Phase-Field Fracture Simulation Dataset: Hyperelastic Multi-Crack Response Under Loading and Unloading

Maryam Hakimzadeh, Lori Graham-Brady, Somdatta Goswami
Department of Civil and Systems Engineering, Johns Hopkins University

This dataset has been generated as a part of the Department of Energy funded project (DE-SC0024162, Physics and Uncertainty Informed Latent Operator Learning).

Phase-field models have emerged as a powerful approach for simulating realistic fracture problems. These models treat fracture as a coupled problem between a vector-valued displacement field and a scalar- or vector-valued phase field that indicates the material damage state. Traditional methods for modeling crack discontinuities face significant numerical challenges due to the singular nature of cracks. Phase-field approaches address this by regularizing the crack using gradient terms to smear out the crack faces, enabling the use of standard numerical methods like finite elements for simulations.

1. Problem Definition

This dataset has been inspired by an example presented in our recent work [1], where we investigate cracks in a cavity under cyclic loading. The material used in the simulations follows a large-deformation neo-Hookean model in 2D, with the model formulation and details comprehensively described in [1].

For the material parameters, we adopt values that approximately correspond to PMMA, a commonly used material in experimental fracture research [2]. After non-dimensionalizing, the material properties are set as follows:

$$\mu = 1, \quad G_c = 1.721 \times 10^{-5}.$$

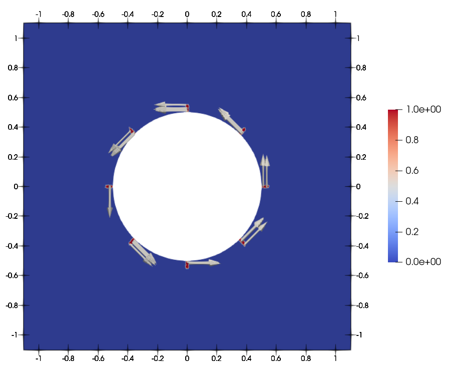
Additionally, we set $\lambda = 0$ to eliminate the effects of Poisson's ratio. For the phase-field regularization parameter introduced in (1.1), we use a non-dimensionalized length scale of $\epsilon = 0.015$. Furthermore, a value of $\eta_\epsilon = 1 \times 10^{-3}$ is employed for the numerical calculations. Overall, the proposed energy formulation is:

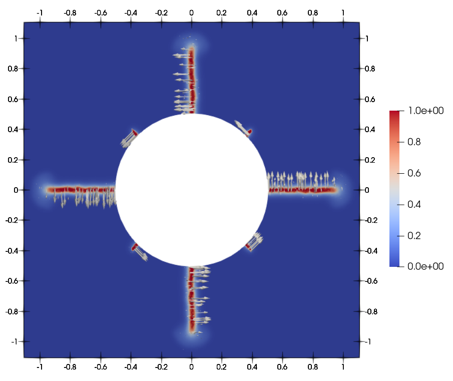
$$E[\boldsymbol{y}, \boldsymbol{d}] = \int_\Omega \left(\left((1 - |\boldsymbol{d}|)^2 + \eta_\epsilon \right) W(\nabla \boldsymbol{y}) + \left(1 - (1 - |\boldsymbol{d}|)^2 \right) \mathcal{W}(\nabla \boldsymbol{y}, \boldsymbol{n}) \right) \mathrm{d}V_{\boldsymbol{x}} + G_c \int_\Omega \left(\frac{|\boldsymbol{d}|^2}{2\epsilon} + \frac{\epsilon}{2} |\nabla \boldsymbol{d}|^2 \right) \mathrm{d}V_{\boldsymbol{x}}, \tag{1.1}$$

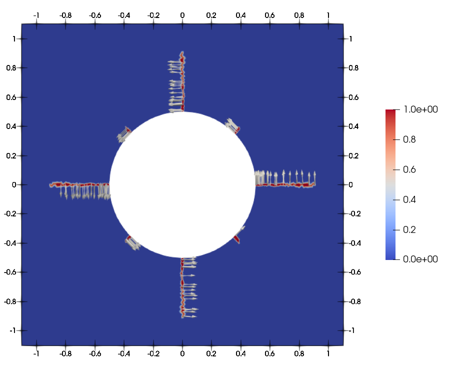
subject to the constraint $|\mathbf{d}| \leq 1$. Additional details, including the formulations of $W(\nabla \mathbf{y})$ and $\mathcal{W}(\nabla \mathbf{y}, \mathbf{n})$ for hyperelastic materials, are provided in [1].

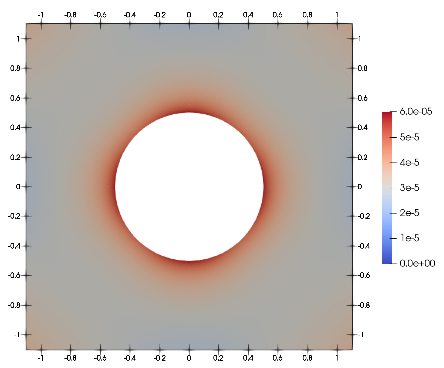
In this problem, we investigate the growth of fracture patterns within a cavity subjected to remote loading. While our work is inspired by recent meticulously controlled experiments that exhibit a variety of intriguing behaviors [3–6], these experiments incorporate additional physical phenomena, such as inertia and cavitation, which are not addressed in the present work.

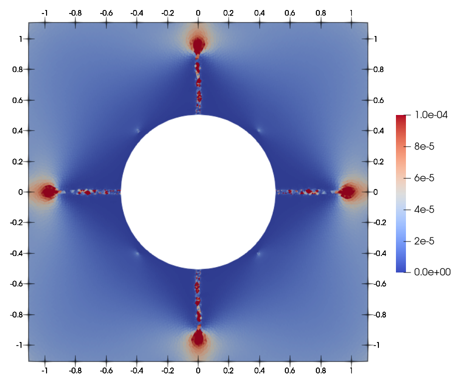
The configuration under study involves a square specimen with a circular cavity, as depicted in Figure 1(a). The initial setup includes eight short pre-cracks. Remote loading is applied by enforcing affine boundary conditions, specifically $\mathbf{u} = \mathbf{F^0 x}$ on the outer boundary, where $\boldsymbol{u}$ is displacement vector and $\mathbf{F^0}$ is a scalar multiple of the identity matrix. Additionally, the surface of the cavity is assumed to be traction-free. Figure 1(b) illustrates the initial elastic energy density of the specimen under compression, serving as a baseline. This represents the elastic energy density in an intact, uncracked specimen. Figures 1(c) and 1(d) depict the crack configuration and the corresponding elastic energy density, respectively, when the specimen is subsequently subjected to tensile loading. The results indicate that four out of the eight pre-cracks propagate, while the others remain dormant. The elastic energy distribution reveals the anticipated stress concentrations at the crack tips. It is noteworthy that the loading process was applied in small increments to achieve the crack configuration shown in the figure. This approach was necessary, as the cracks tend to grow rapidly and reach the boundary shortly after the state depicted in the figures. Figures 1(e) and 1(f) depict the crack configuration and the corresponding elastic energy density, respectively, when the specimen is subsequently subjected to compressive loading. The crack configuration remains largely unchanged, although the cracks appear visually narrower. This effect arises from the implemented healing approach, which allows healing when the magnitude of $\mathbf{d}$ falls below a critical threshold. A significant observation is that the elastic energy density in Figure 1(f) is identical to the uncracked configuration shown in Figure 1(b), despite the markedly different crack configurations in these scenarios. This result highlights that the elastic response of the cracked configuration under compression matches that of the uncracked configuration, as all cracks have fully closed. This outcome underscores the model's efficacy in accurately capturing the desired mechanical response.


(a) Initial unloaded configuration with pre-cracks.


(b) Crack growth under tensile load.


(c) Crack configuration under a second cycle of compressive load; crack patterns do not evolve under compression.


(d) Elastic energy density of the initial configuration under compression.


(e) Elastic energy density of the cracked configuration under tensile load.

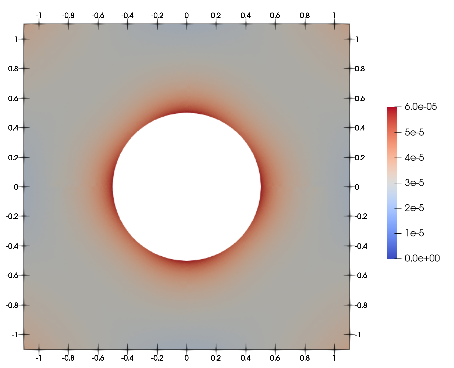

(f) Elastic energy density of the cracked configuration under compressive load; response is identical to the intact initial configuration.

Figure 1. Crack growth in a circular cavity under cyclic remote loading. [1]

2. Dataset Generation

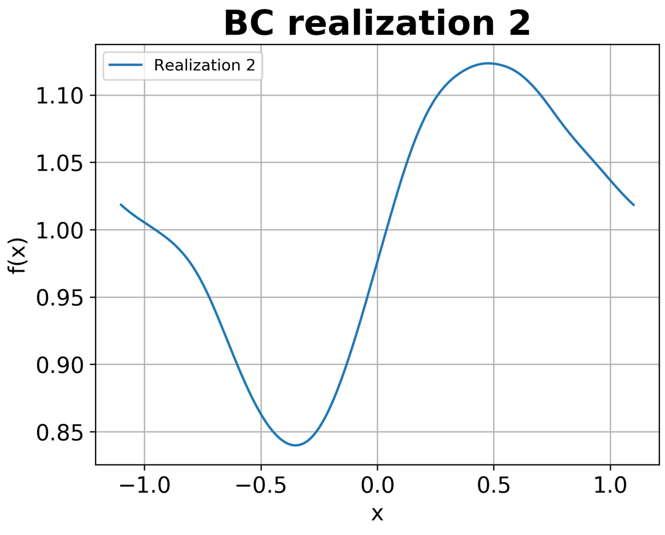
Tension loading was performed in the dataset generation process, followed by compression. For brevity, we omit the initial compression cycle described in detail in [1]. To generate the dataset for use with neural operators, we applied 1000 distinct boundary conditions on the outer boundary of the domain and computed crack growth under these varying conditions. In the original problem presented in [1], a constant load was applied to the outer boundary using $\mathbf{y} = \mathbf{F}^0\mathbf{x}$, where $\mathbf{F}^0$ is a constant scalar multiple of the identity tensor. However, for dataset generation, Gaussian random fields (GRFs) were employed to create varying functions, which were then used as multiplicative factors for $\mathbf{F}^0$. Section 2.A further explains the applied boundary condition.

2.A. Boundary Condition

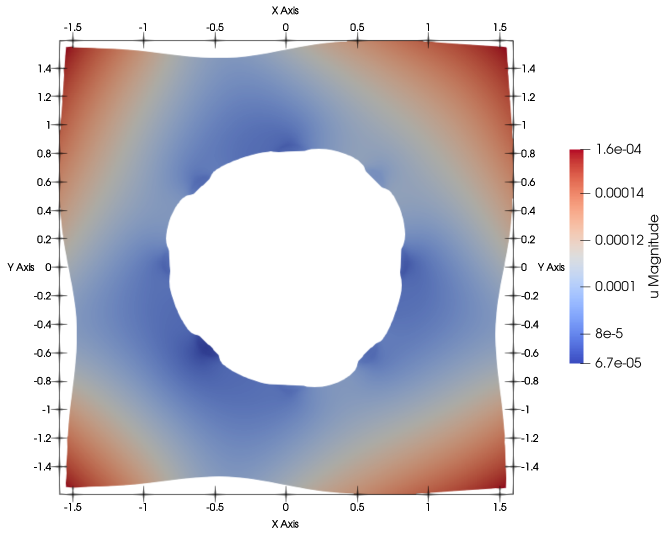
This subsection further discusses the boundary condition that has been used to generate this dataset. First, let's discuss the affine boundary condition, where we have $\mathbf{u} = \mathbf{F^0}\mathbf{x}$, where $\mathbf{u}$ is the displacement vector, $\mathbf{F^0}$ is the affine tensor, and $\mathbf{x}$ is the location vector. We can write $\mathbf{u} = \mathbf{F^0}\mathbf{x}$ as:

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \tag{2.1}$$

where u_1 and u_2 show the displacement along the x_1 axis and x_2 axis, respectively. In [1], $\mathbf{F^0}$ was taken to be the identity tensor multiplied by a constant scalar α. However, for the dataset generation, we take α to be a scalar function of space, where $\alpha = f(x_1)$ on the top and bottom faces of the boundary, and $\alpha = f(x_2)$ on the left and right faces of the boundary. Functions $f(x_1)$ and $f(x_2)$ are equal to the function $f(x)$ that is created using GRF for each realization, and then projected onto the x_1 and x_2 axes to obtain $f(x_1)$ and $f(x_2)$. The plot below shows $f(x)$ for one of the realizations and the deformed shape after applied displacement. It is worth noting that the reference configuration comprises a square domain with coordinates spanning from $(-1.1, -1.1)$ to $(1.1, 1.1)$. Under these boundary conditions, the entire domain experiences tensile deformation, albeit with spatially heterogeneous magnitudes throughout the computational domain. Figure 3 shows the load-displacement plot for realization 2.



(a) Function $f(x)$



(b) Deformed shape after tensile displacement boundary condition is applied. The colorbar shows the displacement field. Deformation has been magnified 4000 times.

Figure 2. Function $f(x)$ and the deformed shape after this function is applied

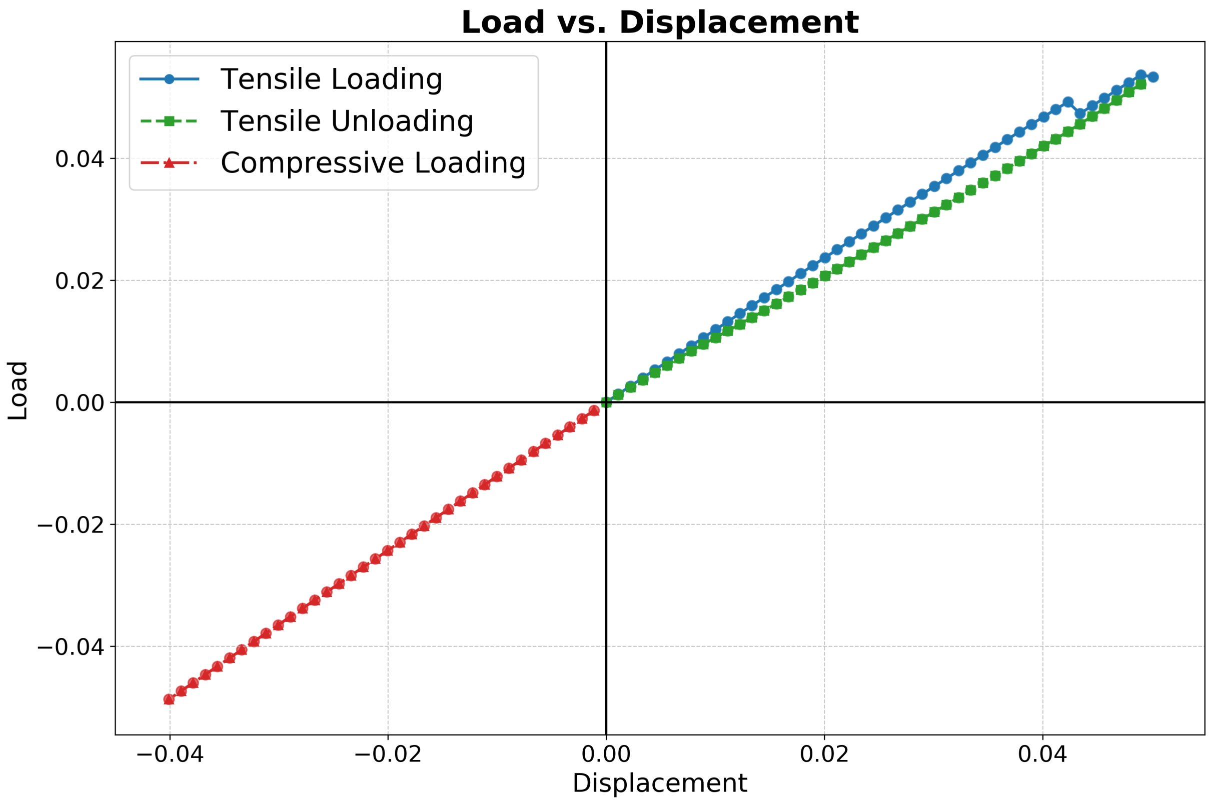


Figure 3. Load-displacement for realization 2

As shown in Figure 3, the load-displacement relationship during tensile unloading differs from that during tensile loading. This difference occurs because cracks form during the tensile loading process. Consequently, for any given load, the displacement is higher during the tensile unloading cycle compared to the tensile loading cycle.

3. Files Description

Figures 4 and 5 illustrate the dataset organization.

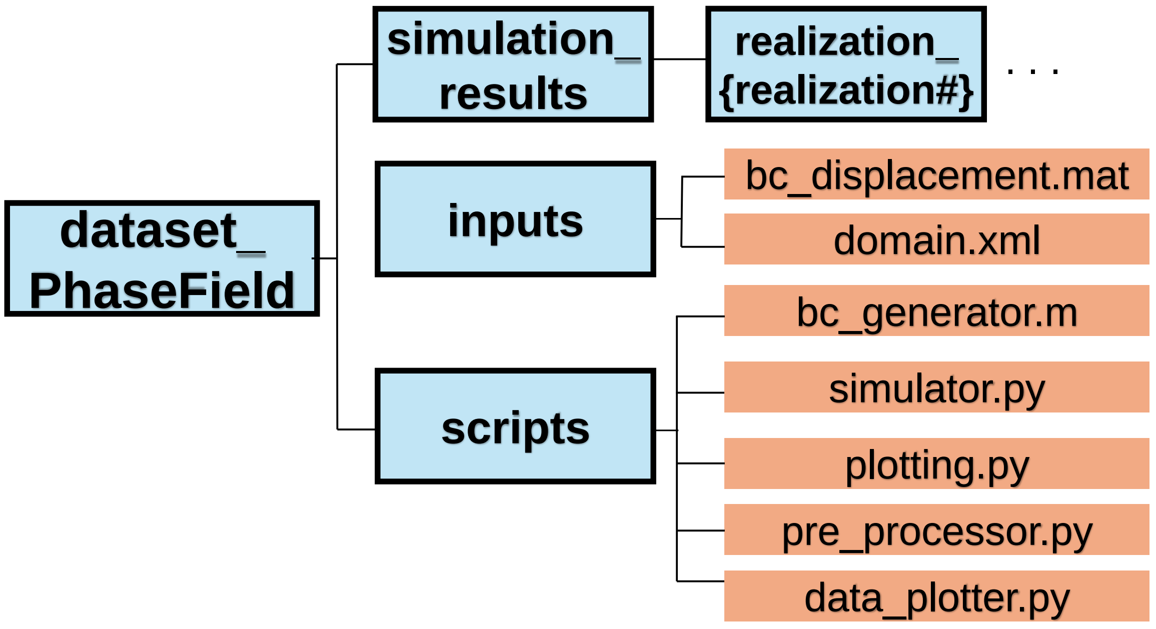


Figure 4. Dataset organization overview. Blue boxes represent folders, while orange boxes represent files.

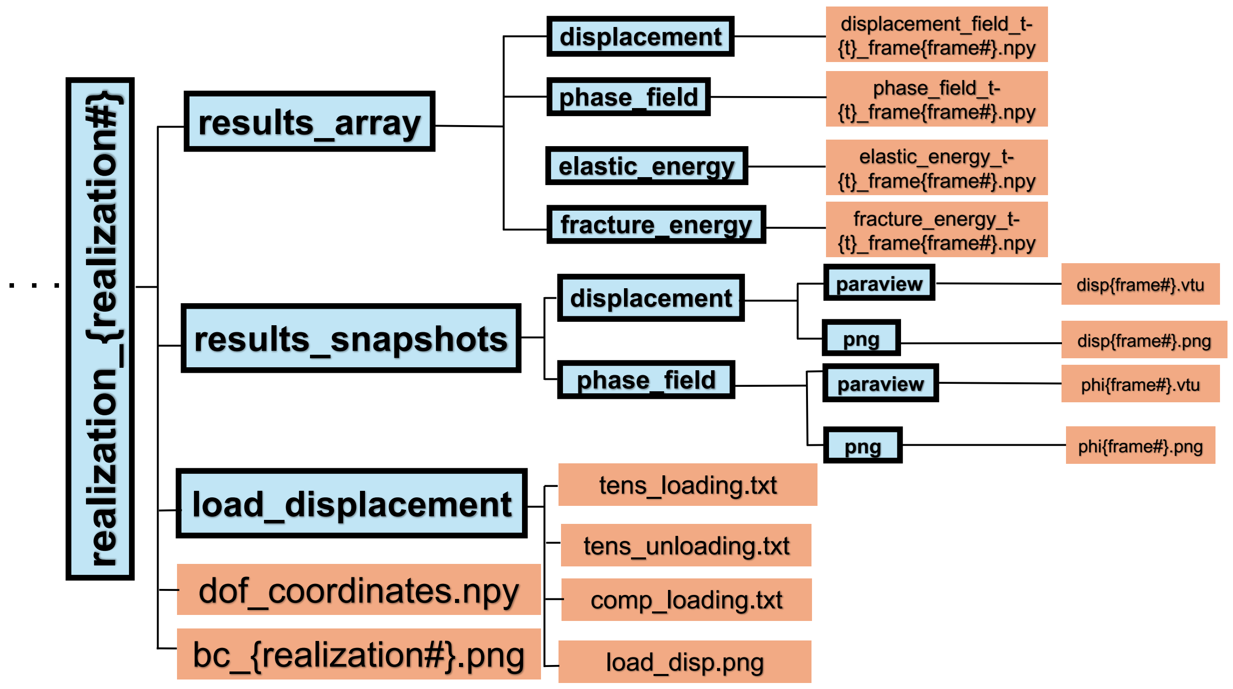


Figure 5. Dataset organization within each realization. Blue boxes represent folders, while orange boxes represent files.

The folder hierarchy and file organization presented in Figures 4 and 5 are detailed as follows:

a. Folder `simulation_results`: This directory contains the phase-field simulation results for distinct realizations, along with the nodal coordinates of the finite element mesh.

b. Folder `inputs`: This directory contains the phase-field simulation input files: `bc_displacement.mat` and `domain.xml`

c. Folder `scripts`: This directory includes the scripts implemented for systematic dataset generation and processing.

d. Folder `realization_{realization#}`: This directory contains computational results from phase-field simulations, with each subdirectory representing an individual statistical realization of the numerical experiments.

e. File `bc_displacement.mat`: This file is the output of file `bc_generator.m`, and it stores different boundary conditions that are used for dataset generation.

f. File `domain.xml`: The mesh configuration shows the computational domain geometry and is consistent with the discretization employed in previous investigations of cavity-induced fracture propagation under cyclic loading conditions[1]. The finite element mesh was generated utilizing the Gmsh [7] computational framework.

g. File `bc_generator.m`: This MATLAB implementation generates one hundred Gaussian Random Field (GRF) realizations that serve as prescribed boundary conditions for mechanical loading. The numerical procedure employs the Chebfun computational framework [8], with the resultant GRF datasets stored in the output file `bc_displacement.mat`.

h. File `simulator.py`: This numerical implementation, developed within the FEniCS finite element framework [9], computes the mechanical response of the computational domain subject to varying prescribed boundary conditions.

i. File `plotting.py`: This post-processing script exports visualization files from Paraview files in the `results_snapshots` directory into image format utilizing a customized colorimetric scheme. The generated images facilitate subsequent implementation of computer vision algorithms on the dataset. Additionally, the implementation generates load vs. displacement response curves characterizing the mechanical behavior during tensile loading, tensile unloading, and compressive loading phases. Furthermore, the corresponding graphical representations of prescribed boundary conditions are stored within their respective `realization_{realization#}` subdirectories.

j. File `pre_processor.py`: This script processes the displacement and phase-field data for all realizations and maps them onto a uniform 128×128 grid. Since the domain contains an internal hole, the displacement and phase-field values within this hole are set to zero. Finally, the script saves the first 900 samples into `train.npz`, and the remaining 100 samples into `test.npz`.

k. File `data_plotter.py`: This script loads the data from `train.npz` and/or `test.npz`, and visualizes the displacement and phase-field results for any specified sample and time step.

l. Folder `results_array`: This directory contains .npy files encompassing the primary field variables of the system, including displacement and phase field parameters. The repository additionally stores elastic and fracture energy quantities, which facilitate inverse modeling applications. Each physical variable is preserved in .npy format, containing the corresponding nodal values. The nomenclature convention adheres to the following structure:

```
vaiable_name_t-{t}_frame{frame#.npy}
```

where t denotes the temporal discretization corresponding to quasi-static loading and unloading conditions, while `frame#` represents the sequential index of the computational output. For instance, `frame`0 corresponds to the initial state of quasi-static tensile loading at the associated time step t. The parameter t decreases after some time, indicating tensile unloading, and then assumes negative values beyond a specific frame, indicating the transition to the compressive loading regime.

m. Folder `results_snapshots`: This directory includes visualization outputs of displacement and phase field variables, including both image files in png format and Paraview-compatible data structures to facilitate subsequent analysis and advanced visualization procedures.

n. Folder `load_displacement`: This folder includes data for displacement and traction integrated over the outer surface of the boundary during tensile loading, tensile unloading, and compression loading at quasi-static loading steps. The data are in the following format:

- `tens_loading.txt`: Traction and displacement data for tensile loading
- `tens_unloading.txt`: Traction and displacement data for tensile unloading
- `comp_loading.txt`: Traction and displacement data for compression loading
- `load_disp.png`: Traction vs displacement plot for tensile loading, tensile unloading, and compression loading

o. File `dof_coordinates.npy`: The file `dof_coordinates.npy` contains the nodal coordinates corresponding to the data stored in the `.npy` files within each `realization_realization#` directory. As both the mesh geometry and data storage protocol remain constant across all realizations, the nodal coordinates in each `dof_coordinates.npy` file are identical.

p. File `bc_{realization#}`: Files named `BC_realization#.png` provide visualizations of the boundary conditions applied to the domain's sides, as generated by the Gaussian random field (GRF).

This structured organization ensures easy access to the dataset, making it well-suited for a wide range of applications, including both forward and inverse modeling tasks. Our full dataset consists of 1000 distinct cases. However, due to the large size of the data, we have provided the files discussed above for a representative subset of 100 cases. To specifically support training and testing of machine learning models, we have prepared dedicated training and testing datasets, as detailed in Section 4.

4. Dataset preparation for learning the ML model

The dataset comprises 1000 distinct boundary conditions, which have been preprocessed and stored in the `train.npz` (containing 900 samples) and `test.npz` (containing 100 samples) files using the `preprocessor.py` script. The `data_plotter.py` script demonstrates how to load and visualize data from the `train.npz` and `test.npz` files. For all the boundary conditions, all relevant codes used to generate the dataset, along with simulation results for the first 96 samples, are provided in the `simulation_results.zip`, `inputs.zip`, and `scripts.zip` folders, with some notable exceptions in the loading cycles.

Specifically, realizations 28 and 60 lack both tensile unloading and compressive loading phases, while realization 77 excludes the compressive loading cycle. These omissions stem from convergence difficulties encountered by the numerical solver under the default parameters. While these convergence issues could potentially be resolved through parameter adjustment or modification of loading steps, we maintained consistent solver settings and loading protocols across all realizations to ensure methodological uniformity in the dataset generation process.

It should be emphasized that the `train.npz` and `test.npz` files contain data pertaining to the tensile cycle. During the compression cycle, identical crack path is observed; however, the diffusivity-related phenomena are no longer present.

[1] M. Hakimzadeh, V. Agrawal, K. Dayal, and C. Mora-Corral, Phase-field finite deformation fracture with an effective energy for regularized crack face contact, Journal of the Mechanics and Physics of Solids **167**, 104994 (2022).

[2] S. Sane and W. G. Knauss, On interconversion of various material functions of pmma, Mechanics of time-dependent materials **5**, 325 (2001).

[3] A. S. Mijailovic, S. Galarza, S. Raayai-Ardakani, N. P. Birch, J. D. Schiffman, A. J. Crosby, T. Cohen, S. R. Peyton, and K. J. Van Vliet, Localized characterization of brain tissue mechanical properties by needle induced cavitation rheology and volume controlled cavity expansion, Journal of the Mechanical Behavior of Biomedical Materials **114**, 104168 (2021).

[4] M. P. Milner and S. B. Hutchens, Dynamic fracture of expanding cavities in nonlinear soft solids, Journal of Applied Mechanics **88** (2021).

[5] S. Raayai-Ardakani, D. R. Earl, and T. Cohen, The intimate relationship between cavitation and fracture, Soft matter **15**, 4999 (2019).

[6] J. Y. Kim, Z. Liu, B. M. Weon, T. Cohen, C.-Y. Hui, E. R. Dufresne, and R. W. Style, Extreme cavity expansion in soft solids: Damage without fracture, Science advances **6**, eaaz0418 (2020).

[7] C. Geuzaine and J.-F. Remacle, Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities, International journal for numerical methods in engineering **79**, 1309 (2009).

[8] T. A. Driscoll, N. Hale, and L. N. Trefethen, Chebfun guide (2014).

[9] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, The fenics project version 1.5, Archive of numerical software **3** (2015).