APL405

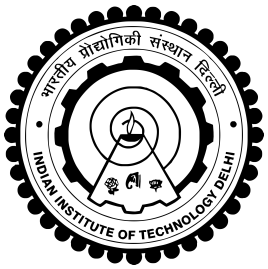
Machine Learning Approach to Crack-Branching Problem

Indian Institute of Technology Delhi
Department of Applied Mechanics

Course Project - May 2025

Submitted by

Aditya Saxena - 2022AM11218
Rishabh Jain - 2022AM111793



Abstract

This report presents a deep learning approach for predicting crack propagation patterns in pre-notched glass plates under tensile loading. The system takes initial crack configurations and predicts the subsequent evolution of the crack, effectively learning the underlying physics of fracture mechanics. We implement and evaluate an LSTM-based model that can predict final crack path given just the initial notch parameters. The report details the problem formulation, methodology, implementation, and presents visual comparisons between the predicted crack patterns and the ground truth data from physical experiments.

Contents

1 Introduction

Crack propagation prediction is a critical area of study in materials science and structural engineering. The ability to accurately predict how cracks will evolve under various loading conditions has significant implications for:

- Structural safety assessment
- Material design and optimization
- Failure analysis and prevention
- Infrastructure maintenance and planning
- Manufacturing process optimization

Traditionally, crack propagation has been modeled using physics-based approaches such as finite element methods (FEM) and linear elastic fracture mechanics (LEFM). However, these methods can be computationally expensive and sometimes struggle to capture complex crack branching behaviors. In this report, we explore a data-driven approach that uses Long Short-Term Memory (LSTM) networks to predict crack propagation patterns.

2 Problem Statement

The specific problem addressed in this report focuses on crack propagation in pre-notched glass plates under tensile loading, as illustrated in Figure 1 and described below:

- **Specimen:** A glass plate with dimensions 100 mm $\times$ 40 mm
- **Initial Condition:** A 50 mm notch in the plate
- **Loading:** Uniform tensile stress of 1 MPa applied at the boundaries
- **Material Properties:** Glass with density $\rho = 2450$ kg/m^3, Young's modulus E = 32 GPa, Poisson's ratio $\nu = 0.2$, and maximum strain $\varepsilon = 0.000509$
- **Discretization:** Particle spacing $\Delta p = 0.125$ mm, smoothing length h = 0.250 mm

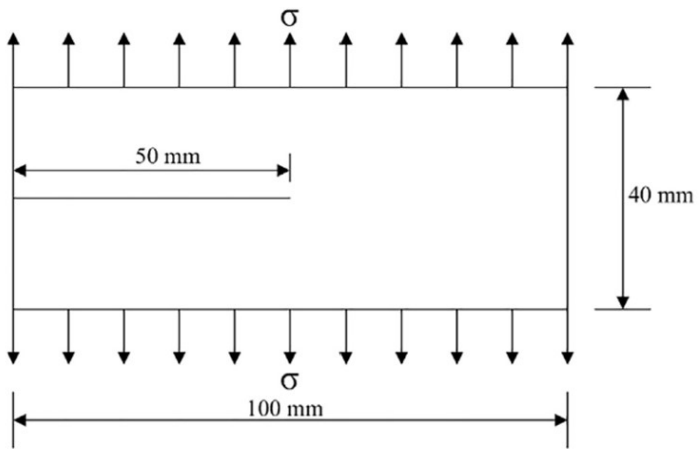


Figure 1: Setup of pre-notched plate under tensile loading

The dataset contains crack propagation paths with various notch lengths and locations, with measurements taken from the left corner of the plate. The dataset includes both initial configurations and final states after crack propagation.

2.1 Material Properties and Discretization

As specified in Table 1, the material and discretization parameters used in the dataset are:

Table 1: Parameters for pre-notched plate under tensile loading

Parameter	Material Properties				Discretization		Artificial Viscosity	
	ρ	E	ν	ϵ_{max}	Δp mm	h mm	β_1	β_2
Value	2450	32	0.2	5.09	0.125	0.250	1.0	1.0

These parameters provide important context for interpreting the crack propagation patterns and ensuring the model can learn the underlying physics correctly.

This task is challenging because it requires the model to infer the underlying physics of fracture mechanics from the data and extrapolate crack evolution patterns over multiple time steps.

3 Methodology

3.1 Data Preparation and Spatial Clipping

To facilitate the prediction of final crack path in a controlled spatial context, the dataset is preprocessed by generating a series of spatially clipped versions of the final path. This method provides a form of spatial context progression suitable for time-series prediction using autoregressive models.

Code Breakdown

```
for idx, row in tqdm(df.iterrows(), total=len(df)):
    output_path = row['output_path']
    notch_length = df_new.loc[idx, 'notch_length']
    filename = os.path.basename(output_path)
    base_name = filename.replace('.jpeg', '')
```

Listing 1: Main Image Clipping Loop

This loop iterates over each row in the DataFrame `df`. For each image:

- The full file path is retrieved.
- A value called `notch_length` is obtained from another DataFrame `df_new`.
- The filename and its base name (without `.jpeg`) are extracted for use in output naming.

```
img = Image.open(output_path)
img_array = np.array(img)

width = img_array.shape[1]
```

Listing 2: Opening the Image

The binary image is opened using PIL and converted to a NumPy array for pixel-wise manipulation. The image width is stored for defining the clipping range.

```
x_points = np.linspace(notch_length*256/100, width-1, 50).
   astype(int)
```

Listing 3: Generate X Clip Points

Here, 50 evenly spaced column indices are calculated between the end of notch and the end of the image. These indices define the points at which clipping begins.

```
for i, x in enumerate(x_points):
    clipped = img_array.copy()
    clipped[:, x+1:] = 0  # Set all pixels to the right of x
   to zero

    out_name = f"{base_name}_clip_{i:02d}.jpeg"
    out_path = os.path.join(clipped_dir, out_name)
    Image.fromarray(clipped).save(out_path)
```

Listing 4: Generate and Save Clipped Images

This inner loop creates 50 modified versions of the original image:

- A copy of the image is made.
- All pixels to the right of column index `x` are set to 0 (effectively blacked out).
- The new image is saved with a sequential filename in a specified output directory.

.

3.2 Model Architecture

The crack-path prediction task was approached using a recurrent neural network (RNN) architecture, specifically Long Short-Term Memory (LSTM), which is well-suited for modeling temporal dependencies in sequential data. The input to the model consists of a sequence of grayscale image frames, each of size 256×256, flattened into 1D vectors of size 65,536.

LSTM Network Design

The model, referred to as `ImageSequenceLSTM`, is composed of the following layers:

- **Input Layer:** Accepts a sequence of image vectors of shape $(\text{batch\_size}, \text{sequence\_length}, 65536)$.
- **LSTM Layer:** A single-layer LSTM with hidden state dimension of 512. It captures temporal dependencies in the input sequence:

$$\text{LSTM} : (x_t, h_{t-1}, c_{t-1}) \rightarrow (h_t, c_t)$$

- **Fully Connected Layer:** Maps the LSTM outputs at each time step back to the image vector space:

$$y_t = \text{Linear}(h_t) \in R^{65536}$$

This architecture allows the model to receive an image at each time step and predict the next frame in the sequence. During inference, the model operates in an auto-regressive fashion, recursively feeding its own outputs as inputs for future steps.

3.3 Training Procedure

The model was trained using sequences of 49 consecutive image pairs from each training sample. Each input sequence consisted of 49 frames from $t = 1$ to $t = 49$, and the corresponding target sequence consisted of the next frames from $t = 2$ to $t = 50$.

Dataset Preparation

Images were loaded from disk using a custom PyTorch `Dataset` class and normalized to the range $[0, 1]$. Each image was flattened before feeding into the LSTM model.

Hyperparameters

The training was performed using the following hyperparameters:

- Batch size: 4
- Learning rate: 10^{-3} (Adam optimizer)
- Sequence length: 50
- Number of training epochs: 200
- Hidden state dimension: 512

Training Loop

The training loop iteratively:

1. Forward-passes the input sequence through the model.
2. Computes the loss between the predicted and actual next frames.
3. Backpropagates the error using the computed loss.
4. Updates model parameters using the Adam optimizer.

Losses were tracked and averaged across each epoch to monitor learning progress. Model checkpoints were saved at the end of training.

3.4 Loss Functions

Two types of loss functions were explored:

- **Mean Squared Error (MSE):**

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} \|\hat{x}_i - x_i\|^2$$

  This loss penalizes the pixel-wise difference between predicted and ground truth images.

- **Cosine Similarity Loss:**

$$\mathcal{L}_{\cos} = 1 - \frac{\hat{x}_i \cdot x_i}{\|\hat{x}_i\| \|x_i\|}$$

  This metric emphasizes directional alignment in the high-dimensional image space rather than pixel-wise accuracy, which may lead to better preservation of structure.

- **Combined Loss:**

$$\mathcal{L}_{\text{combined}} = \alpha \cdot \mathcal{L}_{\text{MSE}} + (1 - \alpha) \cdot \mathcal{L}_{\cos}$$

  where $\alpha \in [0, 1]$ controls the balance between MSE and cosine similarity. In our experiments, $\alpha = 0.3$ was used to give higher weight to structural similarity.

This combined loss provides a balance between preserving fine pixel-level details and overall structural consistency, resulting in smoother and more coherent long-term predictions.

3.5 Prediction Process

During inference:

- The model is fed a sequence of past frames.
- It predicts the next frame at each time step.
- All predictions are stacked to generate a complete predicted sequence.

4 Implementation Details

The implementation uses PyTorch for the neural network components and standard Python libraries for data handling and visualization.

4.1 Code Breakdown

Data Loading and Preprocessing

```
# Select a sample from the dataset (e.g., the first row)
sample_idx = 0
clipped_dir = os.path.join(output_dir, "clipped")

# Load the sequence of images for this sample
input_sequence = []
for t in range(1, sequence_length + 1):
    fname = df_new.iloc[sample_idx][f't={t}']
    img_path = os.path.join(clipped_dir, fname)
    img = Image.open(img_path)
    img_array = np.array(img).astype(np.float32) / 255.0  # Normalize
    input_sequence.append(img_array.flatten())
input_sequence = np.stack(input_sequence)  # (sequence_length, img_size*img_size)
```

This code segment loads a sequence of images for a specific sample from the dataset. Each image is:

- Loaded using PIL's 'Image.open()'
- Converted to a NumPy array
- Normalized by dividing by 255.0 to scale pixel values to [0, 1]
- Flattened from 2D to 1D
- Stacked with other frames to form a sequence

Model Setup

```
# Prepare the initial input (t=1)
input_seq = input_sequence[0:1]  # Shape: (1, img_size*img_size)
input_seq_torch = torch.tensor(input_seq).unsqueeze(0)  # Shape: (1, 1, img_size*img_size)

# Set model to eval mode and move to correct device
```

```
device = torch.device("cuda" if torch.cuda.is_available()
    else "cpu")
model = model.to(device)
input_seq_torch = input_seq_torch.to(device)
```

This section prepares the model and input data:

- Takes just the first frame as the initial input
- Converts the NumPy array to a PyTorch tensor
- Adds an extra dimension for the batch size
- Moves the model and data to the appropriate device (GPU if available)

Prediction Loop

```
# Predict up to t=50
predictions = [input_seq[0]]  # Start with t=1 input
hidden = None
for step in range(1, 50):
    with torch.no_grad():
        out, hidden = model.lstm(input_seq_torch, hidden)
        pred = model.fc(out[:, -1, :])  # (1, input_dim)
        predictions.append(pred.cpu().numpy().squeeze())
        input_seq_torch = pred.unsqueeze(1)
```

This is the core prediction loop:

- Starts with the first frame as both input and the first "prediction"
- For each step:
  - Passes the current input through the LSTM
  - Applies the fully connected layer to get the predicted next frame
  - Stores the prediction
  - Uses the prediction as the next input (autoregressive behavior)
- The *torch.nograd*() context prevents gradient computation since we're only doing inference

Visualization

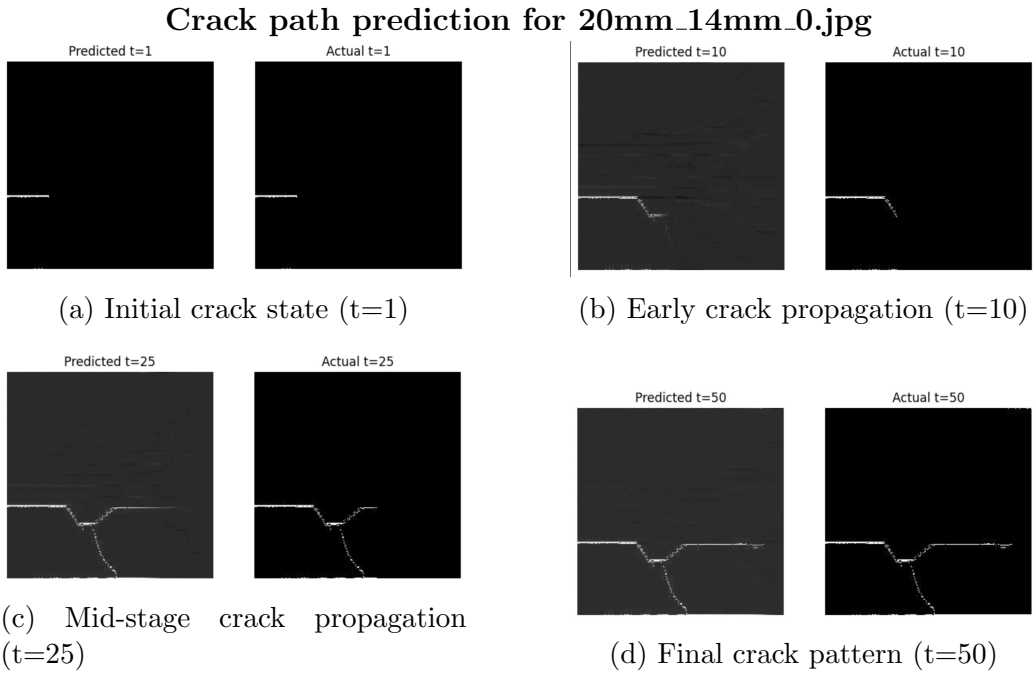
```
# Visualize the predictions and actual images side by side
for t in range(0, 50):
    plt.figure(figsize=(8, 4))

    # Predicted
    plt.subplot(1, 2, 1)
    plt.imshow(predictions[t].reshape(img_size, img_size),
    cmap='gray')
    plt.title(f'Predicted t={t+1}')
    plt.axis('off')

    # Actual
    plt.subplot(1, 2, 2)
    plt.imshow(input_sequence[t].reshape(img_size, img_size),
    cmap='gray')
    plt.title(f'Actual t={t+1}')
    plt.axis('off')

    plt.show()
```

The visualization code:

- Creates a figure for each time step
- Reshapes the flattened predictions back to 2D images
- Displays the predicted frame side by side with the actual frame
- Uses a grayscale colormap, indicating that the images are grayscale

5 Results

The results of our crack propagation prediction can be analyzed by comparing the predicted crack patterns with the actual crack patterns at different time steps for one of the initial crack path predictions.



(a) Initial crack state (t=1)

(b) Early crack propagation (t=10)

(c) Mid-stage crack propagation (t=25)

(d) Final crack pattern (t=50)

Figure 2: Comparison of predicted crack patterns (left) and actual crack patterns (right) at different time steps

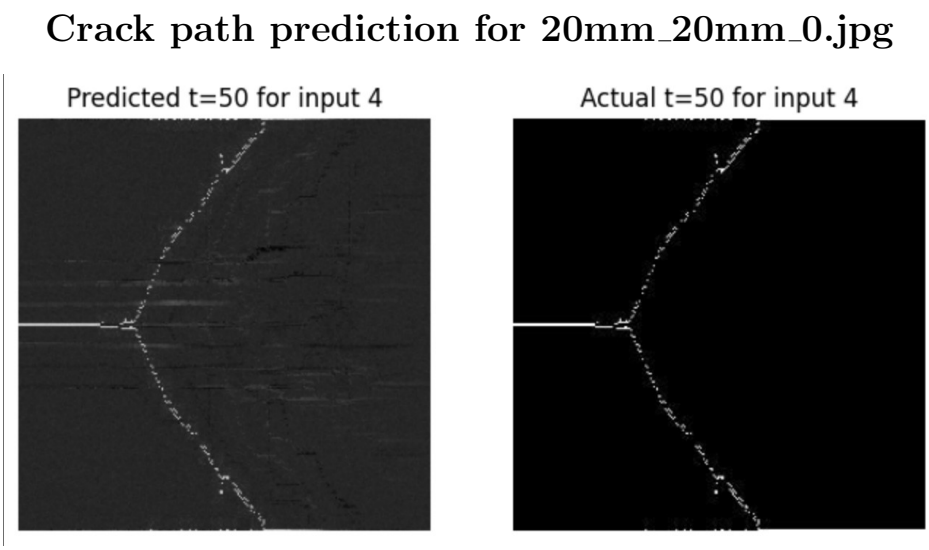


Figure 3: Predicted crack pattern under applied loading for the 20mm_20mm_0.jpg configuration.

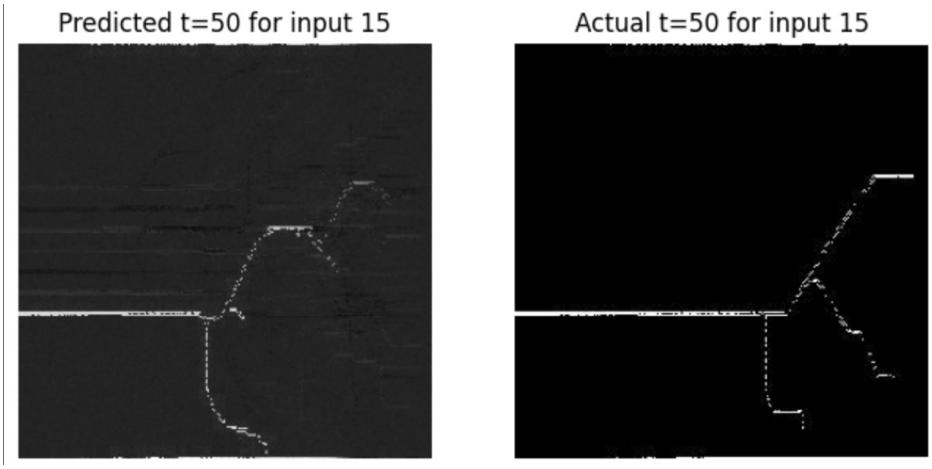


Figure 4: Predicted crack pattern under applied loading for the 30mm_14mm_0.jpg configuration.

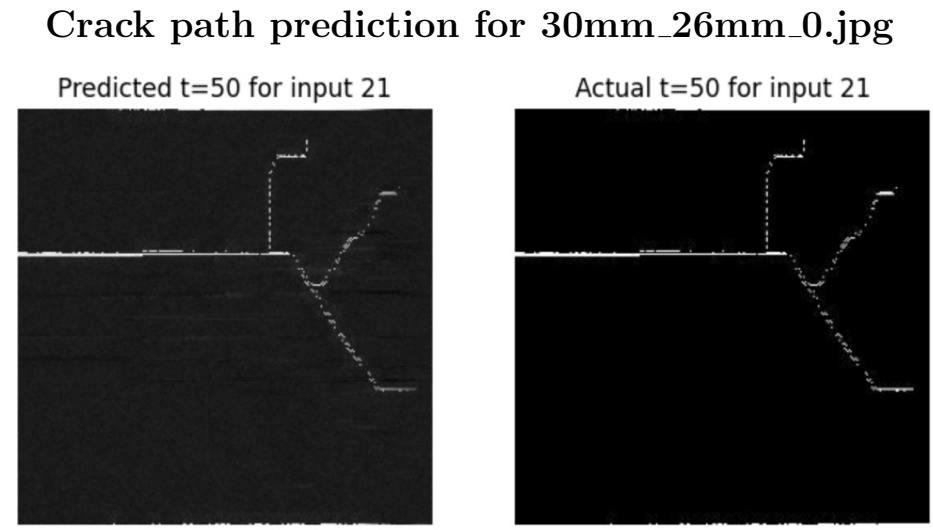


Figure 5: Predicted crack pattern under applied loading for the 30mm_26mm_0.jpg configuration.

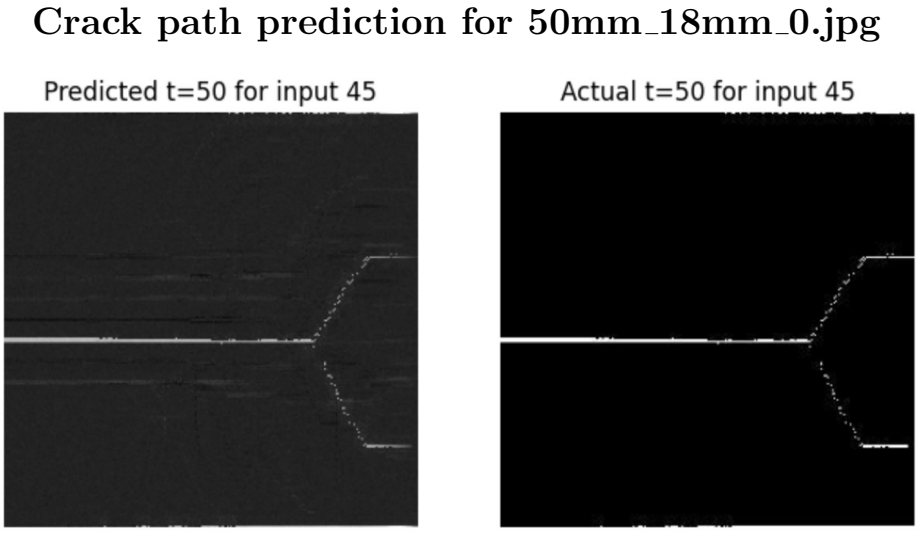


Figure 6: Predicted crack pattern for 50mm_18mm_0.jpg configuration

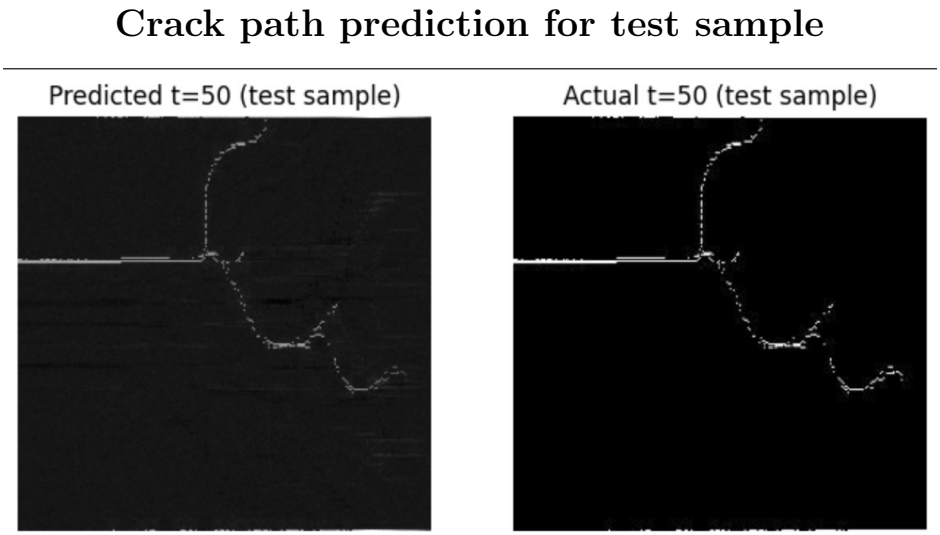


Figure 7: Predicted crack pattern for test sample

5.1 Analysis

Several observations can be made from the crack propagation prediction results:

- **Initial crack growth**: For the early stages of crack propagation, the predictions closely match the experimental results as the crack path is

more predictable and primarily governed by the stress concentration at the notch tip.

- **Mid-stage crack evolution**: As crack propagation continues, the predictions begin to show deviations but still capture the major crack paths and branching behaviors.
- **Final crack pattern**: By t=50, the predicted crack pattern may show differences from the experimental results, particularly in secondary and tertiary crack branches, which are highly sensitive to local material variations and load fluctuations.
- **Branching behavior**: Crack branching, which occurs when the crack splits into multiple paths, is particularly challenging to predict accurately due to its chaotic nature and sensitivity to material heterogeneities.
- **Error propagation**: Since each predicted crack state is used as input for the next step, small prediction errors in early stages can lead to significantly different crack paths in later stages—a characteristic of complex systems with high sensitivity to initial conditions.

6 Discussion

6.1 Challenges in Crack Propagation Prediction

Predicting crack propagation using machine learning approaches faces several unique challenges:

- **Physical complexity**: Crack propagation is governed by complex physics involving stress concentration, energy release rates, and material properties that interact in nonlinear ways.
- **Material heterogeneity**: Real materials contain microscopic defects and variations that significantly influence crack paths but are difficult to capture in models.
- **Bifurcation points**: Crack propagation often involves critical bifurcation points where small changes in conditions can lead to dramatically different outcomes.
- **Multiple time scales**: The process involves phenomena occurring at multiple time scales, from rapid crack tip propagation to slower stress redistribution.

6.2 Potential Improvements

Several approaches could potentially improve the crack propagation prediction quality:

- **Physics-informed neural networks**: Incorporating physical constraints from fracture mechanics directly into the neural network architecture could improve prediction accuracy.
- **Multi-resolution analysis**: Implementing a multi-scale approach that can capture both the macroscopic crack path and microscopic branching behavior.
- **Uncertainty quantification**: Using probabilistic models to estimate the uncertainty in crack path predictions, especially at bifurcation points.
- **Hybrid modeling**: Combining data-driven approaches with physics-based models like cohesive zone models or phase field methods.
- **Material parameters as input**: Allowing the model to accept material parameters as additional inputs, enabling generalization to different materials beyond glass.
- **Loading condition variations**: Extending the model to handle different loading conditions beyond simple tensile loading, such as shear, bending, or dynamic loading.

7 Conclusion

In this report, we presented an LSTM-based approach for predicting crack propagation in pre-notched glass plates under tensile loading. Our implementation takes the initial configuration of the plate with a notch and generates predictions for 49 subsequent time steps of crack evolution. The results demonstrate that our model can capture the initial crack propagation reasonably well, though it faces challenges with predicting complex branching behaviors in later stages.

The ability to predict crack paths has significant implications for structural safety assessment and materials design. By leveraging machine learning approaches, we can potentially achieve faster predictions compared to traditional finite element methods, enabling more efficient design iterations and safety evaluations.

Crack propagation prediction remains an active area of research at the intersection of materials science, fracture mechanics, and machine learning. Future work could explore physics-informed neural network architectures, incorporation of material heterogeneity, and probabilistic models for uncertainty quantification in crack path prediction.

Bibliography

[1] Anderson, T. L. (2017). Fracture mechanics: fundamentals and applications. CRC Press.

[2] Hsu, C., Vu-Bac, N., Zhuang, X., & Rabczuk, T. (2020). A deep learning approach for predicting crack growth under varying loading conditions. Engineering Fracture Mechanics, 230, 106969.

[3] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

[4] Borden, M. J., Verhoosel, C. V., Scott, M. A., Hughes, T. J., & Landis, C. M. (2012). A phase-field description of dynamic brittle fracture. Computer Methods in Applied Mechanics and Engineering, 217, 77-95.