10	3.461179	0.001005	[0.003 0.0	[0.014 0.0	[0.001004	66000	133	[[
27	9.01E-11	3.563365	0.001135	[0.003 0.0	[0.014 0.0	[0.001135	58500	118	[[
28	1.01E-10	3.470415	0.001207	[0.003 0.0	[0.014 0.0	[0.001206	60000	121	[[

Figure 4.12: syntatic data set

The synthetic dataset used in this work was generated following the methodology in [17]. The table below summarizes the key parameters.

Parameter	Description
`C`	Paris law coefficient (material-dependent constant that governs the crack growth rate).
`m`	Paris law exponent (determines the sensitivity of crack growth rate to stress intensity).
`a_0`	Initial crack length in meters.
`x_gauges`	X-positions of the three strain gauges used for measurement.
`y_gauges`	Y-positions of the three strain gauges used for measurement.
`crack_lengths`	List of crack lengths at each cycle, computed using Paris' law.
`nb_cycles`	Number of loading cycles until the crack reaches critical size.
`Nb_measures`	Number of strain measurements taken per sample.
`strains`	Strain readings at the three gauges for each cycle. Stored as 2D arrays.

Table 4.1: Description of Parameters in the Synthetic Training Set

4.2.5 Cleaning the syntatic data and generate the graphs to compare with comsol simulation :

To analyze and prepare the synthetic dataset for comparison with COMSOL simulation results, we used the `syntatic_study.py` script from our GitHub repository .1.

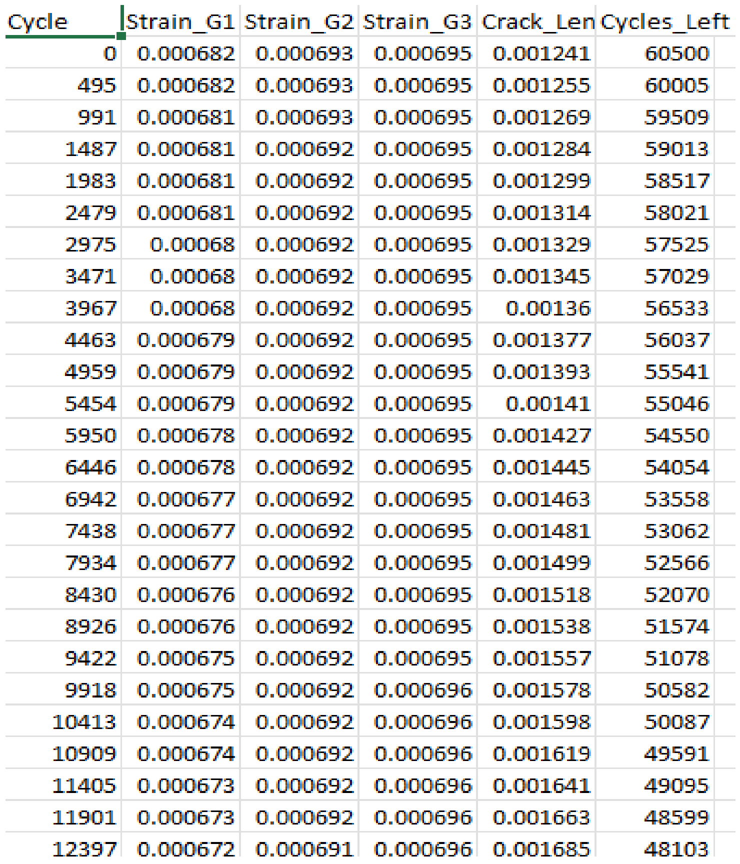
This script processes the raw synthetic training dataset (`training_set2.csv`) generated using the FrameworkFDPP method [6]. The processing steps include:

- Parsing and cleaning the embedded strain arrays for each of the three gauges.
- Extracting strain evolution across the fatigue cycles for each gauge.

- Extracting the corresponding crack length evolution per cycle.
- Calculating and plotting:
  - Strain vs. Cycles for each gauge (G1, G2, G3).
  - Crack Length vs. Cycles curve.

the new dataset used to generate the new graphs 4.13:

(`training_set_clean_ready_realcycles.csv`) .1



Cycle	Strain_G1	Strain_G2	Strain_G3	Crack_Len	Cycles_Left
0	0.000682	0.000693	0.000695	0.001241	60500
495	0.000682	0.000693	0.000695	0.001255	60005
991	0.000681	0.000693	0.000695	0.001269	59509
1487	0.000681	0.000692	0.000695	0.001284	59013
1983	0.000681	0.000692	0.000695	0.001299	58517
2479	0.000681	0.000692	0.000695	0.001314	58021
2975	0.00068	0.000692	0.000695	0.001329	57525
3471	0.00068	0.000692	0.000695	0.001345	57029
3967	0.00068	0.000692	0.000695	0.00136	56533
4463	0.000679	0.000692	0.000695	0.001377	56037
4959	0.000679	0.000692	0.000695	0.001393	55541
5454	0.000679	0.000692	0.000695	0.00141	55046
5950	0.000678	0.000692	0.000695	0.001427	54550
6446	0.000678	0.000692	0.000695	0.001445	54054
6942	0.000677	0.000692	0.000695	0.001463	53558
7438	0.000677	0.000692	0.000695	0.001481	53062
7934	0.000677	0.000692	0.000695	0.001499	52566
8430	0.000676	0.000692	0.000695	0.001518	52070
8926	0.000676	0.000692	0.000695	0.001538	51574
9422	0.000675	0.000692	0.000695	0.001557	51078
9918	0.000675	0.000692	0.000696	0.001578	50582
10413	0.000674	0.000692	0.000696	0.001598	50087
10909	0.000674	0.000692	0.000696	0.001619	49591
11405	0.000673	0.000692	0.000696	0.001641	49095
11901	0.000673	0.000692	0.000696	0.001663	48599
12397	0.000672	0.000691	0.000696	0.001685	48103

Figure 4.13: syntatic data set after cleaning

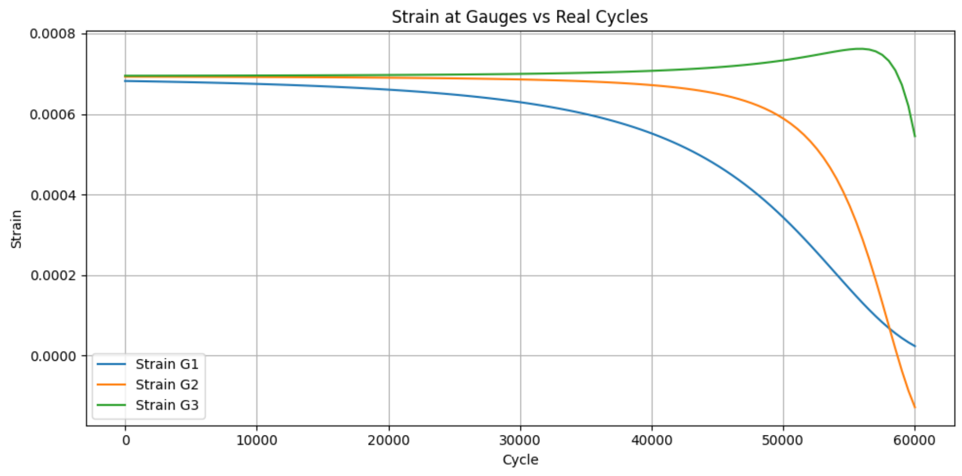


Figure 4.14: Strain vs. Cycle Number for 3 Virtual Gauges (syntatic dataset)

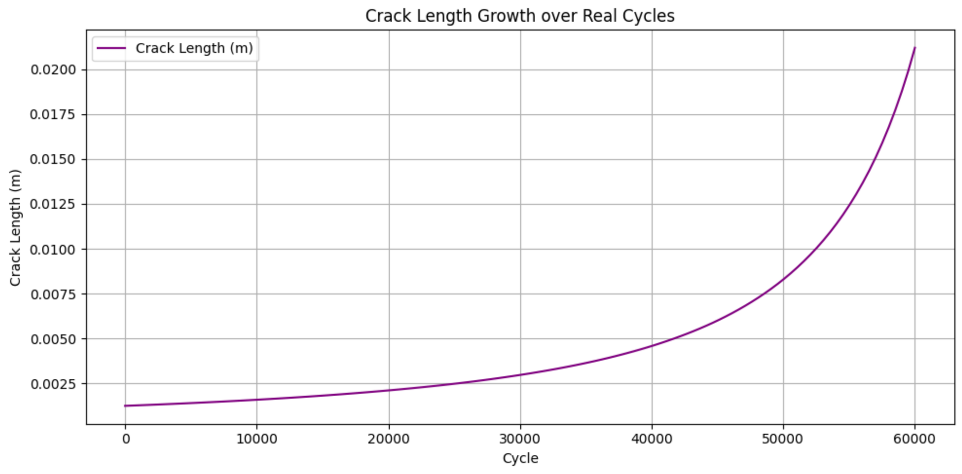


Figure 4.15: Crack Length vs. Cycle Number (syntatic dataset)

4.3 Analysis and Comparison of Synthetic Data

To assess the reliability of the synthetically generated fatigue data, we compared it against simulation results obtained using COMSOL Multiphysics. This comparison focused on two key metrics: crack length evolution and strain response at gauge points over fatigue cycles.

4.3.1 Crack Length vs. Cycles

The simulation performed in COMSOL, both for Figures 4.9 and 4.11 showed that the crack length remained constant during the first 100 and 1000 cycles. This behavior aligns closely with the synthetic dataset, where the crack length also remains static up to around 10,000 cycles before beginning to increase progressively due to fatigue accumulation like shown in figure 4.15. This matching pattern confirms that the synthetic dataset captures realistic initial damage accumulation phases, validating its use for fatigue prognosis beyond the limits of physical simulation.

4.3.2 Strain at Gauges vs. Cycles

Similarly, in the COMSOL simulation, the strain response measured at the three strain gauges remains nearly constant across the 100 and 1000 cycle simulations like shown in figures 4.8 and 4.10. This behavior is also observed in the synthetic dataset for the same early range (0 to 1000 cycles), where strain values appear nearly static (figure 4.14) . This consistency suggests that the synthetic strain gauge behavior replicates the early-cycle mechanical response of the aircraft panel as realistically as COMSOL allows under practical computational constraints.

4.3.3 Conclusion

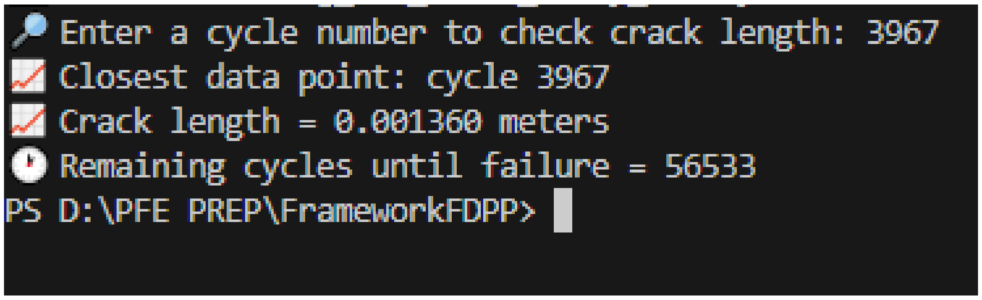
The alignment of both crack length stability and strain behavior in the initial loading cycles between the COMSOL simulations and the synthetic dataset supports the use of the synthetic data for extended fatigue analysis. These observations confirm that the synthetic data generation pipeline provides a credible approximation of physical behavior, especially in the early stages of crack initiation and propagation. Consequently, the synthetic data is validated for use in model training, especially when long-term fatigue simulation is not computationally feasible.

4.4 Querying Crack Length Using Processed Synthetic Data

In addition to preprocessing the training dataset, the `synthetic_study.py` script includes functionality that enables the user to query the estimated crack length at a specific cycle count. The script processes the `training_set2.csv` file and generates two key graphs: crack length vs. number of cycles and strain at each gauge vs. number of cycles figure 4.16 (the number 3967 cycles exist in the data set) . Furthermore, it prompts the user to input a desired cycle number, then returns:

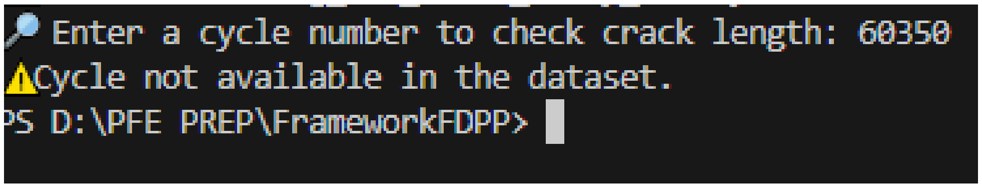
- The estimated crack length at that cycle.
- The number of cycles left until the critical crack length is reached (i.e., failure).

This interaction is helpful for quick inspection and understanding of structural health based on precomputed data. However, it has a limitation: if the user inputs a cycle value that is not explicitly listed in the dataset, the program returns an error ("Cycle not available in the dataset")as shown in figure 4.17 (the number 60350 cycles doesn't exist in the data set) . This is due to the fact that the dataset consists of a finite number of measured points.



```
🔎 Enter a cycle number to check crack length: 3967
📈 Closest data point: cycle 3967
📈 Crack length = 0.001360 meters
⏱ Remaining cycles until failure = 56533
PS D:\PFE PREP\FrameworkFDPP>
```

Figure 4.16: python terminal when the number of cycles is mentioned in the dataset



```
🔎 Enter a cycle number to check crack length: 60350
⚠️Cycle not available in the dataset.
PS D:\PFE PREP\FrameworkFDPP>
```

Figure 4.17: python terminal when the number of cycles is not mentioned in the dataset

Motivation for Predictive Modeling

This limitation motivates the need for a predictive model. A trained machine learning model can:

- Generalize to unseen cycle values.
- Predict the crack length and strain for any number of cycles—even beyond the ones present in the dataset.
- Improve decision-making and maintenance planning in real-time applications.

Hence, the next phase of this work involves developing and training a machine learning model capable of forecasting the remaining useful life (RUL) and crack length at arbitrary cycle counts using the synthetic dataset as a foundation.

4.5 Generating the Data sets :

4.5.1 Fracture Mechanics and Crack Modeling

The `crack.py` script implements analytical solutions from linear elastic fracture mechanics (LEFM) for a finite crack under Mode I (opening mode) loading, based on complex potentials. The formulations are primarily derived from [18]. The goal is to simulate stress, strain, and displacement fields around a crack tip, as well as crack growth over fatigue cycles.

- **Stress Functions:** The script defines the complex stress function $\phi(z)$, along with its first and second derivatives, $\phi'(z)$ and $\phi''(z)$, used to compute stresses and displacements around the crack tip.

- **Displacement Field:** Using Eq. (2.70) from [18], the displacement fields (u, v) are calculated for either plane stress or plane strain conditions. The complex variable $z = x + iy$ is used for evaluating the displacement in the material.
- **Stress Components:** The stress components $\sigma_{11}, \sigma_{22}, \tau_{12}$ are computed from $\phi'(z)$ and $\phi''(z)$ using Eq. (2.69) of the reference. In the plane strain case, σ_{33} is also estimated using the appropriate constitutive relations.
- **Strain Components:** The corresponding strain components are derived based on linear elasticity theory. Both plane stress and plane strain assumptions are supported, and ϵ_{33} is returned only under the plane stress scenario.
- **Crack Growth Law:** The function `length_paris_law` calculates the crack length a_k after k fatigue cycles using Paris' Law:

  $$a_k = \left(kC\left(1 - \frac{m}{2}\right)(\Delta\sigma\sqrt{\pi})^m + a_0^{1-m/2}\right)^{\frac{2}{2-m}} \tag{4.1}$$

  where C and m are material-specific parameters, $\Delta\sigma$ is the stress range, and a_0 is the initial crack length.

This script forms the theoretical backbone for generating synthetic strain and crack length data that mimic real-world fatigue crack propagation under repeated loading.

Synthetic Data Utility Functions: `utils.py`

The `utils.py` script provides essential tools for generating and processing synthetic crack growth datasets based on fracture mechanics principles. This utility file includes routines for simulating sensor readings, sampling material parameters, and formatting the dataset for machine learning applications.

Key Components:

- `build_dataset`: Converts raw simulation outputs into a structured Pandas DataFrame, organizing cycle indices, sensor (gauge) readings, and Remaining Useful Life (RUL) labels. This function also supports optional truncation at a critical crack length using a `t_star` value.

- `gen_param_sample`: Randomly samples initial crack length a_0, Paris law coefficient C, and exponent m using truncated and multivariate normal distributions. The sampling ensures realistic crack propagation behavior by enforcing physically meaningful constraints (e.g., $m > 0$).

- `gen_strain_value_gauge`: Computes synthetic strain values at arbitrary gauge locations and orientations (x, y, θ) using the analytical strain fields from fracture mechanics (via `crack.strain`).

- `gen_dataset`: Generates a collection of synthetic crack sequences. For each sample, it:

    1. Samples a_0, C, and m
    2. Computes strain evolution using Paris law and fracture equations
    3. Records sensor data and crack length at intervals determined by the `thinning` parameter

    The output is a list of dictionaries, each representing one sequence (used later for training/testing ML models).

The module relies on the analytical stress and strain functions implemented in `crack.py` and uses the JIT compiler from `Numba` to speed up computationally intensive routines.

4.5.2 Generation Procedure : `main.py`

The `main.py` script acts as the central driver for the synthetic dataset generation process used in crack length and Remaining Useful Life (RUL) prediction tasks. This script begins by parsing a series of user-defined parameters using the `argparse` library. These include material properties such as Young's modulus (`E`) and Poisson's ratio (`nu`), fracture parameters like the Paris' Law constants (`C`, `m`) and fracture toughness (`K_IC`), and the configuration of strain gauges (positions and angles).

Once the parameters are loaded, the script computes the critical crack length (`a_crit`) based on the given fracture toughness and maximum stress, using linear elastic fracture mechanics principles. It then creates a data directory for storing outputs and saves the current configuration for reproducibility.

The datasets are generated using the `utils.gen_dataset` function. Three separate sets are created: training, validation, and test. The test set includes an additional field `t_star` to represent the fraction of the full lifetime that is observable, mimicking real-world scenarios where complete crack growth history is not always available. The datasets are saved in both `.pkl` and `.csv` formats.

To structure the raw simulation output into a format suitable for machine learning workflows, the script uses the `utils.build_dataset` function. This transformation produces final CSV files that include the cycle number, strain readings from each gauge, and the corresponding RUL at each cycle. The training and validation datasets contain complete histories, while the test dataset is truncated based on `t_star`. Ultimately, the `main.py` script ensures scalable, reproducible generation of synthetic fatigue datasets suitable for data-driven prognostic modeling.

4.6 Cleaning the Data sets Procedure :

4.6.1 Data Transformation Pipeline

The original datasets generated using the `main.py` script—namely `training_set.csv`, `validation_set.csv`, and `test_set.csv`—contained structural information in a nested format. Specifically, each row in these files represented an entire experiment, with key fields such as the list of strain readings from multiple gauges (stored as an array in the `strains` column) and corresponding crack lengths over cycles (stored in `crack_lengths`). Although this format is compact, it is not directly suitable for supervised learning models, which typically require a tabular structure with one measurement per row.

4.6.2 Cleaning and Flattening the Dataset

To restructure these raw datasets into a usable machine learning format, a script named `clean_dataset.py` was developed (included in Appendix .1). This script flattens the nested data by iterating over each experiment and unrolling the strain and crack length sequences.

For each experiment:

1. The total number of measurements (`Nb_measures`) is retrieved.
2. The initial cycle index is computed from the total number of cycles using the known interval (500 cycles between measurements).
3. For each time step, a new row is created containing:
    - `sample_id`: an identifier for the original experiment,
    - `cycle`: the specific cycle number at which the measurement was taken,
    - `G1`, `G2`, `G3`: strain readings from the three gauges,
    - `crack_length`: the corresponding crack length at that cycle.

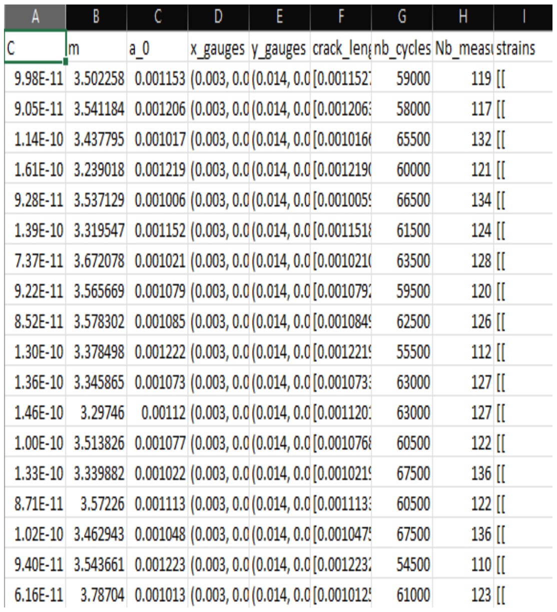
This transformation was applied to all three original datasets. The resulting cleaned datasets—`training_set_cleaned.csv`, `validation_set_cleaned.csv`, and `test_set_cleaned.csv`—are stored in the GitHub repository .1 associated with this project and were used for all model training and evaluation tasks.

Example of Transformation

Table 4.2 summarizes the difference between the raw and cleaned formats. As shown in Figure 4.18, the raw CSV contained nested data, which was flattened for machine learning use.

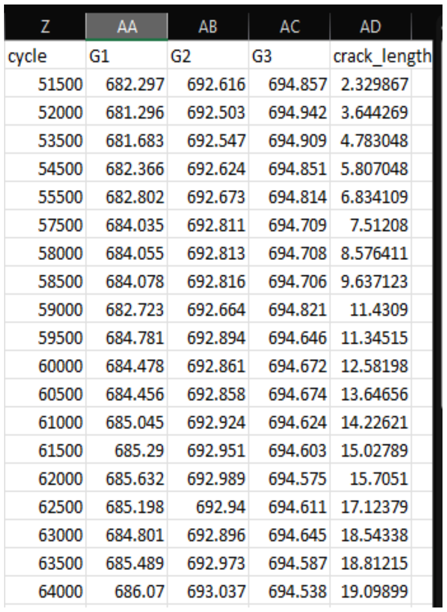
Table 4.2: Comparison of Raw vs Cleaned Dataset Format

Format	**Structure**	**Description**
Raw (`validation_set.csv`)	1 row per sample	Contains lists of strains and crack lengths
Cleaned (`validation_set_cleaned.csv`)	1 row per cycle	Each row includes G1–G3, cycle, crack length



A	B	C	D	E	F	G	H	I
C	m	a_0	x_gauges	y_gauges	crack_leng	nb_cycles	Nb_measu	strains
9.98E-11	3.502258	0.001153	(0.003, 0.0	(0.014, 0.0	[0.0011527	59000	119	[[
9.05E-11	3.541184	0.001206	(0.003, 0.0	(0.014, 0.0	[0.0012063	58000	117	[[
1.14E-10	3.437795	0.001017	(0.003, 0.0	(0.014, 0.0	[0.0010166	65500	132	[[
1.61E-10	3.239018	0.001219	(0.003, 0.0	(0.014, 0.0	[0.0012190	60000	121	[[
9.28E-11	3.537129	0.001006	(0.003, 0.0	(0.014, 0.0	[0.0010059	66500	134	[[
1.39E-10	3.319547	0.001152	(0.003, 0.0	(0.014, 0.0	[0.0011518	61500	124	[[
7.37E-11	3.672078	0.001021	(0.003, 0.0	(0.014, 0.0	[0.0010210	63500	128	[[
9.22E-11	3.565669	0.001079	(0.003, 0.0	(0.014, 0.0	[0.0010792	59500	120	[[
8.52E-11	3.578302	0.001085	(0.003, 0.0	(0.014, 0.0	[0.0010849	62500	126	[[
1.30E-10	3.378498	0.001222	(0.003, 0.0	(0.014, 0.0	[0.0012219	55500	112	[[
1.36E-10	3.345865	0.001073	(0.003, 0.0	(0.014, 0.0	[0.0010733	63000	127	[[
1.46E-10	3.29746	0.00112	(0.003, 0.0	(0.014, 0.0	[0.0011201	63000	127	[[
1.00E-10	3.513826	0.001077	(0.003, 0.0	(0.014, 0.0	[0.0010768	60500	122	[[
1.33E-10	3.339882	0.001022	(0.003, 0.0	(0.014, 0.0	[0.0010219	67500	136	[[
8.71E-11	3.57226	0.001113	(0.003, 0.0	(0.014, 0.0	[0.0011133	60500	122	[[
1.02E-10	3.462943	0.001048	(0.003, 0.0	(0.014, 0.0	[0.0010475	67500	136	[[
9.40E-11	3.543661	0.001223	(0.003, 0.0	(0.014, 0.0	[0.0012232	54500	110	[[
6.16E-11	3.78704	0.001013	(0.003, 0.0	(0.014, 0.0	[0.0010125	61000	123	[[

(a) Raw CSV file format: each row contains nested lists of G1–G3 strain readings and crack lengths.



Z	AA	AB	AC	AD
cycle	G1	G2	G3	crack_length
51500	682.297	692.616	694.857	2.329867
52000	681.296	692.503	694.942	3.644269
53500	681.683	692.547	694.909	4.783048
54500	682.366	692.624	694.851	5.807048
55500	682.802	692.673	694.814	6.834109
57500	684.035	692.811	694.709	7.51208
58000	684.055	692.813	694.708	8.576411
58500	684.078	692.816	694.706	9.637123
59000	682.723	692.664	694.821	11.4309
59500	684.781	692.894	694.646	11.34515
60000	684.478	692.861	694.672	12.58198
60500	684.456	692.858	694.674	13.64656
61000	685.045	692.924	694.624	14.22621
61500	685.29	692.951	694.603	15.02789
62000	685.632	692.989	694.575	15.7051
62500	685.198	692.94	694.611	17.12379
63000	684.801	692.896	694.645	18.54338
63500	685.489	692.973	694.587	18.81215
64000	686.07	693.037	694.538	19.09899

(b) Cleaned CSV file format: each row corresponds to a single cycle with explicit G1–G3 values and crack length.

Figure 4.18: Comparison between raw and cleaned CSV formats used for model training.

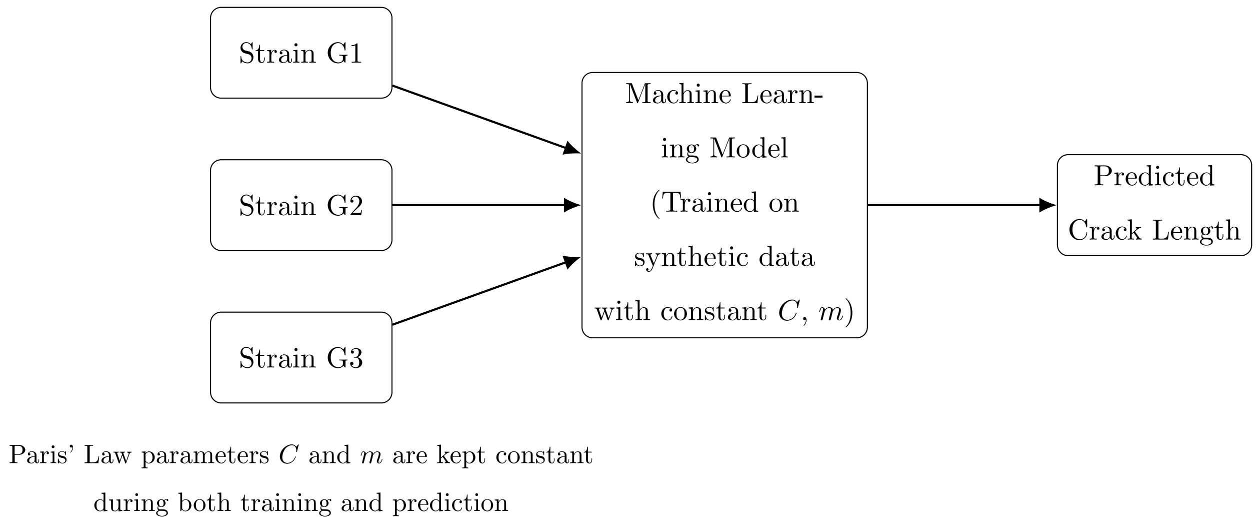
Availability

All original and cleaned datasets, as well as the associated scripts (`main.py`, `utils.py`, `crack.py`, `clean_dataset.py`), are available in the project's public GitHub repository .1. note that the crack length is in mm and the strain of the three gauges (G1 , G2 , G3) is in microstrain $\mu\varepsilon$.

4.6.3 Assumption of Constant Paris' Law Parameters

In this study, the Paris' Law parameters C and m were treated as constants throughout the data generation and model training process. This assumption is physically justified by the fact that both parameters are material-dependent and do not vary significantly under consistent loading and environmental conditions. Since the dataset was synthetically generated for a single material and under uniform stress range, it is valid to assume fixed values for C and m as shown in 4.19.

The goal of the machine learning models is not to estimate these parameters, but rather to learn a mapping from local strain measurements (G1, G2, G3) to the corresponding crack length at each cycle. By keeping C and m constant, the dataset encodes a consistent crack growth behavior, allowing the models to learn from strain patterns alone. This simplification reduces unnecessary complexity while still providing a reliable framework to evaluate the effectiveness of various data-driven approaches to fatigue prognosis.



Strain G1
Strain G2
Strain G3
Machine Learning Model (Trained on synthetic data with constant C, m)
Predicted Crack Length
Paris' Law parameters C and m are kept constant during both training and prediction

Figure 4.19: Schema illustrating the data-driven mapping from strain gauge readings (G1–G3) to crack length. The machine learning model learns this relationship based on training data where C and m are held constant.

4.7 Model Development and Training

In this work, three machine learning models were evaluated for predicting crack length in aircraft fuselage structures using strain gauge data: XGBoost, GRU, and Transformer.

4.7.1 XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful and efficient gradient boosting algorithm that has demonstrated strong performance on structured tabular data. It was trained to predict crack length using four features: number of cycles, and strain values from three sensors (G1, G2, G3). The final trained model was saved as `xgb_crack_model.pkl` and integrated into the GUI application.

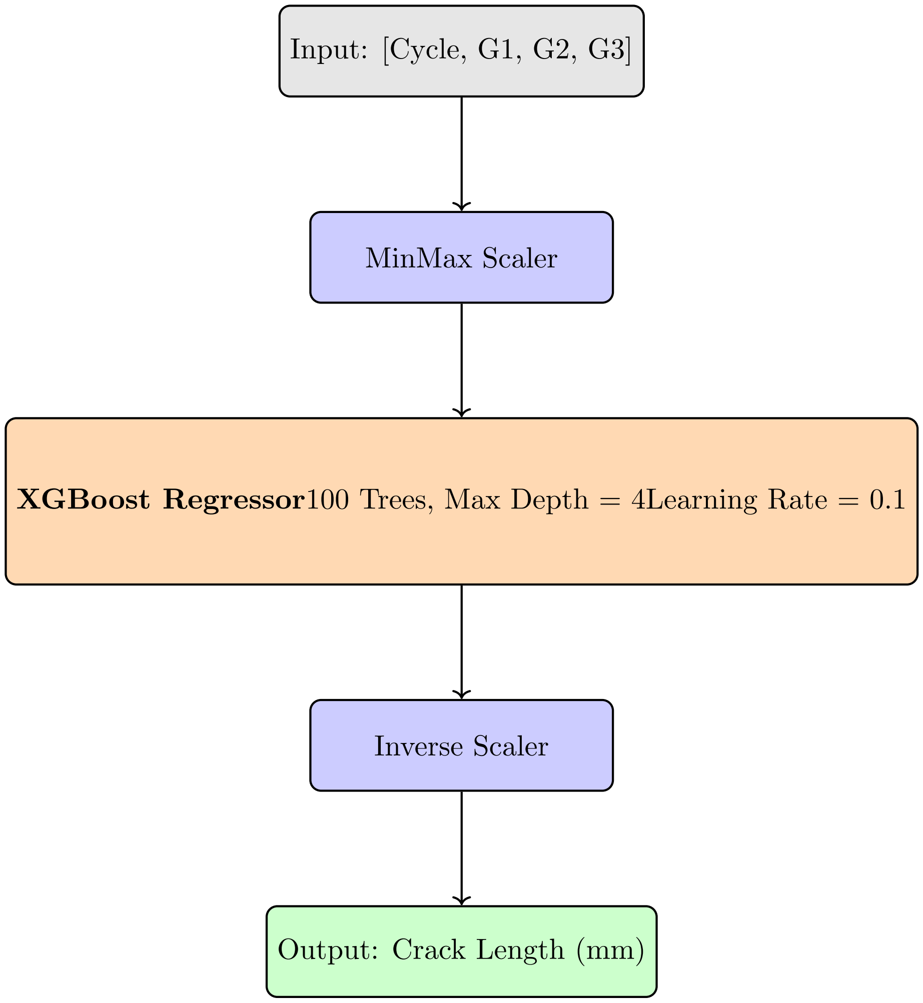

Input: [Cycle, G1, G2, G3]
MinMax Scaler
XGBoost Regressor100 Trees, Max Depth = 4Learning Rate = 0.1
Inverse Scaler
Output: Crack Length (mm)

Figure 4.20: Schematic of the XGBoost-based regression pipeline for crack length prediction.

Description

The XGBoost model was trained as a regression model to predict crack length based on the number of cycles and strain values (G1, G2, G3). Input features were normalized using a MinMax scaler. The model consisted of 100 decision trees, each with a maximum depth of 4, and used a learning rate of 0.1. Predictions were transformed back to the original scale using an inverse MinMax scaler.

Hyperparameters

Parameter	Value
Model Type	XGBoost Regressor
Number of Trees	100
Maximum Tree Depth	4
Learning Rate	0.1
Objective Function	`reg:squarederror`
Evaluation Metric	MAPE
Feature Scaler	MinMaxScaler
Input Features	Cycle, G1, G2, G3
Target Variable	Crack Length (mm)

Table 4.3: Hyperparameters used in the XGBoost model.

Justification

XGBoost was chosen due to its excellent performance on tabular data and its robustness to noise. The hyperparameters were selected through experimentation to balance accuracy and generalization.

4.7.2 GRU :Gated Recurrent Units

Gated Recurrent Units (GRUs) are a type of recurrent neural network designed to efficiently learn dependencies in sequential data using update and reset gates. In this study, GRUs were used to model the relationship between applied cycles, strain readings, and the resulting crack length.

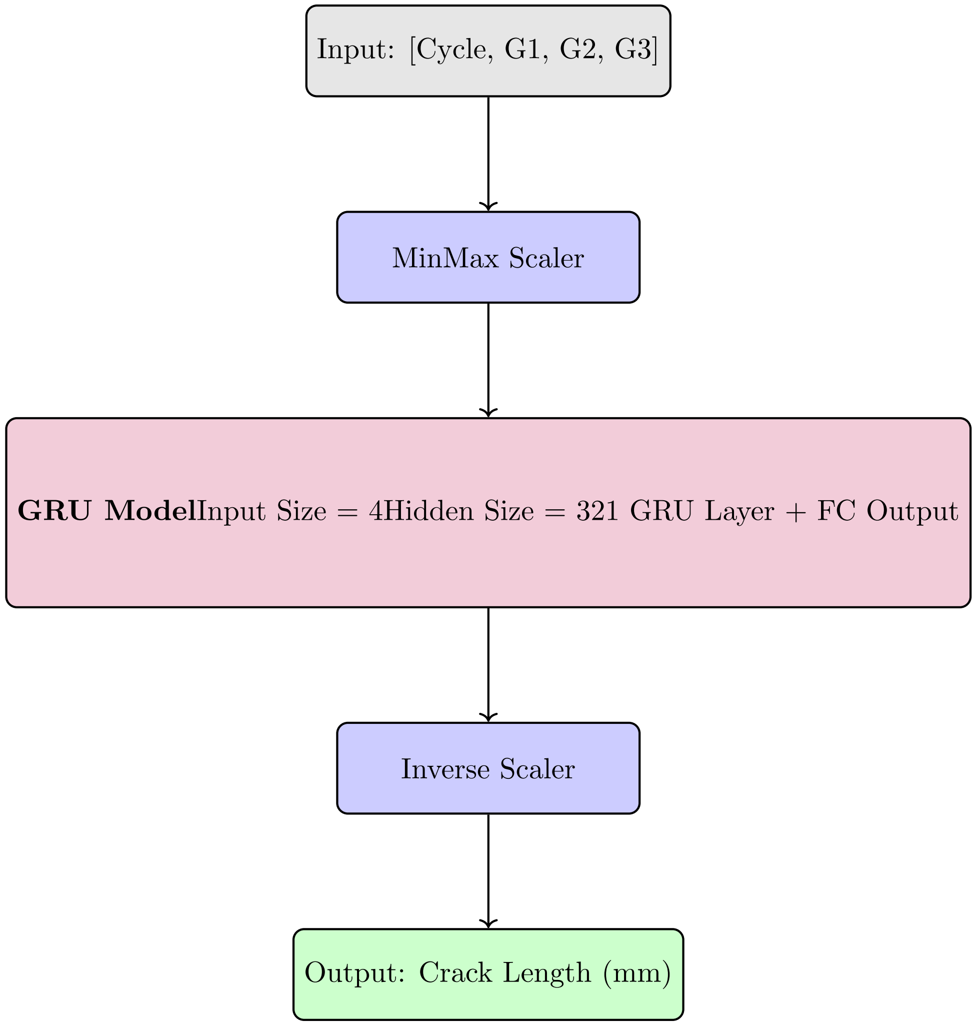

Input: [Cycle, G1, G2, G3]
MinMax Scaler
GRU ModelInput Size = 4Hidden Size = 321 GRU Layer + FC Output
Inverse Scaler
Output: Crack Length (mm)

Figure 4.21: GRU-based model architecture for crack length prediction.

Architecture

Description

The GRU model receives a vector of four features—Cycle, G1, G2, and G3—as input. These features are first scaled using a MinMax scaler. The sequence is then passed through a single-layer GRU network with a hidden dimension of 32. The output of the GRU is processed through a fully connected (FC) layer to produce the predicted (normalized) crack length, which is then inverse-scaled to obtain the crack length in millimeters.

Training Procedure

The model is trained using the Adam optimizer and Mean Squared Error (MSE) loss function. Training is conducted over 20 epochs with a batch size of 32. The training dataset is scaled prior to input, and predictions are inverse-scaled post-model to obtain results in the physical domain. A new model is trained from scratch for each specified training size to ensure fairness and consistency in model comparisons.

Hyperparameters

Parameter	**Value**
Model Type	GRU
Input Size	4
Hidden Size	32
GRU Layers	1
Output Layer	Fully Connected
Loss Function	Mean Squared Error (MSE)
Optimizer	Adam
Learning Rate	0.001
Epochs	20
Batch Size	32
Feature Scaler	MinMaxScaler
Input Features	Cycle, G1, G2, G3
Target Variable	Crack Length (mm)

Table 4.4: GRU model training configuration.

Justification

GRUs are well-suited for sequential regression tasks due to their ability to learn long- and short-term dependencies without incurring the full computational cost of LSTMs.

Their relative simplicity and efficiency make them a strong baseline for modeling temporal patterns in strain-based crack growth behavior.

4.7.3 Transformer

Transformers are attention-based deep learning models that have shown state-of-the-art performance in many sequential data tasks. Unlike recurrent models, Transformers process entire input sequences in parallel using self-attention mechanisms, making them highly effective for learning temporal dependencies.

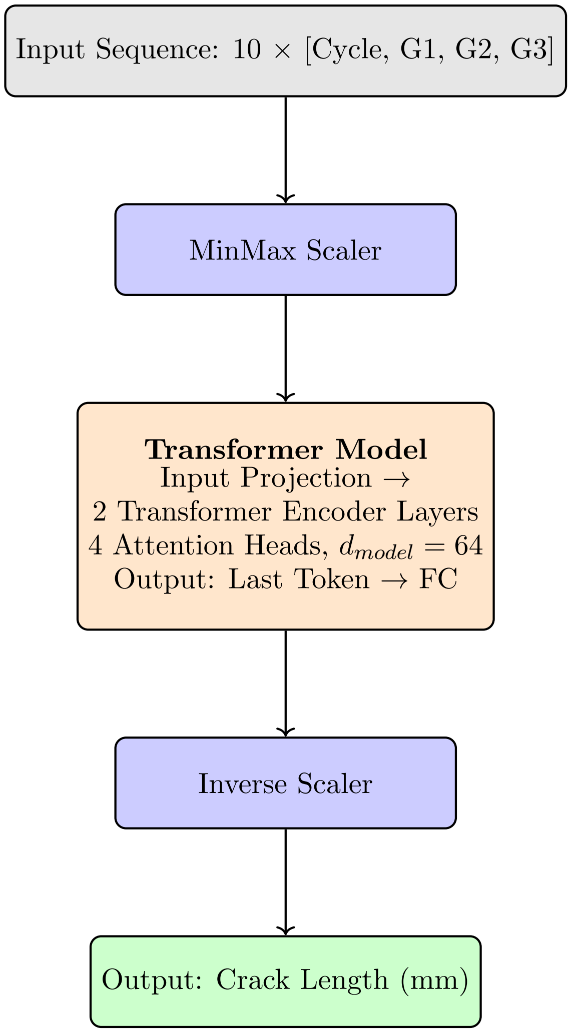

Input Sequence: 10 × [Cycle, G1, G2, G3]
MinMax Scaler
Transformer Model
Input Projection →
2 Transformer Encoder Layers
4 Attention Heads, $d_{model} = 64$
Output: Last Token → FC
Inverse Scaler
Output: Crack Length (mm)

Figure 4.22: Transformer-based model architecture for crack length prediction.

Architecture

Description

The Transformer model is designed to operate on sequences of strain data and associated cycle numbers. Each input sequence contains 10 time steps, where each step includes the features Cycle, G1, G2, and G3. After feature scaling, the input is projected to a higher dimensional space ($d_{model} = 64$) using a linear layer. This is followed by two stacked Transformer encoder layers with four attention heads each. The final prediction is based on the output corresponding to the last time step, passed through a fully connected layer. The output is then inverse-scaled to recover the predicted crack length in millimeters.

Training Procedure

For each training size, the input sequences and targets are generated using a sliding window of size 10. The model is trained using the Adam optimizer and Mean Squared Error (MSE) loss over 50 epochs with a batch size of 32. A new model is trained from scratch for each training size (100, 500, 1000) to ensure consistent and fair evaluation. Validation and test predictions are made using the same sequence-based windowing approach.

Hyperparameters

Justification

Transformers provide a powerful modeling approach for time series data due to their ability to capture long-range dependencies and parallelize computation. By leveraging attention mechanisms, the model can weigh the relative importance of each time step in a sequence, which is particularly advantageous in capturing subtle patterns in crack propagation dynamics.

4.8 Evaluation Metrics

To evaluate the performance of the models used for crack length prediction, the Mean Absolute Percentage Error (MAPE) is employed. MAPE is a common metric for regression

Parameter	Value
Model Type	Transformer
Sequence Length	10
Input Size	4
Embedding Dimension (d_{model})	64
Number of Layers	2
Number of Attention Heads	4
Dropout	0.1
Loss Function	Mean Squared Error (MSE)
Optimizer	Adam
Learning Rate	0.001
Epochs	50
Batch Size	32
Feature Scaler	MinMaxScaler
Input Features	Cycle, G1, G2, G3
Target Variable	Crack Length (mm)

Table 4.5: Transformer model training configuration.

tasks that expresses the average prediction error as a percentage of the true values:

$$\text{MAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

where A_t is the actual value and F_t is the predicted value. A lower MAPE indicates higher prediction accuracy.

4.8.1 Experimental Setup

Three models were evaluated: **XGBoost**, **GRU**, and **Transformer**. Each was trained using synthetic datasets of increasing size: 100, 500, and 1000 samples. Validation and test performance were recorded separately. All models take strain gauge measurements

(G1, G2, G3) as inputs and output predicted crack length.

4.8.2 MAPE Results

Table 4.6 presents the MAPE (%) on validation and test sets for all models and training sizes. The best values in each column are highlighted in **bold**.

Table 4.6: MAPE (%) Comparison Across Models and Training Sizes

Train Size	**XGBoost**		**GRU**		**Transformer**	
	Val MAPE	Test MAPE	Val MAPE	Test MAPE	Val MAPE	Test MAPE
100	0.20	0.28	3.37	3.80	5.15	5.42
500	**0.05**	**0.09**	0.68	0.57	4.76	5.40
1000	**0.04**	**0.04**	0.66	0.57	4.90	5.36

4.8.3 MAPE Visualization

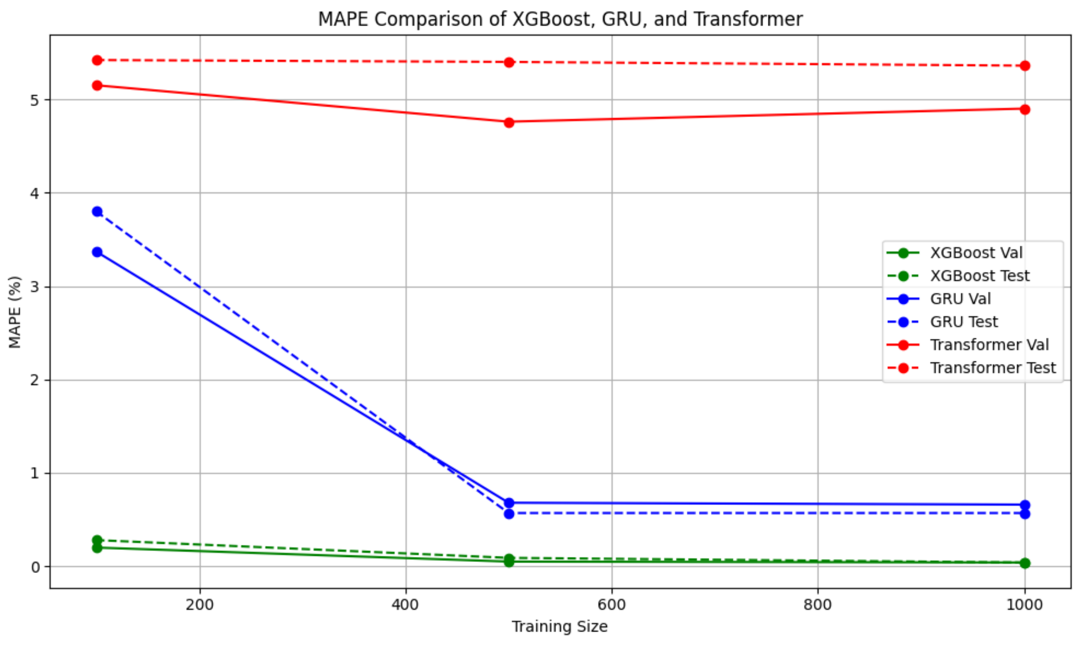


Figure 4.23: MAPE (%) of XGBoost, GRU, and Transformer across different training sizes

Figure 4.23 illustrates the performance trends for each model. XGBoost shows rapid improvement and maintains top performance, while GRU improves steadily with more data. Transformer performance remains relatively poor regardless of dataset size.

4.8.4 Discussion

From Table 4.6 and Figure 4.23, we observe:

- **XGBoost** consistently achieved the lowest MAPE scores across all training sizes. As a gradient boosting tree model, it is robust with small datasets and captures nonlinear interactions efficiently.
- **GRU**, a sequential deep learning model, performed reasonably well with medium and large datasets. It captures temporal dependencies but requires more data and training time to generalize well.
- **Transformer** underperformed relative to the other models. This is attributed to its complexity and the need for large-scale data to effectively learn self-attention patterns, which this application lacks.

4.8.5 Conclusion

Based on the MAPE values:

- The **best model** is **XGBoost**, due to its superior performance across all dataset sizes and its ability to generalize well with minimal data.
- The **worst model** is the **Transformer**, whose complexity is not justified by its performance in this regression task with relatively small datasets.

Chapter 5

Model Simplification and Practical Deployment

5.1 Data Interpolation Strategy for Crack Length Modeling

In the initial phase of the project, our raw experimental datasets posed a critical modeling challenge: the crack length remained nearly constant across a large number of initial cycles. Specifically, the raw training data began around cycle 44000, where changes in crack length were minimal and difficult to learn from. When attempting to use these early sections of the dataset, machine learning models such as XGBoost, GRU, or Transformer were unable to capture meaningful patterns due to the lack of variation. As a result, prediction behavior was inaccurate or unresponsive when users entered input values like cycle 1000 or 1100 into the deployed GUI model.

To resolve this, we implemented a data restructuring approach using linear interpolation. The goal was to simulate a progressive and realistic increase in crack length while preserving the original end cycle and final crack growth trends observed in the raw data. For each dataset—training, validation, and testing—we applied interpolation that adhered to the following principles:

- **Fixed sample size**: Each set retained its original number of elements (e.g., 1000 samples for training, 100 for validation and test).

- **Fixed cycle range endpoints**: All interpolated datasets ended at the same maximum cycle count (e.g., 72,000), consistent with the original measurements.

- **Staggered starting points**: The training set was interpolated from cycle 1000 to 72,000, validation from 1200 to 72,000, and test from 1300 to 72,000. This ensured that the model did not overfit to the same pattern and had a staggered learning context.

- **Interpolated crack growth**: The crack length was linearly interpolated between the starting and ending cycle points, based on the known minimum and maximum crack values. This created a smooth and realistic growth pattern, allowing the model to learn continuous degradation behavior.

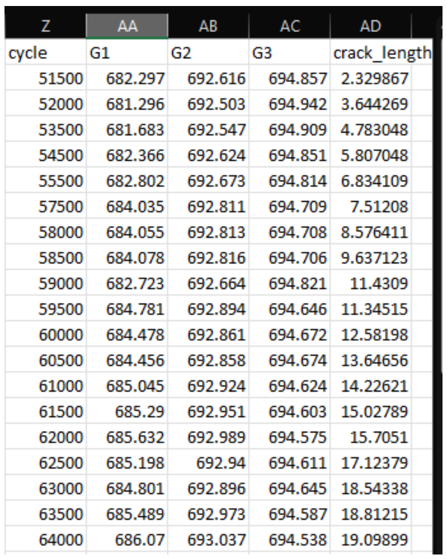
This interpolation method is justified for the following reasons:

1. **Preserving physical behavior**: The interpolated crack length values respect the monotonicity of real fatigue crack growth, which typically increases gradually before accelerating.

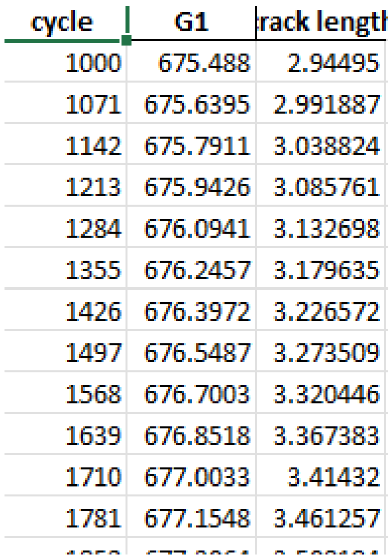
2. **Model responsiveness**: It enabled the model to generate realistic crack length predictions across a wider range of inputs, especially in the early-cycle domain where the raw data was flat.

3. **Consistent evaluation**: By maintaining the same sample count, comparisons of different training sizes (100, 500, 1000) on validation and test sets remained consistent and statistically meaningful.

This data augmentation technique significantly improved model generalization and was crucial in the successful deployment of the final GUI-based crack length and RUL prediction application.



Z	AA	AB	AC	AD
cycle	G1	G2	G3	crack_length
51500	682.297	692.616	694.857	2.329867
52000	681.296	692.503	694.942	3.644269
53500	681.683	692.547	694.909	4.783048
54500	682.366	692.624	694.851	5.807048
55500	682.802	692.673	694.814	6.834109
57500	684.035	692.811	694.709	7.51208
58000	684.055	692.813	694.708	8.576411
58500	684.078	692.816	694.706	9.637123
59000	682.723	692.664	694.821	11.4309
59500	684.781	692.894	694.646	11.34515
60000	684.478	692.861	694.672	12.58198
60500	684.456	692.858	694.674	13.64656
61000	685.045	692.924	694.624	14.22621
61500	685.29	692.951	694.603	15.02789
62000	685.632	692.989	694.575	15.7051
62500	685.198	692.94	694.611	17.12379
63000	684.801	692.896	694.645	18.54338
63500	685.489	692.973	694.587	18.81215
64000	686.07	693.037	694.538	19.09899



cycle	G1	rack length
1000	675.488	2.94495
1071	675.6395	2.991887
1142	675.7911	3.038824
1213	675.9426	3.085761
1284	676.0941	3.132698
1355	676.2457	3.179635
1426	676.3972	3.226572
1497	676.5487	3.273509
1568	676.7003	3.320446
1639	676.8518	3.367383
1710	677.0033	3.41432
1781	677.1548	3.461257

(a) cleaned syntatic data before interpolation (b) cleaned syntatic data after interpolation

Figure 5.1: Comparison between interpolated and non original data

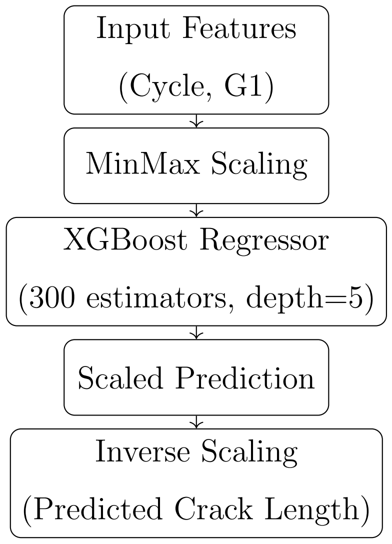
5.2 Model Deployment :

After developing several models for crack length prediction using features such as cycle count and strain gauge readings (G1, G2, G3), we integrated our work into a user-friendly GUI intended for maintenance technicians. However, during practical testing, we identified usability issues in the interface. Specifically, technicians were required to input multiple parameters (G1, G2, G3), which was not only cumbersome but also error-prone. Notably, G2 and G3 strain gauges exhibited minimal variation across samples and did not significantly impact prediction accuracy. Consequently, we removed them and focused on Cycle and G1 as inputs.

5.2.1 XGBoost Model Architecture for 2 features :

The final model chosen for deployment was an XGBoost regressor. The XGBoost architecture is an ensemble of decision trees trained sequentially, where each new tree attempts to correct the residual errors of the previous trees.

The general schema of the XGBoost pipeline is illustrated below:



5.2.2 MAPE Evaluation of the XGBoost Model for 2 features

The XGBoost model was trained using different subsets of the training data to evaluate its performance. The table below summarizes the Mean Absolute Percentage Error (MAPE) on both the validation and test sets:

Table 5.1: MAPE Results for XGBoost Model with Different Training Sizes

Training Size	**Validation MAPE (%)**	**Test MAPE (%)**
100	4.07	3.07
500	3.52	1.7
1000	0.05	0.17

in the table the MAPE table shows good values at 1000 samples with using two features G1 and number of cyles now we move to trying the trainig of the model with one feature which is number of cycles .

XGBoost with Only Cycle

To further reduce user input requirements and simplify the GUI, a second version of the model was trained using only the `Cycle` feature. The motivation was to see if reliable predictions could still be achieved from time alone, removing the need for strain measurements entirely.

Despite this reduction in input information, the model still demonstrated acceptable accuracy for deployment, especially when high interpretability and ease-of-use were prioritized.

- This version was particularly suited for systems where strain sensor data (`G1`) is unavailable or difficult to obtain.
- It enabled a fully automatic prediction based only on the number of cycles, simplifying deployment.

Table 5.2: MAPE (%) for XGBoost Model Trained with Only Cycle

Training Size	**Validation MAPE (%)**	**Test MAPE (%)**
100	4.07	3.7
500	1.71	0.21
1000	0.09	0.04

As shown, the accuracy remained high, especially with larger training sizes. This allowed the deployment of a user-friendly interface with minimal input fields while maintaining predictive performance.

5.2.3 Graphical User Interface (GUI) Development

After training the XGBoost models, two graphical user interfaces were developed to make crack length prediction accessible for technicians in the field. These tools allow the user to input relevant information and receive an instant crack length prediction along with a remaining useful life (RUL) estimate and a visual crack growth graph.

Motivation for Two GUI Versions

The initial version of the GUI required input of both the `G1` strain gauge value and the number of `cycles`. However, in real-world aircraft maintenance environments, it was found to be impractical for technicians to measure and input strain gauge values consistently. Moreover, features `G2` and `G3` were removed during preprocessing due to minimal variance and negligible impact on prediction performance.

This led to two stages of simplification:

1. First, the model was trained using only `Cycle` and `G1` as input features.
2. Then, a minimal version of the GUI was created using only `Cycle`, requiring no sensor data at all.

Full Input GUI (Cycle + G1)

The first interface includes two input fields:

- `Cycle` (number of loading cycles)
- `G1` (strain value from sensor 1)

The GUI predicts:

- Current crack length (in mm)
- Remaining crack growth before reaching critical length
- Estimated RUL in cycles

It also provides a graph showing projected crack growth until failure (when the crack reaches the critical length threshold for this case we chose crack length to be 10mm which is 1cm this can be modified).

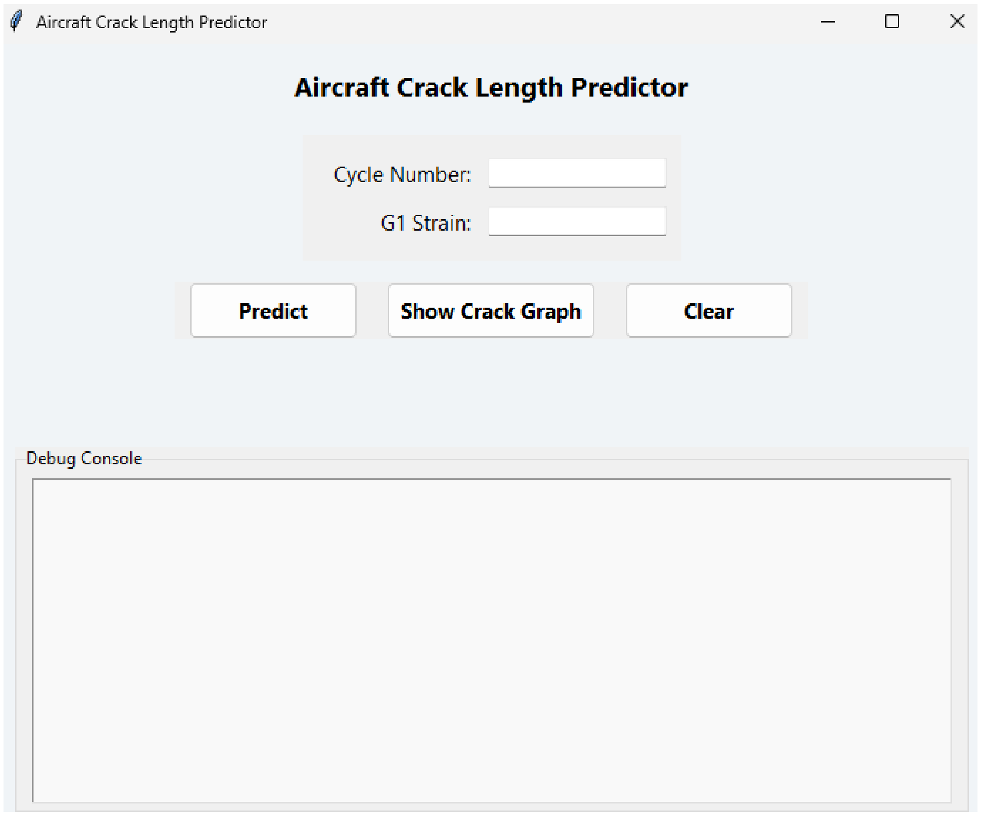


Figure 5.2: GUI Version Requiring Cycle and G1 Inputs

Minimal Input GUI (Cycle Only)

In the simplified version, only the number of `cycles` is required as input. This GUI version is suitable for environments where sensors like `G1` are unavailable or impractical to measure. Despite using only one input feature, the model achieved strong predictive performance as shown in the earlier MAPE tables.

The layout retains:

- Crack length prediction
- RUL estimate
- Crack growth visualization

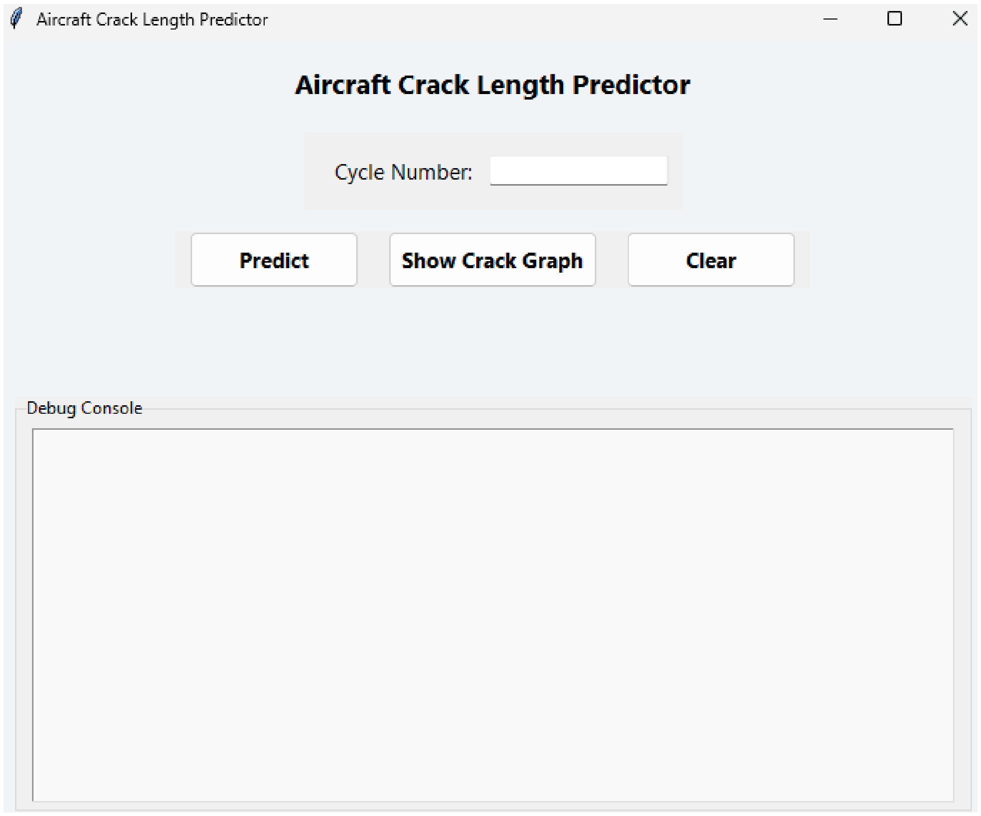

Aircraft Crack Length Predictor
Aircraft Crack Length Predictor
Cycle Number:
Predict
Show Crack Graph
Clear
Debug Console

Figure 5.3: GUI Version Requiring Only Cycle Input

Deployment Notes

Both GUIs were implemented in Python using the `Tkinter` library. The plotting feature was enabled via `Matplotlib`, and the models were loaded using `Joblib`.

Scripts for both versions are available in Appendix .1, along with screenshots of the interfaces and generated prediction plots.

5.2.4 Example Usage of GUI Interfaces

To illustrate the behavior and usability of the developed interfaces, we present one example for each GUI version.

Example 1: GUI with Cycle and G1 Input

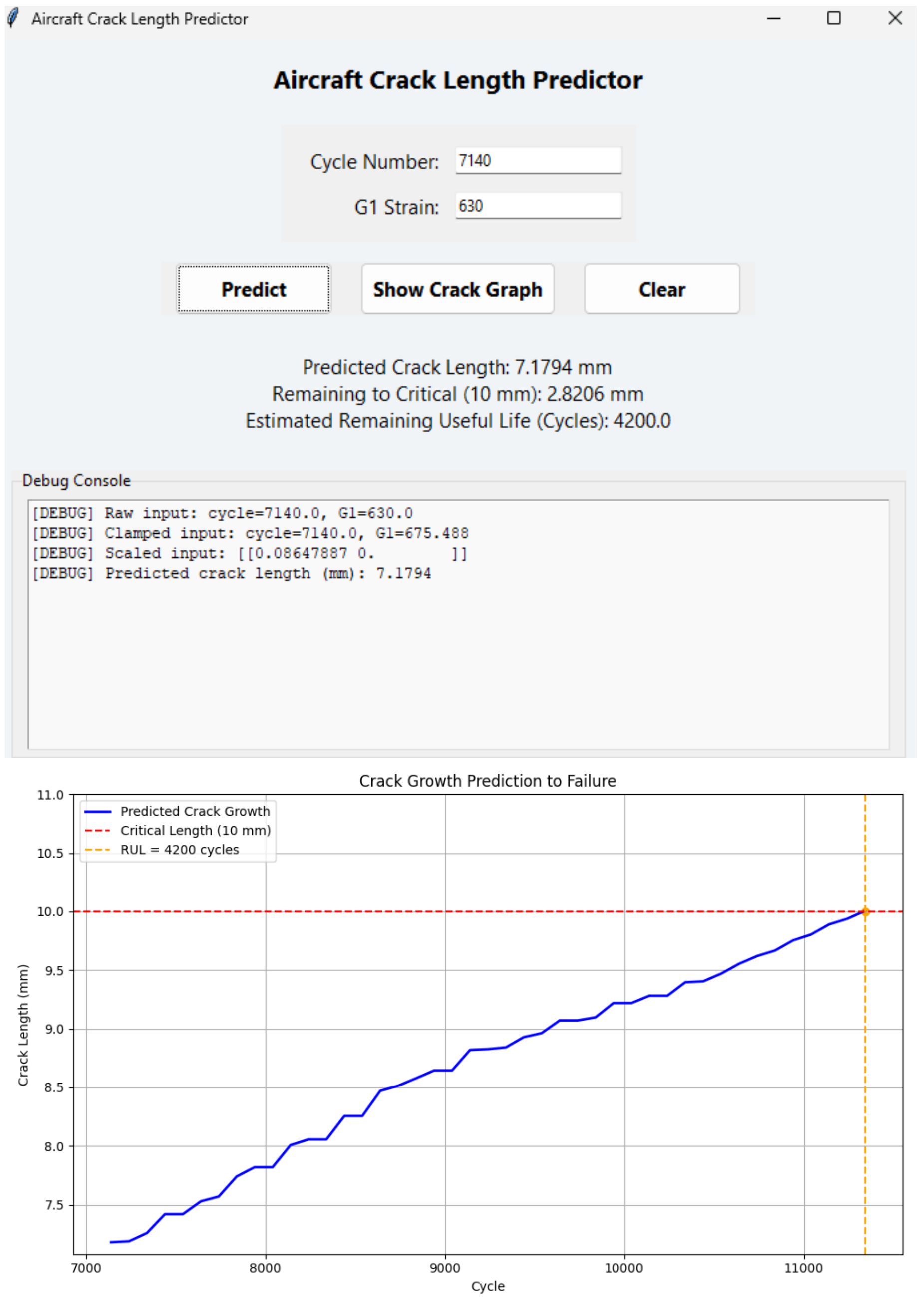
we set the critical length to $10mm$ and this can be changed . note that the crack length is in mm and the strain of the gauge (G1) is in microstrain $\mu\varepsilon$. In this example, the user provides:

- **Cycle** = 7140
- **G1** = 630 $\mu\varepsilon$.

The model predicts:

- **Predicted Crack Length:** 7.1794 mm
- **Remaining to Critical Length (10 mm):** 2.8206 mm
- **Estimated Remaining Useful Life:** 4200 cycles

A plot of the projected crack growth is automatically generated:



Aircraft Crack Length Predictor

Cycle Number: 7140

G1 Strain: 630

Predicted Crack Length: 7.1794 mm
Remaining to Critical (10 mm): 2.8206 mm
Estimated Remaining Useful Life (Cycles): 4200.0

```
[DEBUG] Raw input: cycle=7140.0, G1=630.0
[DEBUG] Clamped input: cycle=7140.0, G1=675.488
[DEBUG] Scaled input: [[0.08647887 0.        ]]
[DEBUG] Predicted crack length (mm): 7.1794
```

Figure 5.4: Crack Growth Prediction - GUI with Cycle and G1 Input

Example 2: GUI with Cycle Only Input

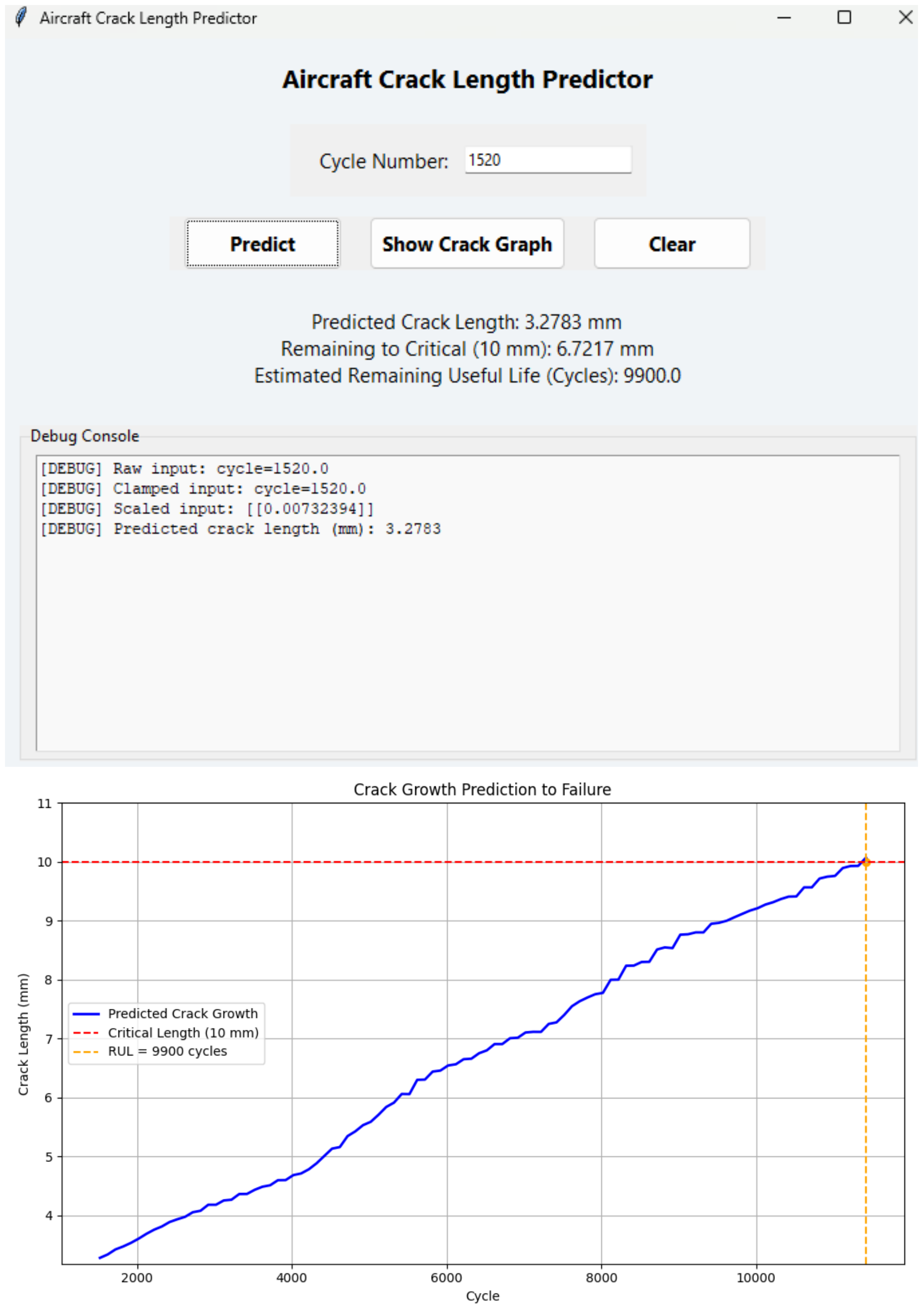
In the simplified version, the user provides only:

- **Cycle** = 1520

The output is:

- **Predicted Crack Length:** 3.2783 mm
- **Remaining to Critical Length (10 mm):** 6.7217 mm
- **Estimated Remaining Useful Life:** 9900 cycles

The following graph shows the predicted crack propagation up to failure:



Aircraft Crack Length Predictor

Cycle Number: 1520

Predicted Crack Length: 3.2783 mm
Remaining to Critical (10 mm): 6.7217 mm
Estimated Remaining Useful Life (Cycles): 9900.0

```
[DEBUG] Raw input: cycle=1520.0
[DEBUG] Clamped input: cycle=1520.0
[DEBUG] Scaled input: [[0.00732394]]
[DEBUG] Predicted crack length (mm): 3.2783
```

Figure 5.5: Crack Growth Prediction - GUI with Cycle Only Input

Chapter 6

Limitations and Future Work

6.1 Limitations

This project relied entirely on synthetic data to simulate crack propagation as a function of cycle count and strain gauge readings (G1, G2, G3). While synthetic data allows flexibility and controlled modeling conditions, it does not fully replicate the complexities and noise of real-world sensor data. As such, the results, including the model accuracy and GUI predictions, should be interpreted as a proof of concept rather than a field-validated solution.

Additionally, the G2 and G3 features, although initially included, showed minimal variation across the dataset and had little effect on model performance. Consequently, they were removed to simplify the model and the GUI interface. This design decision, while pragmatic, may overlook subtler mechanical behaviors that would appear in real-life measurements.

Another limitation is the lack of external validation or cross-hardware testing. The models have only been tested on datasets that were synthetically generated and cleaned. Therefore, their robustness under variable conditions, sensor noise, and irregular sampling intervals remains unverified.

6.1.1 Future Work

The most immediate improvement to this work would involve acquiring and integrating real-world data collected from physical strain gauge sensors (G1, G2, G3) mounted on aircraft components. Conducting a controlled laboratory experiment to track crack growth over time would provide a valuable benchmark for validating the model's predictions and assessing generalization capabilities.

With real sensor data:

- The preprocessing pipeline can be re-used with minimal adaptation.
- The trained models (GRU, Transformer, XGBoost, etc.) can be retrained using the same framework.
- The GUI application is already developed and ready to accept live or recorded inputs from technicians.

In addition, future work may explore:

- Incorporating more diverse sensor data (e.g., acoustic emission, vibration, or ultrasonic).
- Applying real-time streaming data with continual model updates.
- Enhancing the GUI to interface directly with embedded sensor systems for on-site deployment.
- Using physics-informed machine learning to better constrain predictions based on fracture mechanics principles.

In conclusion, this project serves as a foundational framework for crack length prediction in aircraft components. Once real experimental data becomes available, the system can be transitioned from a hypothetical prototype to a fully deployable diagnostic tool.

Chapter 7

Conclusion

In this project, we explored the development of a machine learning-based framework for predicting crack length growth in aircraft components, using strain data collected from simulated sensors. The central objective was to support maintenance decision-making by providing an intelligent system capable of estimating the current crack length and predicting the remaining useful life (RUL) of a structure before reaching a critical failure point.

We began by generating synthetic data, simulating strain gauge measurements (G1, G2, G3) and corresponding crack lengths over thousands of cycles. This allowed for experimentation in the absence of real sensor data. The data was interpolated, normalized, and split into training, validation, and test sets. A variety of machine learning and deep learning models were evaluated, including GRU, LSTM, Transformer, Temporal Convolutional Networks (TCN), and XGBoost. Among these, XGBoost was selected for deployment due to its balance between performance, interpretability, and speed.

After extensive experimentation, we discovered that some features, such as G2 and G3, did not significantly contribute to predictive accuracy. To simplify the system and improve usability, we progressively reduced the feature space from four features (Cycle, G1, G2, G3) to two (Cycle and G1), and finally to a single feature (Cycle). This simplification had minimal impact on accuracy, with the XGBoost model still achieving extremely low Mean Absolute Percentage Error (MAPE) values on both validation and test datasets.

To complete the pipeline, we developed two Graphical User Interface (GUI) applications: one that accepts both Cycle and G1 as input, and another that only requires Cycle. These interfaces display the predicted crack length, the remaining distance to the critical threshold, and an estimated RUL. They also feature graphical plots of predicted crack growth over time, making them intuitive and accessible for use by technicians.

While the models demonstrated high accuracy, it is important to note that the results are based entirely on synthetic data. Therefore, the findings should be interpreted as a theoretical proof-of-concept. With real-world strain gauge data, the models can be re-trained and validated to ensure robust performance in practical applications. Nonetheless, the core architecture, model evaluation framework, and deployment-ready GUI have all been successfully implemented and validated.

In conclusion, this project offers a complete end-to-end solution for fatigue crack prediction in aircraft structures. From synthetic data generation and preprocessing, to model training, evaluation, and deployment, the system is designed to be both practical and extensible. With access to real experimental data, the groundwork laid in this thesis can be transformed into a powerful diagnostic tool to enhance safety and efficiency in aerospace maintenance operations.

Bibliography

[1] J. Schijve, *Fatigue of Structures and Materials.* Dordrecht: Springer, 2009.

[2] R. K. Mobley, *An Introduction to Predictive Maintenance.* Butterworth-Heinemann, 2nd ed., 2002.

[3] J. Z. Sikorska, M. Hodkiewicz, and L. Ma, "Prognostic modelling options for remaining useful life estimation by industry," *Mechanical Systems and Signal Processing*, vol. 25, no. 5, pp. 1803–1836, 2011.

[4] Y.-L. Lee, J. Pan, R. B. Hathaway, and M. E. Barkey, *Metal Fatigue Analysis Handbook: Practical Problem-solving Techniques for Computer-aided Engineering.* Elsevier, 2005.

[5] COMSOL Inc., *COMSOL Multiphysics User's Guide*, 2024. https://www.comsol.com.

[6] R. Liu, X. Liang, X. Yu, and S. Ghosh, "A framework for generating large data sets for fatigue damage prognostic problems," *Structural Health Monitoring*, 2023. https://github.com/FDPP/FrameworkFDPP.

[7] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] K. Cho and et al., "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[10] S. Kiranyaz, T. Ince, and M. Gabbouj, "1d convolutional neural networks and applications: A survey," *Mechanical Systems and Signal Processing*, vol. 151, p. 107398, 2021.

[11] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[12] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," pp. 785–794, 2016.

[13] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[14] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[15] A. Akrim, C. Gogu, T. Guillebot de Nerville, P. Strähle, B. Waffa Pagou, M. Salaün, and R. Vingerhoeds, "A framework for generating large data sets for fatigue damage prognostic problems," in *2022 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pp. 25–33, IEEE, 2022.

[16] C. Farhat *et al.*, "A framework for generating large data sets for fatigue damage prognostic problems," *HAL preprint*, 2023.

[17] S. Kim and B. D. Youn, "A framework for generating large data sets for fatigue damage prognostic problems," *Mechanical Systems and Signal Processing*, vol. 165, p. 108383, 2022.

[18] A. T. Zehnder, *Fracture Mechanics.* Springer New York, Springer, 2012.

Appendix

.1 Appendix A: Simulations and syntatic data

The Python scripts and datasets developed for this thesis are available on GitHub:

`https://github.com/menialoubna/fatigue-prognostics-thesis`

and the GitHub Repository for the framework :

`https://github.com/ansak95/FrameworkFDPP`

.1.1 Script 1: COMSOL Data Processing (100 Cycles)

`comsol100.py` Processes strain gauge data from the 100-cycle COMSOL simulation. It extracts strain, estimates crack growth, and visualizes results.

.1.2 Script 2: COMSOL Data Processing (1000 Cycles)

`comsol1000.py` Handles extended simulation data over 1000 cycles. Used to compare simulation results with synthetic data.

.1.3 Script 3: Synthetic Fatigue Data Generator

`main.py` Runs the fatigue data generation pipeline using the FrameworkFDPP approach.

.1.4 Script 4: Crack Growth via Paris' Law

`crack.py` Contains Paris' Law implementation for synthetic crack growth modeling.

.1.5 Script 5: Data Utilities

`utils.py` Provides helper functions for data reshaping, cleaning, and exporting.

.1.6 Script 6: Training Dataset Cleaning

`syntatic_study.py`

Cleans and reshapes the training dataset to include columns: cycles, strains (G1–G3), crack length, and cycles left.

.1.7 Script 7: Data Cleaning Script

`clean_data.py`

Located in the GitHub repository. This script transforms the nested format in `training_set.csv`, `validation_set.csv`, and `test_set.csv` into flat, row-based formats suitable for machine learning. The cleaned outputs include:

`training_set_cleaned.csv`, `validation_set_cleaned.csv`, and `test_set_cleaned.csv`.

.2 Appendix B : Model training and GUI APP

- `Thexgboostmodel.py` – training the data with XGboost.
- `GRU-Model.py` – training the data with GRU.
- `Transformers-model.py` – training the data with transformers .

.2.1 Script 9: Data augmentation and interpolation

- `data-augment.py`
- `data-intterpo.py`

.2.2 Script 10: Interface and GUI

- `mini-app-G1-cycle.py` : interface where there's two inputs G1 and Cycle

- `mini-app-cycle.py` : interface where there's one input Cycle

.3 Appendix C : Data Sets

- `validation_set.csv`, `test_set.csv`, `training_set.csv`: Original nested synthetic datasets.
- `validation_set_cleaned.csv`, `test_set_cleaned.csv`, `training_set_cleaned.csv`: Flattened datasets generated by the `clean_data.py` script for model training.
- `strain_comsol100.csv`: COMSOL 100-cycle output
- `strain_comsol_1000.csv`: COMSOL 1000-cycle output
- `training_set2.csv`: Raw synthetic training set from FDPP
- `training_set_clean_ready_realcycles.csv`: Processed training data with strain and crack length
- `train3.xlsx` : used in augmentation
- `test3.xlsx` : used in augmentation
- `val3.xlsx`: used in augmentation
- `aug-train3.xlsx` : used in data interpolation
- `aug-test3.xlsx` : used in data interpolation
- `aug-val3.xlsx` : used in data interpolation
- `train3-adjusted-final.xlsx` : data after interpolation
- `test3-adjusted-final.xlsx` : data after interpolation
- `val3-adjusted-final.xlsx` : data after interpolation