applied sciences

MDPI

Article

Embedding Consistent Recurrent Neural Networks for Fracture Modelling in Large Scale Shell Structures

Muhammed Adil Yatkın [1,*], Mihkel Kõrgesaar [1] and Hasan Kurban [2]

[1] School of Engineering, Kuressaare College, Tallinn University of Technology, Tallinna 19, 93811 Kuressaare, Saaremaa, Estonia; mihkel.korgesaar@taltech.ee
[2] College of Science and Engineering, Hamad Bin Khalifa University, Doha P.O. Box 34110, Qatar; hkurban@hbku.edu.qa
* Correspondence: muyatk@taltech.ee

Abstract

Data-driven surrogates could replace phenomenological fracture criteria in structural-scale shell element simulations, but a recurrent surrogate can operate inside an explicit solver only if its predictions are independent of the increment count. This condition was tested by embedding a trained recurrent neural network (RNN) localisation surrogate as the erosion criterion in an Abaqus finite element material subroutine. A SimpleRNN was compared against the proposed consistent RNN (ConsRNN), whose transition function makes predictions converge with increasing increment count. Both were trained on 19,098 bilinear strain paths from solid-element simulations of one structural steel. At the 400-increment training resolution, the SimpleRNN was more accurate but drifted with the increment count, whereas the embedded ConsRNN's prediction varied by only 1.5% between 400 and 200,000 increments. In a clamped steel plate indented to rupture, the force–displacement curves coincided to the peak for network sampling intervals differing by over an order of magnitude. The ConsRNN and an experimentally validated two-factor-scaling (2FS) criterion for the same steel predicted nearly identical fracture onset and peak force at indistinguishable computational cost. No benefit from the surrogate's history dependence was demonstrated, because the crack-path elements were loaded nearly proportionally, and ranking the criteria requires a fracture experiment. The paper's main contribution is the embedding of a customised recurrent cell with a non-standard transition function as a live fracture criterion in a structural-scale explicit FE analysis.

Keywords: consistent recurrent neural networks; surrogate modelling; finite element method; Abaqus; VUMAT; path-dependent damage prediction; computational mechanics

Check for updates

Academic Editor: Valerio Belardi

Received: 20 August 2026
Revised: 21 October 2026
Accepted: 26 October 2026
Published: 29 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Machine learning models are increasingly used as surrogates for the nonlinear relationship between deformation and mechanical response in engineering structures [1–4]. Instead of solving explicitly defined constitutive relations within finite element (FE) frameworks, data-driven models learn the input–output mapping directly from simulation data [2], motivating hybrid frameworks in which individual simulation components [5–9], or even entire simulation stages [10–13], are replaced by learned surrogates. Such models allow knowledge gained from expensive simulations to be reused efficiently across analyses, and a single network formulation can represent multiple material classes through transferable learned representations [14]. In this spirit, Yatkın and Kõrgesaar [15] introduced a mapping from the deformation history of a material unit, sized to a large shell element (an element

length five times the sheet thickness in the present application), to its strain localisation response and demonstrated that this relationship can be learned directly from high-fidelity simulation data. In the present work, we take this localisation surrogate from offline validation to deployment, embedding a consistency-preserving recurrent network directly inside an explicit finite element solver.

Recurrent neural networks (RNNs) are natural candidates for learning such history-dependent mappings because their internal state advances across loading increments in the same way an FE solver advances its solution. Unlike fully connected neural networks, which require deformation history to be represented through hand-crafted state variables [16,17], recurrent architectures naturally retain historical information within their hidden state and have been used to learn homogenised constitutive behaviour [5], the response to arbitrary non-proportional loading paths [18], and parametrised finite element simulations with varying mesh topologies [19]. However, predictive accuracy alone is insufficient for deployment as a finite element surrogate. Since explicit FE solvers determine their own increment sizes, surrogate predictions must remain consistent under temporal refinement. Specialised recurrent transition functions have therefore been developed to produce consistent estimates during simulation. Bonatti and Mohr [9] established self-consistency as a fundamental property of recurrent transition functions intended for use as surrogates inside FEM-based simulations, and Yatkin et al. [20] showed that any sequential learning architecture deployed as a surrogate must satisfy the truncation and consistency requirements. Of these two requirements, the present paper examines consistency, which is defined in Section 2.5.

This observation motivated several consistency-preserving recurrent formulations, including the linearised minimal state cell (LMSC) [9] and its extensions [21–23]. Our earlier studies [24–26] extended the localisation surrogate to recurrent architectures, leading to the consistent recurrent neural network (ConsRNN) [27]. ConsRNN differs in construction from the LMSC. Namely, LMSC achieves self-consistency by learning a compact internal state whose update rule embeds physics-motivated variables tailored to the constitutive response. ConsRNN, in contrast, retains a standard recurrent hidden state and secures consistency purely through its mathematical design, a norm-scaled convex-combination state-transition update that can be interpreted as an explicit-Euler discretisation of an arc-length neural ordinary differential equation (Table 1). Throughout this paper, consistency in this sense means that the prediction converges under refinement of the discretisation of a fixed strain path rather than depending on how finely that path is sampled; the property is stated precisely in Section 2.5. Because consistency follows from the transition function itself rather than from embedded physical quantities, the property is architecture-intrinsic and transfers across problem settings, as demonstrated in Yatkin et al. [27] on two independent datasets. Moreover, ConsRNN produced consistent damage-initiation predictions under varying increment counts on both synthetic and FEM-generated data, including the elasto-plastic benchmark of Bonatti and Mohr [28]. A consequence of the ConsRNN construction is that consistency does not depend on physically interpretable state variables, which is why the same cell could be applied without modification to two independent datasets, as in [27]. However, the tradeoff is that the norm-scaled update sacrifices accuracy at a single fixed increment count in return for consistency across increment counts.

Surrogate deployment goes beyond defining a mapping and training an architecture. Inside an FE solver, the trained network must execute incrementally while preserving its internal state across solver-driven time steps. The documented deployment routes rely either on automated Keras-to-Fortran frameworks, which support standard NN layers [29,30], or on inference hard-coded directly in Fortran [31,32]. The networks hard-coded in these works are memoryless and carry no hidden state from one increment to the next. Bonatti and Mohr [9] implemented the recurrent LMSC in a user material subroutine as a material

model, whereas the present work embeds a different consistency-preserving recurrent cell, the ConsRNN, as a fracture criterion in shell elements. The automated route does not cover the ConsRNN, whose consistency-preserving transition function is non-standard. So far, the network has been validated only in Python/TensorFlow [27]. We therefore follow the hard-coding strategy and deploy the ConsRNN. Its architecture is optimised through Bayesian hyperparameter tuning, and the network is then reimplemented in pure Fortran, with the learned weights transferred directly into an Abaqus VUMAT. Within a structural-scale FE simulation, the embedded network serves as the criterion for the path-dependent onset of ductile fracture in shell elements. Trained on localisation data from high-fidelity solid-element simulations, the network supplies the shell elements with information how damage evolves depending on the history of deformation. In empirical 2FS criterion, which is used in structural calculations for comparison, history effects are included only in the average sense. The SimpleRNN, a standard ungated recurrent cell, serves as the baseline. Under solver-driven computation with varying increment sizes, the ConsRNN retains its consistency but the SimpleRNN does not, so the consistency comes from the network formulation rather than the embedding itself.

Table 1. Comparison of ConsRNN and SimpleRNN cells in terms of input transformation, normalisation, and state updates.

Step	ConsRNN	SimpleRNN
Input	$X_t = (\Delta\varepsilon_{1,t}, \Delta\varepsilon_{2,t})$	$X_t = (\varepsilon_{1,t}, \varepsilon_{2,t}, \phi_t)$
Normalisation	$s_t = \mathrm{clip}\left(\lVert X_t \rVert_2, 10^{-15}, 10^{2}\right), \quad \hat{X}_t = \dfrac{X_t}{s_t}$	-
Transition	$A_t = \tanh(\hat{X}_t W_x + S_{t-1} W_s + b_h)$	$S_t = \tanh(X_t W_x + S_{t-1} W_s + b_h)$
Update	$S_t = S_{t-1} + (A_t - S_{t-1}) \cdot s_t$	-
Output	$\sigma(S_t \cdot W_{\text{out}} + b_y)$	$\sigma(S_t \cdot W_{\text{out}} + b_y)$

This paper tests the hypothesis that discretisation consistency is the property that decides whether an RNN can be embedded inside an explicit FE solver. Under this hypothesis, if the transition function guarantees this property, the damage predictions of the embedded network are governed by the loading path itself and not by how the solver divides that path into increments. Section 2 presents the Bayesian hyperparameter optimisation of the ConsRNN and SimpleRNN architectures, and Section 3 details the pure-Fortran implementation and the transfer of the learned weights into the Abaqus VUMAT. Section 4 tests the hypothesis in two different simulation scenarios: a single large shell element under bi-linear loading path and a plate indented with a rigid cone. Section 5 discusses the main findings and limitations. Section 6 concludes the paper and outlines future work.

2. Neural Network Model Development and Evaluation

Before deployment within the FEM solver, the proposed NN architectures are first developed and evaluated through a standard machine learning workflow in Python 3.10/TensorFlow 2.15. The workflow comprises data analysis, architecture design, Bayesian hyperparameter optimisation, and comprehensive qualitative and quantitative performance evaluation to ensure a reproducible comparison between the candidate models. The following subsections describe each stage of this workflow and motivate the selection of the architecture used for the subsequent Fortran implementation. Sections 2.1–2.5 cover the material model, the dataset, the two architectures, and their hyperparameter optimisation, while the remaining subsections report the offline results used to select the architecture carried forward to the Fortran implementation.

2.1. Material Model

All analyses in this paper are performed with the rate-independent, isotropic elastic–plastic structural steel with yield stress of $\sigma_0 = 272$ MPa. Material follows von Mises yield, associated flow rule, and isotropic hardening with elastic modulus of $E = 200$ GPa and Poisson ratio of $\nu = 0.3$. The flow stress is described with modified Swift law

$$\sigma_f = K\left(\varepsilon_p + \varepsilon_0\right)^n, \tag{1}$$

where ε_p is the equivalent plastic strain, and $K = 620$ MPa, $n = 0.2$, and $\varepsilon_0 = 0.0162$ are the hardening parameters.

2.2. Path-Dependent Damage Prediction Using the Bilinear Loading Dataset

In [15], fine solid-element simulations were used to understand how in-plane deformation history affects damage evolution D and subsequent localisation in the studied steel material. Bilinear loading paths were defined in terms of the principal major ε_1 and minor ε_2 plastic strain together with orientation angle $\cot(\phi)$. A total of 19,098 distinct bilinear paths, each discretised into 400 increments, were simulated. The paths were generated by random sampling from continuous parameter ranges rather than on a grid, so the random split over paths does not place test paths at fixed grid offsets from training paths; the angle range and segment-length rule are given in [15]. In the present study, the SimpleRNN and ConsRNN architectures were trained on this dataset, split into 13,367 training, 2866 validation, and 2865 test paths. The validation set guided model selection during hyperparameter tuning, and final performance was reported on the held-out test set using the best-trained weights.

Because the sigmoid output of both networks lies between 0 and 1, the damage target is mapped for training through Equation (2), which takes non-negative damage values into $[0, 1)$, and the predictions are recovered at inference through Equation (3):

$$f(D_t) = 1 - e^{-D_t} \tag{2}$$

$$g(f(D_t)) = -\log(1 - f(D_t) + \epsilon) \tag{3}$$

Here, ϵ is a small numerical guard that keeps the logarithm bounded as $f(D_t) \to 1$. Throughout this paper, errors on the converted scale are computed on $f(D_t)$, and errors on the original damage scale are computed on D_t.

2.3. Architectures

Our previous study [20] showed that conventional recurrent architectures such as GRU and LSTM fail to satisfy the consistency requirements of path-dependent deformation and that their gating mechanisms also complicate a fixed-form Fortran VUMAT implementation and add computational cost. In contrast, the SimpleRNN and ConsRNN cells each use a single recurrent transition and are comparably simple, matching in depth, capacity, and the absence of gating. This makes the SimpleRNN a natural baseline for the proposed consistency-preserving transition formulation. A single-layer SimpleRNN also proved sufficient to learn the bilinear deformation paths over 400 increments at low computational cost, so both architectures were restricted to a single recurrent layer. Each was tuned through repeated training runs.

The two cells differ in their inputs and in transition function that updates the state vectors (Table 1). At each step t, where t indexes the loading increments rather than physical time, the SimpleRNN takes the accumulated major and minor strains together with the loading orientation, $(\varepsilon_{1,t}, \varepsilon_{2,t}, \phi_t)$, as shown in Figure 1a, where, by the convention used throughout this work, $\cot \phi_t = \Delta\varepsilon_{2,t}/\Delta\varepsilon_{1,t}$. The ConsRNN architecture takes only the strain

increments $\Delta\varepsilon_{1,t} = \varepsilon_{1,t} - \varepsilon_{1,t-1}$ and $\Delta\varepsilon_{2,t} = \varepsilon_{2,t} - \varepsilon_{2,t-1}$ and normalises them internally, as shown in Figure 1b. Because the input representation, input dimensionality, and normalisation strategy differ as well, the comparison evaluates two complete surrogate formulations rather than the transition function in isolation. In both cases, the strains are principal plastic strains. The FE implementation advances the network only on plastic increments, so an elastic increment leaves the network state and the predicted damage unchanged.

In Table 1, X_t denotes the cell input at time step t, that is, the strain increments $(\Delta\varepsilon_{1,t}, \Delta\varepsilon_{2,t})$ for ConsRNN and the accumulated strains with orientation $(\varepsilon_{1,t}, \varepsilon_{2,t}, \phi_t)$ for SimpleRNN. Furthermore, $\hat{X}_t$ is its normalised form, s_t the guarded Euclidean norm of the input increment, S_t the recurrent state, and A_t the transition activation obtained by applying the tanh nonlinearity to the affine combination of input and previous state. The matrices W_x and W_s are the trainable input and recurrent weights, b_h the recurrent bias, W_{out} the output weight, and b_y the output bias. These parameters are learned during training rather than prescribed analytically. Here, $\|X_t\|_2$ denotes the Euclidean norm of the input increment and σ the sigmoid function. The bounds entering s_t act purely as numerical safeguards, the lower one against division by zero and the upper one against pathologically large increments. All strain increments encountered in this work satisfy $10^{-15} \leq \|X_t\|_2 \leq 10^2$, so $s_t = \|X_t\|_2$ throughout, and the state update reduces to the unguarded form used in Equation (4). The paths considered here have lengths of order unity in principal strain space and are traversed in hundreds to hundreds of thousands of increments, so $s_t < 1$ throughout and the update is a convex combination of S_{t-1} and A_t. Both networks start from the zero state $S_0 = 0$ (Appendix A).

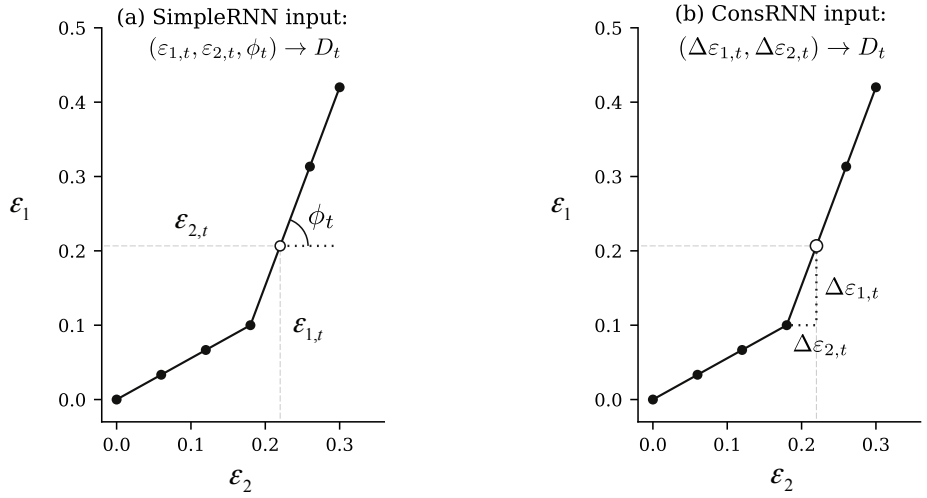


Figure 1. Input representations of the same bilinear loading path for the two architectures in principal strain space. The empty marker highlights the state of deformation at step t. Panel (**a**) shows the input to the SimpleRNN and panel (**b**) the input to the ConsRNN. Both panels depict the same underlying loading path, but the SimpleRNN encodes it through the accumulated strain state, whereas the ConsRNN encodes it through the incremental strain difference.

Both formulations share one caveat: neither constrains the predicted damage to be non-decreasing, since the output in Table 1 is recomputed from the current state at every step. For the ConsRNN, this follows directly from the update rule, since the state increment

$$\Delta S_t = S_t - S_{t-1} = (A_t - S_{t-1})\,\|X_t\|_2 \tag{4}$$

is negative whenever $A_t < S_{t-1}$, and the SimpleRNN state is equally free to decrease. The predicted damage D can therefore decrease along a path, which is non-physical, although the damage target of Section 2.2 is non-decreasing by construction. In addition, at the zero initial

state $S_0 = 0$, the output mapping returns a small positive offset $D_0 = -\log(1 - \sigma(b_y) + \epsilon)$, so every predicted damage history starts from D_0 rather than from zero. To obtain a non-decreasing damage measure, the running maximum of the network output is used in all structural simulations:

$$D_{\text{irr}}(t) = \max_{\tau \le t} D(\tau). \tag{5}$$

Since D remains below unity until it first reaches unity, D_{irr} reaches unity at exactly the same instant, so the instant of element deletion is unchanged, and only the damage values before deletion differ. D_{irr} also does not feed back into the network state, so the discretisation consistency of the ConsRNN is unaffected. It is, however, a correction applied to the output rather than to the formulations themselves, and a network whose predicted damage is non-decreasing by construction while preserving consistency is left for future work.

2.4. SimpleRNN-Based NN Architecture

The first architecture pairs the Keras/TensorFlow [33] SimpleRNN cell, whose update follows the standard formulation given in Table 1, with a single fully connected output layer using sigmoid activation. Its inputs are supplied at each time step through the mapping defined in Figure 1a. Each candidate model was trained for 100 epochs, and the hyperparameters were tuned through Bayesian optimisation over 20 trials in a joint search over the learning rate, batch size, and number of hidden units. The trial-by-trial progression is shown in Figure 2.

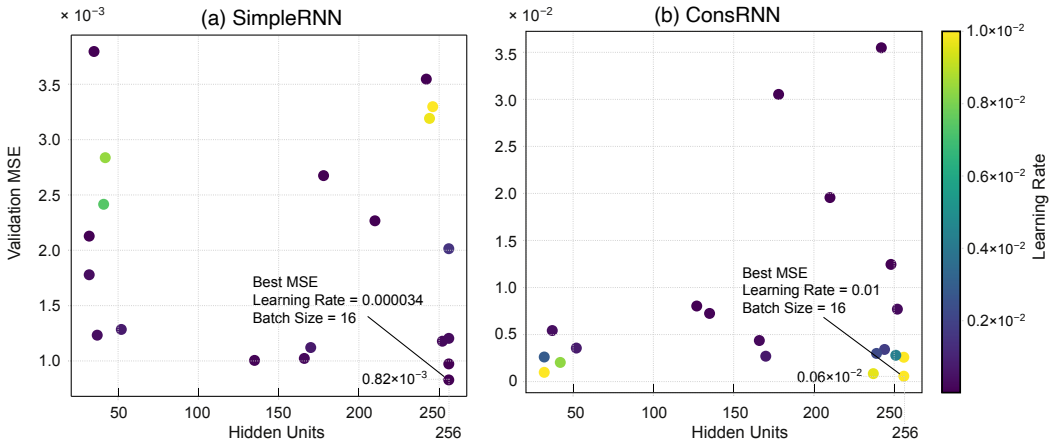


Figure 2. Hyperparameter tuning of the (**a**) SimpleRNN and (**b**) ConsRNN-based architectures over 20 Bayesian optimisation trials on the validation dataset. Each marker denotes one trial, positioned by hidden units and validation MSE, coloured by learning rate. The lowest MSE trials are highlighted. The deployed ConsRNN-B256 differs in batch size and learning rate and was selected for its lower localisation error on the bilinear FLC paths at increment counts above the 400-increment training resolution (Table 2).

Table 2. RMSE of the major strain ε_1 at localisation relative to the bilinear FLC for ConsRNN-B16 and ConsRNN-B256. Values in parentheses give the number of paths, out of 40, for which no localisation is predicted; these are excluded from the RMSE.

Increments	$\cot \boldsymbol{\phi}_1 = -0.55$, $\boldsymbol{R}_1 = 0.12$		$\cot \boldsymbol{\phi}_1 = -0.75$, $\boldsymbol{R}_1 = 0.4$	
	ConsRNN-B16	**ConsRNN-B256**	**ConsRNN-B16**	**ConsRNN-B256**
400	0.0103 (1)	0.0183	0.0138 (5)	0.0218 (9)
10^3	0.1610 (2)	0.0200	0.0884	0.0209 (7)
10^4	0.1446 (2)	0.0208	0.0947	0.0202 (7)
10^5	0.1453 (2)	0.0209	0.0951	0.0202 (7)

The best configuration uses 256 hidden units, a learning rate of 3.4×10^{-5}, and a batch size of 16, yielding a mean squared error of 0.0010 on the training set and 0.0008 on the validation set (Figure 2a). The close train–validation agreement indicates a well-conditioned fit. Generalisation is assessed separately on the held-out test set (Table 3).

Table 3. Test set results of SimpleRNN and ConsRNN-B16 (lowest-validation-MSE configurations) on the converted scale $f(D_t)$ and the original damage scale D_t. nRMSE: RMSE normalised by the range of the true values; P95 abs err: 95th percentile of the absolute error.

Output Type	Model	RMSE	MAE	nRMSE	MAPE (%)	P95 Abs Err	R^2
Converted Output	SimpleRNN	0.0281	0.0188	0.0289	6.23	0.0586	0.9832
	ConsRNN-B16	0.0236	0.0150	0.0243	5.40	0.0502	0.9882
Original Damage Output	SimpleRNN	0.0616	0.0378	0.0174	7.92	0.1161	0.9784
	ConsRNN-B16	0.0440	0.0281	0.0124	6.71	0.0869	0.9890

2.5. ConsRNN-Based NN Architecture

The consistency property of the ConsRNN cell follows from its transition function, see Table 1. Rather than taking the accumulated strains directly, the cell operates on the strain increments between successive time steps: it forms a candidate state, subtracts the previous state, and scales the difference by the Euclidean norm of the input increment before applying the update. Because each state advance is proportional to the increment norm, the prediction accumulated over a fixed path converges as the increments are refined rather than drifting with their number. This property is referred to as discretisation consistency, or consistency for short: when the same strain path is traversed with different increment sequences, the predicted final state converges to a common value as the increments are refined, so the prediction is set by the path and not by how finely it is sampled. It is the requirement that Bonatti and Mohr [9] term self-consistency; its formal treatment is given in [20].

The second architecture pairs the ConsRNN cell with a single fully connected output layer with sigmoid activation. Its inputs are the strain increments supplied through the mapping in Figure 1b. Training and tuning followed the SimpleRNN protocol: 100 epochs per candidate and 20 Bayesian-optimisation trials, in a joint search over the number of hidden units in $[32, 256]$, the learning rate on a log-uniform scale over $[10^{-5}, 10^{-2}]$, and the batch size in $[16, 256]$ (Figure 2b). Because the trials of both searches are ranked on the same validation set, the selected trial is the best found within the search rather than a global optimum, and the held-out test set is not used until the comparison of Table 3.

The lowest-validation-MSE ConsRNN trial uses 256 hidden units, a learning rate of 1.0×10^{-2}, and a batch size of 16, with a mean squared error of 0.0006 on both the training and validation sets. Comparison in Figure 3 shows that the SimpleRNN trials cluster at low validation MSE, whereas the ConsRNN trials scatter more widely, although the best ConsRNN trial (0.0006) lies below the best SimpleRNN trial (0.0008).

Two ConsRNN configurations are carried forward, named by their batch size. Both are among the 20 Bayesian trials in Figure 3, with their learning rates set by the optimisation, and only these two configurations were compared on the forming-limit paths. ConsRNN-B16 is the lowest-validation-MSE trial above and serves the fixed-resolution comparison with the SimpleRNN on the held-out test set (Table 3). ConsRNN-B256, trained with a batch size of 256, is the configuration embedded in Fortran and used in all finite element results; which configuration each offline result uses is stated where that result is reported. ConsRNN-B256 was chosen for deployment because an explicit solver, not the user, sets the number and size of the increments, so accuracy must hold away from the 400-increment training

resolution. On the forming-limit paths of Section 2.8, both configurations remain consistent, but ConsRNN-B16 loses accuracy at the other increment counts, whereas ConsRNN-B256 does not (Table 2). ConsRNN-B256 is therefore not the validation-optimal ConsRNN, and because its batch size was chosen on the same paths that Section 2.8 reports, those results are not independent of the selection.

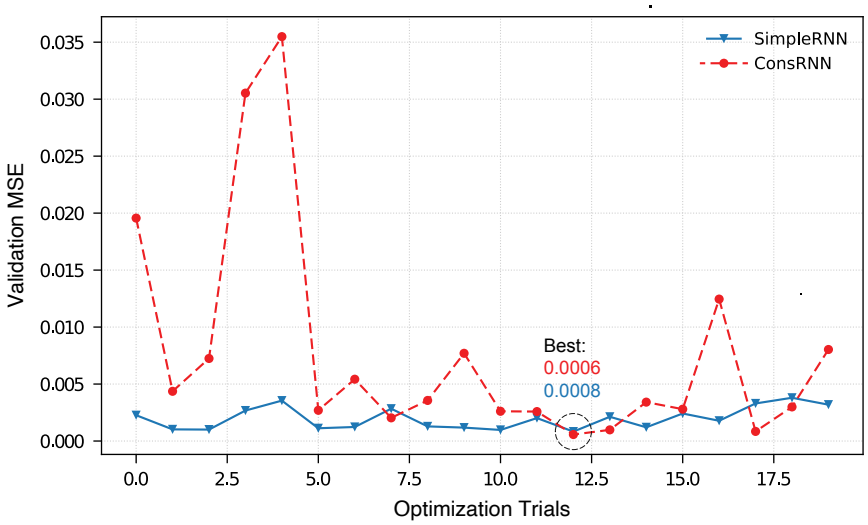


Figure 3. Validation mean squared error (MSE) across the 20 Bayesian optimisation trials for the SimpleRNN and ConsRNN architectures. The SimpleRNN trials vary less, but ConsRNN-B16 reaches the lowest value of either search (0.0006 versus 0.0008 for the best SimpleRNN trial).

2.6. Held-Out Test-Set Performance

The held-out test set results, summarised in Table 3, are consistent with the hyperparameter optimisation results on the bilinear loading dataset. For the lowest-validation-MSE configurations, ConsRNN-B16 yields lower errors than the corresponding SimpleRNN on all reported metrics and on both output scales. On the converted scale (Equation (2)), ConsRNN-B16 reduces the RMSE and the MAE while also raising the coefficient of determination. The same ordering holds on the original damage scale, where ConsRNN-B16 lowers the RMSE and improves R^2.

The advantage extends to the tail of the error distribution, since the 95th-percentile absolute error, which characterises worst-case behaviour across the test paths, is also lower for ConsRNN-B16 on both scales. These results characterise the two architectures at their respective validation optima. As noted in Section 2.5, the ConsRNN configuration carried forward to the VUMAT and all subsequent experiments is ConsRNN-B256, selected for its accuracy across increment counts rather than for its validation MSE at the training resolution, based on the two FLC sets of Section 2.8 (Table 2).

To assess the sensitivity of the deployed recurrent surrogates to random initialisation, the selected SimpleRNN and the deployed ConsRNN-B256 configurations were each retrained using five independent random seeds while keeping the architecture, hyperparameters, data split, and training procedure fixed. Table 4 reports the test MSE obtained for each seed together with the mean and standard deviation across the five runs. The ConsRNN configuration considered here is ConsRNN-B256, used in the subsequent deployment and discretisation-consistency analyses, whereas the ConsRNN reported in Table 3 is ConsRNN-B16. ConsRNN-B16 represents the best fixed-resolution predictive configuration, whereas ConsRNN-B256 was retained for deployment because of its behaviour under changing increment counts.

Table 4. Five-seed test MSE results for the selected SimpleRNN and the deployed ConsRNN-B256. The mean ± standard deviation is calculated across the five independent training runs.

Seed	Model	Converted Output	Original Damage Output
0	SimpleRNN	0.000856	0.00433
	ConsRNN-B256	0.00224	0.00673
1	SimpleRNN	0.000790	0.00379
	ConsRNN-B256	0.00242	0.00677
2	SimpleRNN	0.000813	0.00393
	ConsRNN-B256	0.00245	0.00700
3	SimpleRNN	0.000736	0.00343
	ConsRNN-B256	0.00116	0.00415
4	SimpleRNN	0.000778	0.00378
	ConsRNN-B256	0.00140	0.00442
Mean ± Std	SimpleRNN	0.000795 ± 0.000045	0.00385 ± 0.00033
	ConsRNN-B256	0.00193 ± 0.00061	0.00581 ± 0.00140

Across the five initialisations (Table 4), the SimpleRNN obtained a lower mean test MSE than ConsRNN-B256 for both the converted and the original damage output, with a smaller spread across seeds. The SimpleRNN therefore retains a fixed-resolution accuracy advantage and exhibits a smaller variation across random initialisations. The motivation for using the ConsRNN-B256 is consequently not superior fixed-resolution accuracy but its discretisation-consistency behaviour under changes in the number of increments, which is examined below.

2.7. Localisation and Damage Predictions for Bilinear Paths

To examine how the predicted localisation points depend on temporal resolution, we evaluated the selected SimpleRNN and ConsRNN-B256 in Python/TensorFlow along a single bilinear deformation path, discretised at 400, 1000, 10,000, and 100,000 increments (Figure 4). The path consists of a first segment of length $R_1 = 0.30$ with direction $\cot\phi_1 = 0.80$ and a second segment of length $R_2 = 0.67$ with $\cot\phi_2 = 0.60$. The ground-truth localisation point, marked by the empty circle in each panel, is obtained directly from the finite element simulation of this path, in which localisation occurs at increment 84,426 of 100,001. The path is held fixed across both panels, so that the only quantity changing between them is the number of increments used to traverse the identical strain history.

When the increment count changes from 400 to 10^3, the SimpleRNN prediction moves away from the ground-truth localisation point, where it stays fixed for higher increment counts. The ConsRNN predictions, in contrast, remain close to the reference and converge as the resolution is refined. This behaviour is consistent with the designed discretisation-consistency property of the norm-scaled ConsRNN update. Because each state advance is scaled by the norm of the strain increment, the accumulated prediction converges under refinement towards a value set by the path itself rather than by how finely it is sampled. This property makes the surrogate usable across the varying step sizes encountered in FE simulations. The two architectures differ in their input representation, as well as in the transition function, so this contrast reflects the two complete designs and not the transition function in isolation.

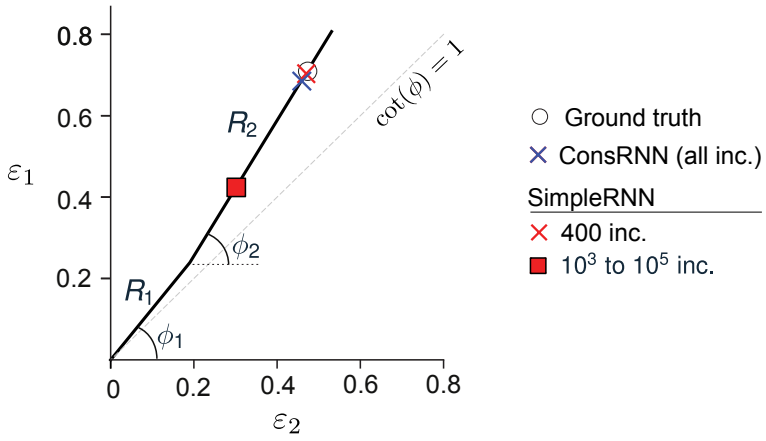


Figure 4. Localisation predictions of the SimpleRNN and the ConsRNN-B256 at 400 to 10^5 increments. The SimpleRNN prediction shifts away from the ground truth when the increment count changes, whereas the ConsRNN prediction remains consistent.

2.8. Forming Limit Curve (FLC) Predictions

The two selected architectures, the deployed ConsRNN-B256 and the SimpleRNN, were used to predict localisation points that define two forming limit curves (FLCs), shown in Figure 5. Each FLC is traced by fixing the first linear segment and sweeping the direction of the second segment: the first FLC fixes the first segment at length $R_1 = 0.12$ with direction $\cot\phi_1 = -0.55$ and the second at $R_1 = 0.4$ with $\cot\phi_1 = -0.75$. In both cases, the second segment is held at fixed length, while its direction $\cot\phi_2$ is varied across 40 values, each value yielding one point on the curve. Here, $\cot\phi = \Delta\varepsilon_2/\Delta\varepsilon_1$ denotes the ratio of the minor- to major-strain increment, following the convention used throughout this work. Each path was simulated at 100,000 increments and then downsampled to 10,000, 1000, and 400 increments for prediction, so that the FLCs probe both accuracy and discretisation consistency. The ground-truth localisation points were obtained from Marciniak–Kuczyński (MK) finite element simulations in Abaqus of these bilinear paths as in [15], following the dataset described in Section 2.2, with the localisation strain read directly from each simulated path. These reference localisations are plotted alongside the SimpleRNN and ConsRNN predictions.

As Figure 5 shows, the ConsRNN formulation is substantially less sensitive to increment count and, for these FLC cases, remains closer to the reference localisation points across the tested discretisations. The SimpleRNN predictions are nevertheless more stable than the GRU results reported in our previous study [27], where both architectures used the same accumulated-strain and loading-angle input representation. However, in the present ConsRNN–SimpleRNN comparison, the models differ in both their recurrent update and input representation; consequently, the observed difference cannot be attributed exclusively to the transition function. Table 2 compares the two ConsRNN configurations on the same FLC paths. ConsRNN-B16 is more accurate at the 400-increment training resolution, but its error increases by a factor of about 7 to 15 at the other increment counts, whereas the error of the deployed ConsRNN-B256 remains between 0.018 and 0.022 across all increment counts. Because that comparison held the input representation fixed, it points to the recurrent transition structure as one source of discretisation sensitivity.

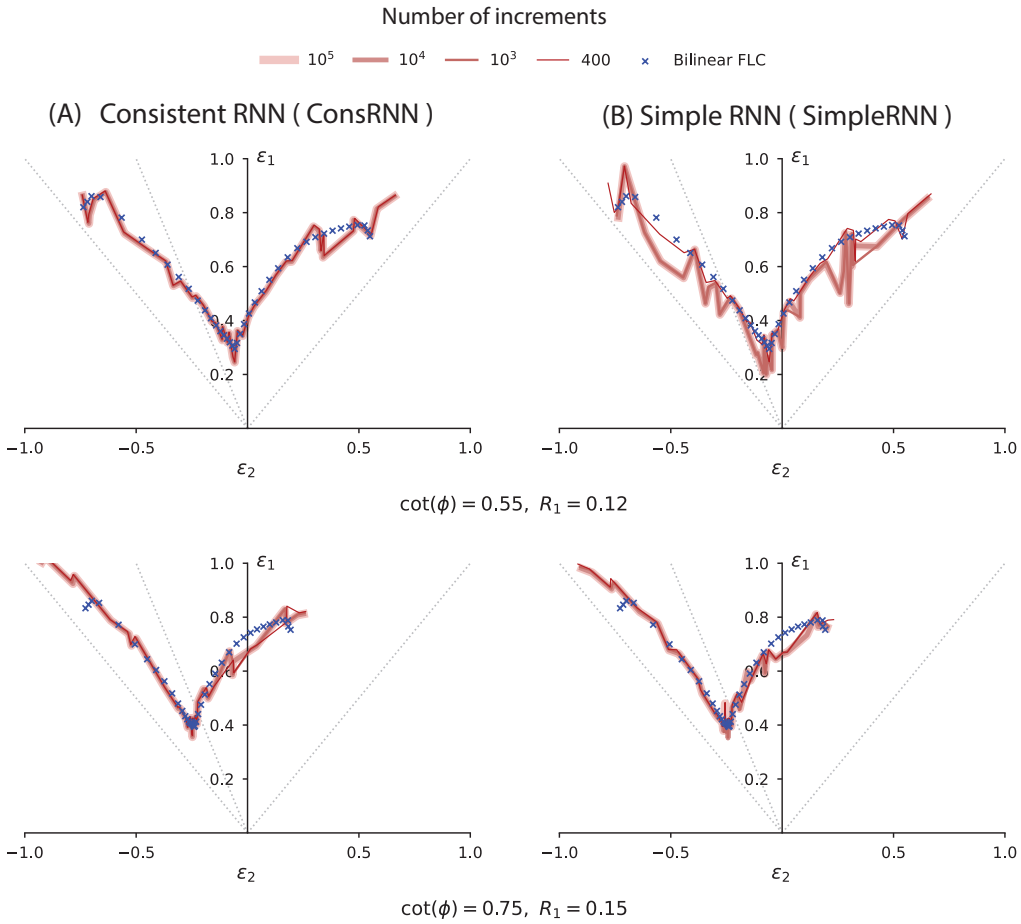


Figure 5. FLC predictions of ConsRNN-B256 (**A**) and the SimpleRNN (**B**) for the two first-segment definitions, $\cot\phi_1 = -0.55$, $R_1 = 0.12$ ((**top**) row) and $\cot\phi_1 = -0.75$, $R_1 = 0.4$ ((**bottom**) row). Line styles denote the increment count; crosses mark the reference localisation points.

2.9. Localisation Predictions for Multilinear Paths

Although both architectures were trained only on bilinear loading paths, the mappings in Figure 1a,b are defined incrementally, so the networks can be evaluated along paths with more segments than those seen during training. Whether the predictions remain accurate is quantified here. For this purpose, we used multilinear paths generated on the same principles as bilinear paths from [15], where details about each path and the ground truth localisation point can be found. Therefore, ground truth values are independent of the trained models. Localisation points predicted by the current architectures are shown in Figure 6. For ConsRNN predictions, the -B256 version was used because of its accuracy across increment counts rather than at the 400-increment training resolution (see Section 2.5).

On these multilinear paths, the SimpleRNN predicts the localisation points more accurately than the ConsRNN (Figure 6). At this fixed resolution, the standard state update of the SimpleRNN is free to fit the training discretisation more closely, and the difference can be qualitatively interpreted as the accuracy cost of the consistency-preserving update at a fixed resolution. Yet, such fixed resolution cannot be expected in structural simulations because the number of increments over which the network is advanced depends on the solver settings, element edge length, and its deformation characteristics and can thus differ widely between analyses. This justifies the implementation and deployment of ConsRNN-B256 in structural simulations reported next.

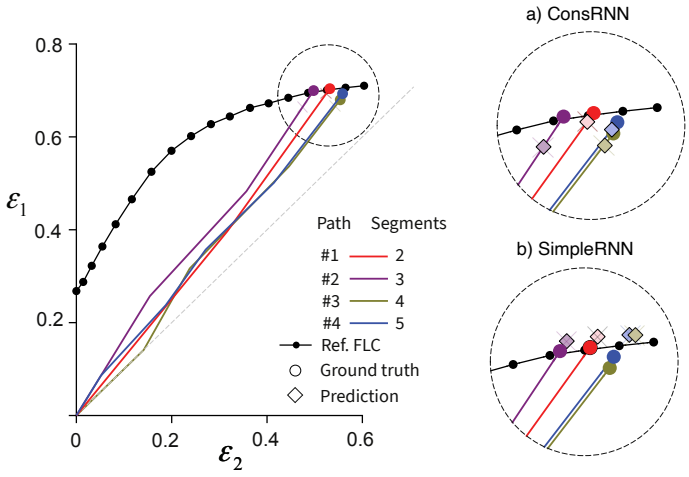


Figure 6. Localisation predictions for multilinear paths (two to five segments) from the deployed ConsRNN-B256 (**a**) and the SimpleRNN (**b**) architectures over 400 increments. The black curve with filled markers is the forming limit curve for proportional loading from [15].

3. Implementation of Neural Networks in VUMAT

Neural networks are typically built and trained in TensorFlow or PyTorch 2.10, whereas the finite-element simulations in this work are run in Abaqus, where the constitutive response is supplied through Fortran user subroutines. Bridging the two ecosystems is not straightforward. Frameworks such as Neural-Fortran [34] and PyTorch-ABAQUS UMAT [35] have been proposed to couple TensorFlow or PyTorch with Fortran, but they are built around standard Keras and PyTorch layers, such as convolutional and recurrent layers. The ConsRNN however uses a non-standard state-update mechanism that these bridging frameworks cannot directly represent.

Therefore, ConsRNN is reimplemented from scratch, reconstructing each computational block in Fortran: normalisation, state transition, state update, and damage output. A similar block-by-block strategy has been adopted in related work [29,31]. This route gives full control over both the model and its numerical evaluation inside the solver. The workflow proceeds in three stages. The models are first trained in TensorFlow, and the trained weights are exported to text files. The forward pass is then replicated in NumPy from these weight files and matched against the TensorFlow output. Finally, each block is rewritten in pure Fortran, with the tensor outputs validated against the NumPy reference at every step. The result is a self-contained Fortran implementation of ConsRNN that integrates directly into Abaqus FEM simulations.

The incremental update mechanism of RNNs, in which a state vector is carried as memory across time steps, makes them well suited to material modelling [9,14,36]. Compared with Transformers and CNNs, which can also represent deformation history, the recurrent update offers a practical advantage for this purpose by advancing the state incrementally without storing or growing a sequence. This matches the structure of an explicit finite-element solver exactly. At each increment, the RNN applies a transition function and then computes the output from the updated state, which is the same order of operations Abaqus performs at each material point.

The forward passes of the SimpleRNN- and ConsRNN-based architectures are summarised as pseudocode in Algorithms A1 and A2 (Appendix A), and the complete Python and Fortran implementations are openly available in the accompanying repository (see the Data Availability section). For both architectures, the Python and Fortran implementations were compared block by block on the same input data. This implementation equivalence check confirms that the Fortran routines reproduce the trained networks and is distinct

from the discretisation consistency examined in Section 4.2.3. The equivalence of the strain increments computed inside the solver against the offline reference is verified separately in Section 4.2.3, where the network is driven by the deformation history produced by the analysis rather than by prescribed input arrays.

At each sampling instant, set by the prescribed sampling frequency, the ConsRNN hidden-state vector is kept in the `StateEnv` variables of the subroutine, together with the strain increment since the previous sampling instant. The state vector is then updated in place with the transition function in Table 1, the damage is computed immediately, and both are stored in `StateEnv`. Unlike typical TensorFlow implementations, the subroutine does not assemble a growing input sequence of minor and major strain values and pass it to the network. ConsRNN therefore works as an incremental, memory-efficient solver that follows the strain evolution of the finite-element simulation increment by increment. This is the central objective of the present implementation.

4. Finite-Element-Based Numerical Experiments

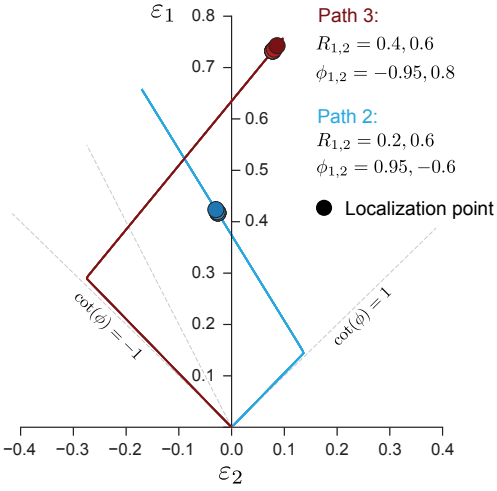
Reproducing a forward pass in Fortran is not the same as operating inside a solver, where the network is driven by solver-chosen increments at a resolution that varies between elements and between runs, along strain paths never seen during training. Deployment also exposes behaviour that offline testing cannot reveal, since the subroutine must carry its state through the solver's own memory management across thousands of material points without disrupting the analysis. The embedded surrogate is therefore evaluated in two stages:

1. prescribed bilinear paths in single-element simulations, where the increment count is varied while the path is held fixed (Section 4.1);
2. element-level damage predictions in a full structural simulation, where each element follows its own loading path (Section 4.2).

4.1. Single-Element Simulations

Two bilinear loading paths from [26], Paths 2 and 3, were run as single-element Abaqus/Explicit simulations with the ConsRNN-B256 VUMAT. In the remainder of Section 4, this deployed configuration is referred to simply as the ConsRNN. Each path is defined by the lengths and directions of its two linear segments in the same way as the bilinear paths of Section 2, with the values given in Figure 7. To test whether the localisation prediction depends on the increment size, each path was traversed with $N = 400$, 1000, 10,000, 50,000, 100,000, and 200,000 increments, which extends the offline test of Figure 4 to the embedded network. Figure 7 shows the two paths together with the ratio i/N, where i is the increment at which the predicted damage reaches unity.

No ground-truth localisation point is used in this comparison, so the reference for each path is the value to which i/N converges as N increases. The predicted localisation converges at about two thirds of Path 2 ($i/N \approx 0.65$) and near the end of Path 3 ($i/N \approx 0.97$). Relative to these values, the coarsest case, $N = 400$, deviates by 1.5% for Path 2 and 1.4% for Path 3, and the deviation decreases monotonically with refinement over the 500-fold range of increment counts. The $N = 400$ case is included as the coarsest test because the elements of the structural analysis accumulate thousands of increments. The embedded network thus behaves as an incremental solver whose state carries the accumulated damage between increments, so that the prediction is governed by the loading path rather than by its temporal discretisation, whether localisation occurs mid-path or near its end.



		Path 2		Path 3	
	N	i	i/N	i	i/N
Darkest	400	264	0.660	393	0.983
	1000	654	0.654	974	0.974
	10,000	6508	0.651	9696	0.970
	50,000	32,526	0.651	48,460	0.969
	100,000	65,048	0.650	96,915	0.969
Lightest	200,000	130,093	0.650	193,825	0.969

Figure 7. Localisation predictions of ConsRNN-B256 embedded in Abaqus/Explicit for the two bilinear strain paths in the $(\varepsilon_2, \varepsilon_1)$ principal strain plane, evaluated at increment counts from 400 to 200,000. Markers denote the localisation points predicted in the FE simulations, and the corresponding values of i and i/N are listed in the table. Path names follow [26].

4.2. Rupture of a Steel Plate

In the single-element simulations, the in-plane strains follow a prescribed path. The elements of a structure instead follow histories set by the deformation of the surrounding structure, which differ from element to element in shape and in the number of plastic increments. This section examines how the embedded ConsRNN behaves under such histories.

4.2.1. FE Model

The indentation of a bare steel plate by a rigid conical indenter was modelled in Abaqus/Explicit (Figure 8). The set-up follows the experiment of Alsos and Amdahl [37] on an unstiffened 1200 mm × 720 mm plate, which was indented until fracture. Their plate was 5 mm thick, whereas the plate modelled here is 3 mm thick, because this thickness was used to generate the bilinear forming-limit dataset of [15], on which the ConsRNN was trained. The plate was discretised with S4R four-node reduced-integration shell elements with three Gauss integration points through the thickness and an edge length of 15 mm. The resulting element length-to-thickness ratio of 5 equals the ratio of cell length to sheet thickness for which the training dataset of [15] was generated.

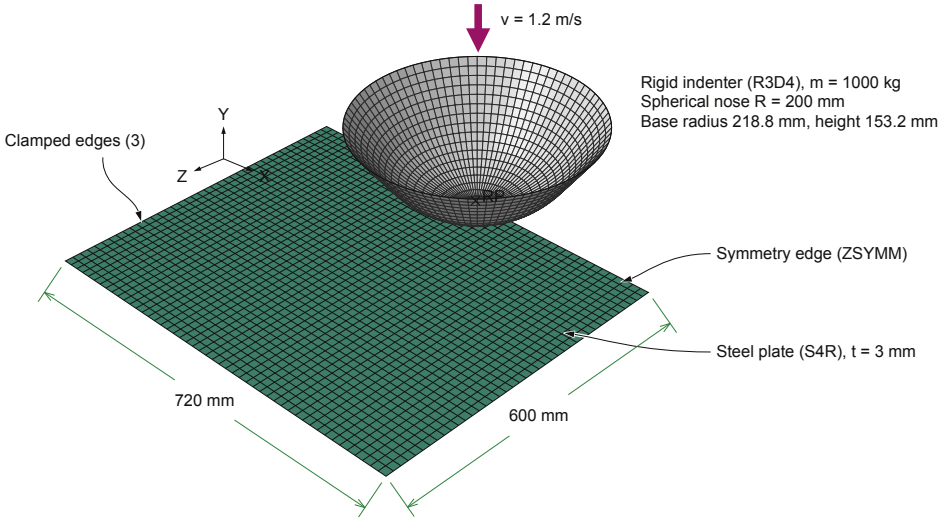


Figure 8. Structural FE model used in the numerical simulations.

Symmetry was exploited by modelling one half of the plate, 600 mm × 720 mm, with symmetry conditions on the cut edge and the remaining three edges fully clamped. The indenter was modelled as a discrete rigid body of R3D4 elements, governed by a reference node carrying a concentrated mass of 1000 kg. It has a spherical nose of 200 mm radius that blends into a cone, and it was driven at a constant velocity of 1.2 m/s over a step time of 0.2 s, giving a prescribed stroke of 240 mm. Contact was enforced through the general contact algorithm applied to all exterior surfaces, with a hard pressure-overclosure relation, a Coulomb friction coefficient of 0.3, and a contact damping coefficient of 0.95.

4.2.2. Material Model and Element Erosion

The constitutive behaviour was supplied through a user-defined material subroutine (VUMAT) that implements the structural steel of Section 2.1, Equation (1), under the plane-stress assumption appropriate for shell elements. The constitutive equations are integrated with a plane-stress-consistent return-mapping algorithm, in which the through-thickness strain component is recovered iteratively so that the zero normal-stress condition is enforced at every increment. The same subroutine evaluates the erosion criterion, and two criteria were used. The first is the ConsRNN, implemented as in Algorithm A2 (Appendix A). The second is the two-factor scaling (2FS) criterion, which depends on the stress triaxiality and the element length and was validated against experiments in [38,39]. The 2FS criterion accumulates damage through a linear rule, as the integral of the equivalent plastic strain increment normalised by the current triaxiality-dependent fracture strain. With either criterion, an element is deleted when its damage reaches unity.

4.2.3. Consistency at Different Temporal Resolutions

In an explicit structural analysis, all elements share one global time increment, set by the element stability limit. A fixed mass-scaling factor of 4 was applied at the start of the step, and the increment remained between 4.4 and 4.7 μs over the 0.2 s of loading, which took about 45,000 increments. The VUMAT is called at every integration point of the plate in every increment, but the network is advanced only on plastic increments (Section 2.3), so the number of network updates differs between integration points. A surrogate used inside a solver must therefore give predictions that are insensitive to this number.

This requirement was tested in three runs with the ConsRNN as the active erosion criterion, R1, R2, and R3, in which the network was advanced at sampling intervals (Section 3) of 10^{-4} s, 10^{-5} s, and every increment, respectively. Over the 0.2 s step, these intervals allow at most 2000, 20,000, and about 45,000 network updates, which lie within the range of increment counts tested in Section 4.1. In a fourth run, R4, element erosion was governed by the 2FS criterion evaluated at every increment, while the ConsRNN was evaluated passively at the three sampling intervals of R1 to R3 without affecting the solution. The three passive predictions of R4 are therefore driven by one and the same strain history.

Figure 9a shows the global force–displacement response of the four runs. The curves of R1, R2, and R3 coincide up to the peak force and differ little on the post-peak branch, although the sampling interval of the network differs by more than an order of magnitude between them. The fractured plate at frame 213, an indenter displacement of about 213 mm, is shown for R4 and R1 in Figure 9b,c. R1 is representative of all three ConsRNN runs, which gave the same fracture morphology. No element is deleted before the peak force, so the ConsRNN runs and R4 follow the same solution up to the peak, and only the fracture onset and the post-peak branch discriminate between the criteria (Figure 9a). Both criteria predict fracture onset at nearly the same indenter displacement and peak force, and the fracture paths at frame 213 are similar (Figure 9b,c).

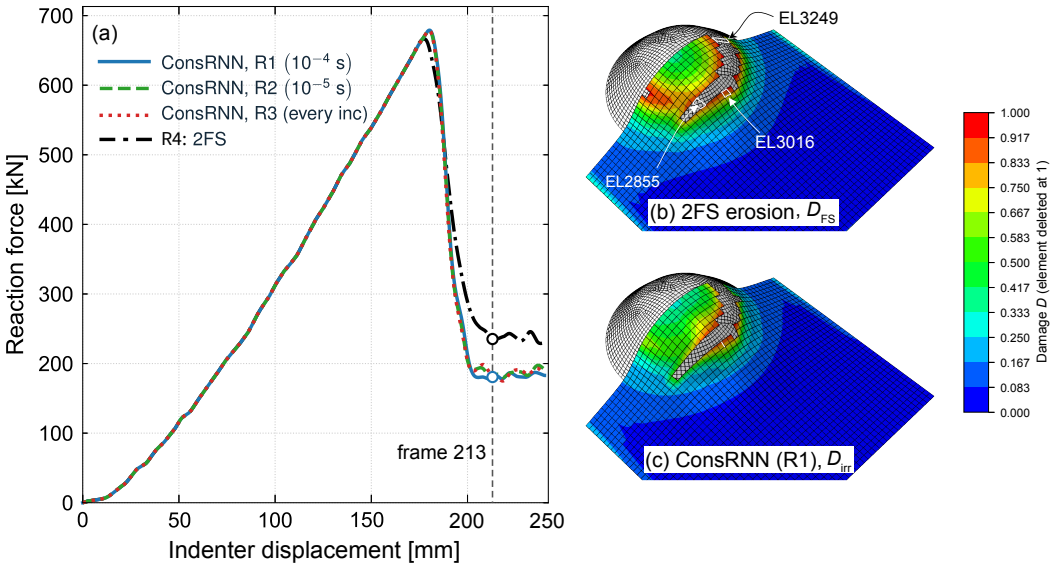


Figure 9. (**a**) Global force–displacement response of the four runs. Fractured plate at frame 213 obtained with (**b**) the 2FS criterion (R4), with contours of the 2FS damage, and (**c**) the ConsRNN criterion (R1), with contours of the running maximum D_{irr}.

To explain the similarities in f-d curve results, Figure 10 plots the principal strain histories of three elements marked in Figure 9b). Two of these elements were selected on the evolving crack path (3249 and 2855) and one neighbouring the crack (3016). Figure 10 shows an excellent overlap between the analysis with all three elements deforming along nearly proportional paths. Under proportional loading, empirical fracture criteria (2FS) are expected to provide good precision, as fracture strain is given as a function of the average stress state over the deformation history. The surrogate model, on the other hand, was developed explicitly to account for the history dependent nature of damage. Since the 2FS criterion used here was calibrated on the same steel from which surrogate training data were generated [15], it is only encouraging that both criteria give very similar results. However, limited variations in the deformation history conceal the expected advantage of a surrogate, a fracture prediction that accounts for changes in the loading path. We note in passing that although a structure with stronger path changes would separate the two criteria, this was not pursued because of lack of experimental validation data.

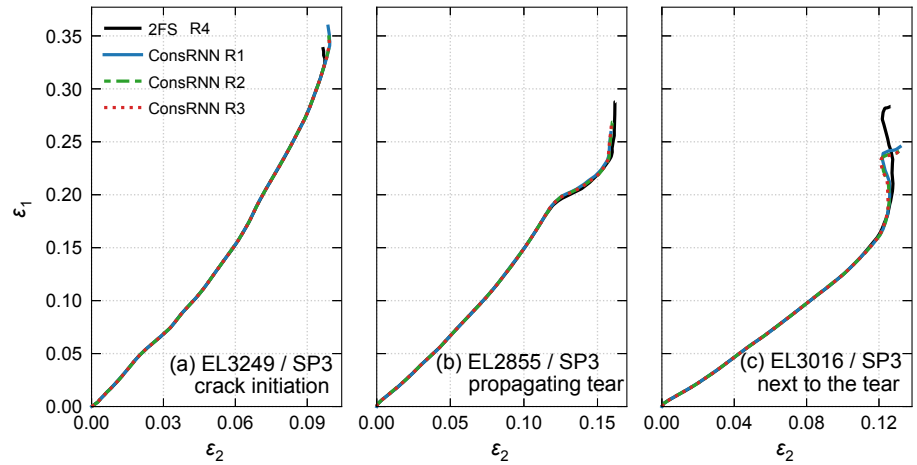


Figure 10. Principal strain histories at integration point SP3 (middle layer) of the three selected elements in the four runs. (**a**) Element 3249 at crack initiation, (**b**) element 2855 on the propagating tear, and (**c**) element 3016 next to the tear.

While consistency of the solution was already inferred by the excellent overlap of f-d curves (Figure 9a) and deformation histories (Figure 10) at different temporal resolutions, it is instructive to provide insight into the actual predicted variable (damage) during the deformation. For that purpose, Figure 11 shows the passive ConsRNN predictions for element 2855 in R4. The strain increments fed to the network differ by more than an order of magnitude between the coarsest and the finest sampling interval (Figure 11a), yet the three damage histories coincide (Figure 11b). The damage predicted by the ConsRNN is independent of the temporal resolution at which the solver supplies the strain history, which is precisely the property required for embedding in an explicit solver. The ConsRNN damage nevertheless decreases temporarily (Figure 11b), despite the increase in plastic strain. In the ConsRNN update, the strain increment sets only the size of each state advance, while its sign is set by $A_t - S_{t-1}$, so the predicted damage decreases whenever the transition activation falls below the accumulated state (Section 2.3); the running maximum D_{irr} removes this decrease from the reported damage.

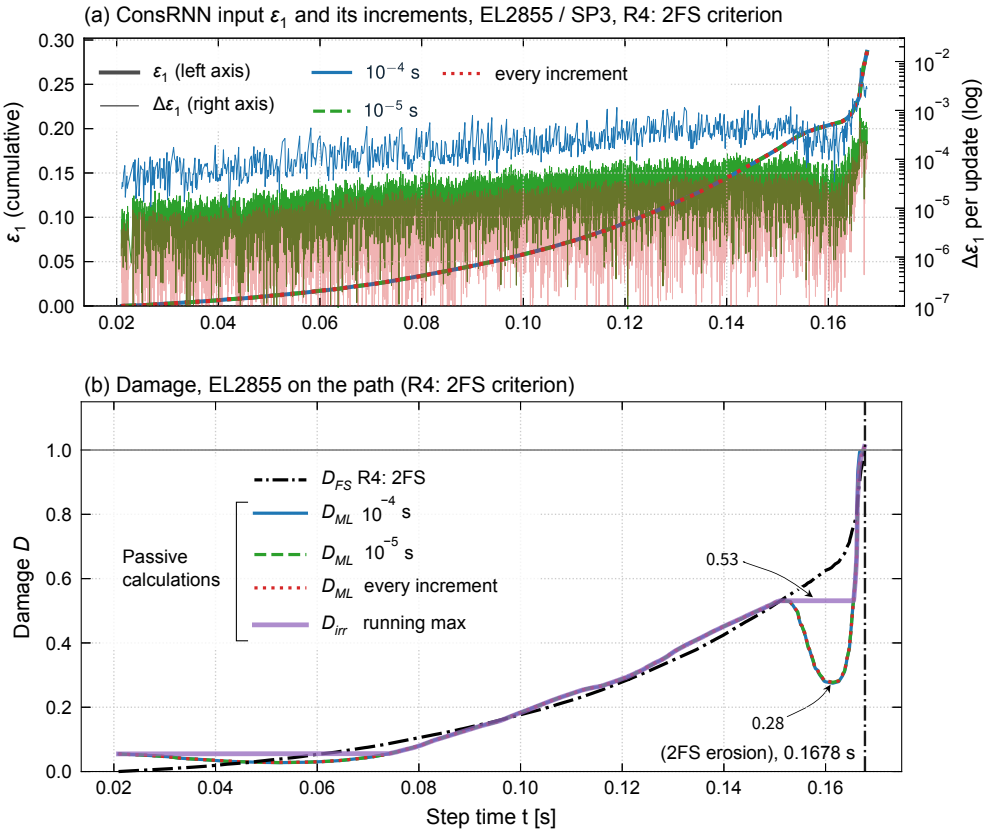


Figure 11. Detailed deformation insight of element 2855 through thickness integration point (SP3) in Run 4 with erosion governed by the 2FS criterion. (**a**) First principal strain fed to the ConsRNN and its increments at the three sampling intervals, up to erosion. (**b**) Damage at the same integration point, with the 2FS damage D_{FS}, the passive ConsRNN predictions D_{ML} at the three sampling intervals, and the running maximum D_{irr}. Passive means that the network was evaluated without governing erosion.

4.2.4. Computational Overhead

Run times were recorded for R4 and for one ConsRNN run on identical hardware with four domains and four processors over the same step time of 0.2 s. Both analyses completed in 45,164 increments, because the mass scaling, which is the same in both, sets the stable time increment independently of the erosion criterion. The 2FS analysis required 255 s of wall-clock time, against 250 s for the ConsRNN analysis. The two differ by 2% in

wall-clock time, and on the basis of a single run with each criterion, they are computationally indistinguishable at this scale.

5. Discussion

The plate simulation does not exercise the history dependence for which the surrogate was built because elements on the crack path deform along nearly proportional paths. Therefore, while the agreement in fracture onset indicates that the surrogate can operate as a fracture criterion in large-shell simulations, it does not demonstrate the expected advantage of the surrogate.

The choice of the ConsRNN over the SimpleRNN is a choice of consistency over accuracy at a fixed increment count. In the offline comparison at the 400-increment training resolution, the SimpleRNN gave a lower mean test MSE over five seeds than the deployed ConsRNN-B256 (Table 4) and located the localisation points of the multilinear paths more closely (Section 2.9). At other increment counts, its prediction shifted away from the ground truth, whereas the ConsRNN-B256 prediction converged as the number of increments increased (Figure 4) and stayed closer to the reference forming-limit points (Section 2.8). The comparison therefore demonstrates an accuracy–consistency trade-off between the two complete surrogate formulations. Because the networks also differ in input representation (Section 2.3), the comparison does not establish that the transition function alone explains their differences. Prediction accuracy is expected to improve with training paths that are more representative of the histories in structures, since the multilinear tests showed a loss of accuracy away from the bilinear training distribution. The consistency property makes such training practical, because the paths can be generated and learned at low increment counts while the trained model is applied at the much higher, solver-determined resolutions of the simulation.

The assessment is confined to simulation. The ConsRNN was trained on bilinear strain paths from finite element simulations and evaluated inside a finite element simulation. It has not yet been used to reproduce a fracture experiment, which is the necessary next step before the model can be regarded as predictive outside the simulation environment. The network is also valid for only one ratio of element length to sheet thickness, the ratio of 5 of the unit cell used to generate the data, and unlike the 2FS criterion, it does not take the element length as an input. Other ratios require the dataset to be regenerated and the network retrained. The predicted damage is not monotone, as the decrease for element 2855 shows. The running maximum D_{irr} removes the decrease from the reported damage without changing the deletion instant, but a formulation that is monotone by construction and preserves consistency remains to be developed.

Within these limits, the approach transfers. Consistency follows from the transition function, which embeds no physical quantity, and the VUMAT reads the trained parameters from separate weight files. Transfer to another metallic material therefore requires no change to the architecture or to the subroutine, provided that the retrained network keeps the present input dimension and number of hidden units. The workflow, from training on high-fidelity finite element data to deployment inside an established finite element solver through a user subroutine in which the forward pass is hard-coded (Section 3), can be reproduced from the openly available code and trained weights, and it can serve as a reference for integrating learned constitutive and damage models into physics-based solvers.

6. Conclusions

This study embedded a consistent recurrent neural network (ConsRNN) surrogate as a fracture criterion in Abaqus/Explicit shell simulations through a Fortran VUMAT subroutine, with the weights, hidden state, and incremental update handled inside the

subroutine rather than through an external coupling interface. The network was trained in Python/TensorFlow on 19,098 bilinear strain paths, simulated with solid elements in [15], that describe the onset of localisation in sheet metal. Inside the FE solver, its predictions were insensitive to the temporal resolution, both on prescribed bilinear paths in single-element simulations and at the integration points of a clamped plate indented to rupture. In the plate, the ConsRNN and the 2FS criterion, which was calibrated on the same steel and has been previously validated against experiments, predicted nearly the same fracture onset and peak force. While this demonstrates that ConsRNN criterion can be used as an operational fracture criterion inside structural scale shell element simulations, its physics-based knowledge about history dependence and how that translates into improved accuracy in fracture simulations with structural shell elements remains to be shown. The network is tied to one steel and to an element length-to-thickness ratio of 5. Retraining for another material needs no change to the architecture or the subroutine.

7. Patents

A US provisional patent application, No. 63/953,756, has been filed covering aspects of the methodology reported in this manuscript.

Author Contributions: M.A.Y.: conceptualisation, methodology, software, validation, formal analysis, investigation, visualisation, writing—original draft. M.K.: conceptualisation, methodology, supervision, funding acquisition, writing—review and editing. H.K.: methodology, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Estonian Research Council, grant number PSG754 (coupled simulation model for ship crashworthiness assessment).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The trained ConsRNN model weights, the standalone NumPy and Fortran implementations, the Abaqus/Explicit VUMAT deployment codes, and the evaluation scripts used to produce the figures in this paper are openly available on Zenodo at https://doi.org/10.5281/zenodo.21207967.

Acknowledgments: The corresponding author wishes to express sincere gratitude to Valeh Farzaliyev for valuable discussions and insightful comments. During the preparation of this manuscript, the authors used Claude Opus 5.5 (Anthropic) for selected rephrasing. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare a US provisional patent application, No. 63/953,756, relating to the methodology described in this manuscript. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A. Pseudocode of the Damage-Evolution Networks

Algorithms A1 and A2 summarise the forward passes of the baseline SimpleRNN and the proposed ConsRNN models as executed at each material point. The notation follows Table 1: W_x and W_s denote the input and recurrent weight matrices, W_{out} the output weight, and b the corresponding biases. The full NumPy and Fortran reference implementations, the trained model weights, and the Abaqus/Explicit VUMAT deployment codes are openly available in the repository at https://doi.org/10.5281/zenodo.21207967.

Algorithm A1 SimpleRNN damage evolution (baseline)

Require: Trained weights W_x, W_s, W_{out}, b; accumulated strain path $\{(\varepsilon_{1,t},\ \varepsilon_{2,t})\}_{t=1}^{T}$
Ensure: Damage predictions $\{D_t\}_{t=1}^{T}$

$\mathbf{S}_0 \leftarrow \mathbf{0}$ ▷ initialize recurrent state
for $t = 1$ **to** T **do**
  $\Delta\varepsilon_{1,t} \leftarrow \varepsilon_{1,t} - \varepsilon_{1,t-1}, \quad \Delta\varepsilon_{2,t} \leftarrow \varepsilon_{2,t} - \varepsilon_{2,t-1}$ ▷ strain increments
  $\phi_t \leftarrow \operatorname{arccot}\left(\Delta\varepsilon_{2,t}/\Delta\varepsilon_{1,t}\right)$ ▷ loading orientation
  $\mathbf{X}_t \leftarrow \left(\varepsilon_{1,t},\ \varepsilon_{2,t},\ \phi_t\right)$
  $\mathbf{S}_t \leftarrow \tanh\left(\mathbf{X}_t W_x + \mathbf{S}_{t-1} W_s + b\right)$ ▷ full state overwrite
  $p_t \leftarrow \sigma(\mathbf{S}_t W_{\text{out}} + b)$
  $D_t \leftarrow -\log(1 - p_t + \epsilon)$
end for

Algorithm A2 ConsRNN damage evolution

Require: Trained weights W_x, W_s, W_{out}, b; incremental strain path $\{(\Delta\varepsilon_{1,t},\ \Delta\varepsilon_{2,t})\}_{t=1}^{T}$
Ensure: Damage predictions $\{D_t\}_{t=1}^{T}$

$\mathbf{S}_0 \leftarrow \mathbf{0}$ ▷ initialize recurrent state
for $t = 1$ **to** T **do**
  $\mathbf{X}_t \leftarrow \left(\Delta\varepsilon_{1,t},\ \Delta\varepsilon_{2,t}\right)$
  $s_t \leftarrow \operatorname{clip}\left(\|\mathbf{X}_t\|_2,\ 10^{-15},\ 10^{2}\right)$ ▷ guarded norm
  $\hat{\mathbf{X}}_t \leftarrow \mathbf{X}_t / s_t$ ▷ unit direction of the increment
  $\mathbf{A}_t \leftarrow \tanh\left(\hat{\mathbf{X}}_t W_x + \mathbf{S}_{t-1} W_s + b\right)$
  $\mathbf{S}_t \leftarrow \mathbf{S}_{t-1} + (\mathbf{A}_t - \mathbf{S}_{t-1}) s_t$ ▷ consistency update
  $p_t \leftarrow \sigma(\mathbf{S}_t W_{\text{out}} + b)$
  $D_t \leftarrow -\log(1 - p_t + \epsilon)$
end for

References

1. Bonatti, C.; Berisha, B.; Mohr, D. From CP-FFT to CP-RNN: Recurrent neural network surrogate model of crystal plasticity. *Int. J. Plast.* **2022**, *158*, 103430. [CrossRef]
2. Khandelwal, S.; Basu, S.; Patra, A. A Machine Learning-based surrogate modeling framework for predicting the history-dependent deformation of dual phase microstructures. *Mater. Today Commun.* **2021**, *29*, 102914. [CrossRef]
3. Suttakul, P.; Vo, D.; Fongsamootr, T.; Wanison, R.; Mona, Y.; Katongtung, T.; Tippayawong, N.; Thawon, I. The role of machine learning for insight into the material behavior of lattices: A surrogate model based on data from finite element simulation. *Results Eng.* **2024**, *23*, 102547. [CrossRef]
4. Yarmohammadian, R.; Put, F.; Van Coile, R. Physics-Informed Surrogate Modelling in Fire Safety Engineering: A Systematic Review. *Appl. Sci.* **2025**, *15*, 8740. [CrossRef]
5. Mozaffar, M.; Bostanabad, R.; Chen, W.; Ehmann, K.; Cao, J.; Bessa, M. Deep learning predicts path-dependent plasticity. *Proc. Natl. Acad. Sci. USA* **2019**, *116*, 26414–26420. [CrossRef] [PubMed]
6. Wu, L.; Nguyen, V.D.; Kilingar, N.G.; Noels, L. A recurrent neural network-accelerated multi-scale model for elasto-plastic heterogeneous materials subjected to random cyclic and non-proportional loading paths. *Comput. Methods Appl. Mech. Eng.* **2020**, *369*, 113234. [CrossRef]
7. Ghavamian, F.; Simone, A. Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network. *Comput. Methods Appl. Mech. Eng.* **2019**, *357*, 112594. [CrossRef]
8. Maia, M.; Rocha, I.; Kerfriden, P.; van der Meer, F. Physically recurrent neural networks for path-dependent heterogeneous materials: Embedding constitutive models in a data-driven surrogate. *Comput. Methods Appl. Mech. Eng.* **2023**, *407*, 115934. [CrossRef]
9. Bonatti, C.; Mohr, D. On the importance of self-consistency in recurrent neural network models representing elasto-plastic solids. *J. Mech. Phys. Solids* **2022**, *158*, 104697. [CrossRef]
10. Flaschel, M.; Kumar, S.; De Lorenzis, L. Unsupervised discovery of interpretable hyperelastic constitutive laws. *Comput. Methods Appl. Mech. Eng.* **2021**, *381*, 113852. [CrossRef]

11. Carrara, P.; Ortiz, M.; De Lorenzis, L. Data-driven rate-dependent fracture mechanics. *J. Mech. Phys. Solids* **2021**, *155*, 104559. [CrossRef]
12. Kirchdoerfer, T.; Ortiz, M. Data-driven computational mechanics. *Comput. Methods Appl. Mech. Eng.* **2016**, *304*, 81–101. [CrossRef]
13. Eggersmann, R.; Kirchdoerfer, T.; Reese, S.; Stainier, L.; Ortiz, M. Model-free data-driven inelasticity. *Comput. Methods Appl. Mech. Eng.* **2019**, *350*, 81–99. [CrossRef]
14. Bonatti, C.; Mohr, D. One for all: Universal material model based on minimal state-space neural networks. *Sci. Adv.* **2021**, *7*, eabf3658. [CrossRef] [PubMed]
15. Yatkın, M.A.; Kõrgesaar, M. Machine learning enabled identification of sheet metal localization. *Int. J. Solids Struct.* **2024**, *288*, 112592. [CrossRef]
16. Palau, T.; Kuhn, A.; Nogales, S.; Böhm, H.J.; Rauh, A. A neural network based elasto-plasticity material model. In *Proceedings of the CD-ROM Proceedings of the 6th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2012)*; TU Wien: Vienna, Austria, 2012; p. 10.
17. Gorji, M.B.; Mohr, D. Towards neural network models for describing the large deformation behavior of sheet metal. *IOP Conf. Ser. Mater. Sci. Eng.* **2019**, *651*, 012102. [CrossRef]
18. Gorji, M.B.; Mozaffar, M.; Heidenreich, J.N.; Cao, J.; Mohr, D. On the potential of recurrent neural networks for modeling path dependent plasticity. *J. Mech. Phys. Solids* **2020**, *143*, 103972. [CrossRef]
19. Greve, L.; van de Weg, B.P. Surrogate modeling of parametrized finite element simulations with varying mesh topology using recurrent neural networks. *Array* **2022**, *14*, 100137. [CrossRef]
20. Yatkin, M.A.; Kõrgesaar, M.; Asan, V.M.; Romanoff, J.; Stuckner, J.; Kurban, H. Self-consistent recurrent neural network for path dependent deformation. *Sci. Rep.* **2026**, *16*, 20480. [CrossRef] [PubMed]
21. Heidenreich, J.N.; Mohr, D. Extended minimal state cells (EMSC): Self-consistent recurrent neural networks for rate- and temperature dependent plasticity. *Int. J. Plast.* **2025**, *188*, 104305. [CrossRef]
22. Girard, A.; Mohr, D. Recurrent neural network model predicting elasto-plasticity and matrix fracture in fiber-reinforced composites. *Int. J. Solids Struct.* **2026**, *326*, 113703. [CrossRef]
23. Wu, L.; Noels, L. Self-consistency Reinforced minimal Gated Recurrent Unit for surrogate modeling of history-dependent non-linear problems: Application to history-dependent homogenized response of heterogeneous materials. *Comput. Methods Appl. Mech. Eng.* **2024**, *424*, 116881. [CrossRef]
24. Yatkin, M.A.; Kõrgesaar, M. Fracture Estimation Based on Deformation History with Recurrent Neural Networks. In *2023 International Conference on Machine Learning and Applications (ICMLA)*; IEEE: New YOrk, NY, USA, 2023; pp. 1025–1030. [CrossRef]
25. Yatkin, M.A.; Korgesaar, M.; Romanoff, J.; Stuckner, J.; Işlak, Ü.; Kurban, H. Exploring various sequential learning methods for deformation history modeling. In *Proceedings of the International Conference on Engineering Applications of Neural Networks*; Springer: Cham, Switzerland, 2025; pp. 168–180.
26. Kõrgesaar, M.; Yatkin, M.A. Machine Learning Based Computational Models for Increased Accuracy and Enabling Digital Twins. In *Proceedings of the International Conference on Offshore Mechanics and Arctic Engineering*; American Society of Mechanical Engineers: New York, NY, USA, 2025; Volume 88964, p. V007T14A014.
27. Yatkın, M.A.; Kõrgesaar, M.; Işlak, Ü. A Topological Approach to Enhancing Consistency in Machine Learning via Recurrent Neural Networks. *Appl. Sci.* **2025**, *15*, 933. [CrossRef]
28. Bonatti, C.; Mohr, D. Data for: On the Importance of Self-Consistency in Recurrent Neural Network Models Representing Elasto-Plastic Solids. 2021. Available online: https://zenodo.org/records/4642208 (accessed on 25 October 2026).
29. Pantalé, O.; Tize Mha, P.; Tongne, A. Efficient implementation of non-linear flow law using neural network into the Abaqus Explicit FEM code. *Finite Elem. Anal. Des.* **2022**, *198*, 103647. [CrossRef]
30. Ott, J.; Pritchard, M.; Best, N.; Linstead, E.; Curcic, M.; Baldi, P. A Fortran-Keras Deep Learning Bridge for Scientific Computing. *arXiv* **2020**, arXiv:004.10652. [CrossRef]
31. Kessler, B.S.; El-Gizawy, A.S.; Smith, D.E. Incorporating Neural Network Material Models Within Finite Element Analysis for Rheological Behavior Prediction. *J. Press. Vessel Technol.* **2006**, *129*, 58–65. [CrossRef]
32. Jang, D.K.; Fazily, P.; Yoon, J.W. Machine learning-based constitutive model for J2-plasticity. *Int. J. Plast.* **2021**, *138*, 102919. [CrossRef]
33. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software. 2015. Available online: http://tensorflow.org/ (accessed on 25 October 2026).
34. Curcic, M. A parallel Fortran framework for neural networks and deep learning. *arXiv* **2019**, arXiv:1902.06714. [CrossRef]
35. Suh, H.S.; Kweon, C.; Lester, B.; Kramer, S.; Sun, W. A publicly available PyTorch-ABAQUS UMAT deep-learning framework for level-set plasticity. *Mech. Mater.* **2023**, *184*, 104682. [CrossRef]
36. Heidenreich, J.N.; Mohr, D. Recurrent neural network plasticity models: Unveiling their common core through multi-task learning. *Comput. Methods Appl. Mech. Eng.* **2024**, *426*, 116991. [CrossRef]

37. Alsos, H.S.; Amdahl, J. On the resistance to penetration of stiffened plates, Part I—Experiments. *Int. J. Impact Eng.* **2009**, *36*, 799–807. [CrossRef]
38. Kõrgesaar, M.; Romanoff, J.; Remes, H.; Palokangas, P. Experimental and numerical penetration response of laser-welded stiffened panels. *Int. J. Impact Eng.* **2018**, *114*, 78–92. [CrossRef]
39. Kõrgesaar, M. The effect of low stress triaxialities and deformation paths on ductile fracture simulations of large shell structures. *Mar. Struct.* **2019**, *63*, 45–64. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.