A Supervised Learning Framework for Arbitrary Lagrangian-Eulerian Simulations

Ming Jiang, Brian Gallagher, Joshua Kallman, Daniel Laney
Lawrence Livermore National Laboratory
Email: {jiang4,gallagher23,kallman2,laney1}@llnl.gov

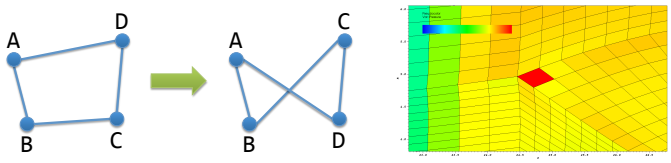
***Abstract*—The Arbitrary Lagrangian-Eulerian (ALE) method is used in a variety of engineering and scientific applications for enabling multi-physics simulations. Unfortunately, the ALE method can suffer from simulation failures that require users to adjust parameters iteratively in order to complete a simulation. In this paper, we present a supervised learning framework for predicting conditions leading to simulation failures. To our knowledge, this is the first time machine learning has been applied to ALE simulations. We propose a novel learning representation for mapping the ALE domain onto a supervised learning formulation. We analyze the predictability of these failures and evaluate our framework using well-known test problems.**

I. Introduction

As high-performance computing (HPC) becomes ever more powerful, scientists are able to perform highly complex simulations that couple multiple physics at different scales. The price that comes with this capability is that the simulations become much more difficult to use. One of the main culprits causing this difficulty are simulation failures. A common example can be found in hydrodynamics simulations that utilize the Arbitrary Lagrangian-Eulerian (ALE) method [1], [7], [23], which requires users to adjust a set of parameters to control mesh motion during the simulation. Finding the right combination of parameters to adjust in order to avoid the failures and complete the simulation is often a trial-and-error process that is a significant pain point for users.

In general, ALE consists of two phases: 1) an arbitrary number of Lagrangian steps followed by 2) an advective remap phase to relax the mesh. There are numerous parameters a user can adjust to determine how the mesh moves during the remap phase, ranging from purely Lagrangian (move mesh with fluid) to purely Eulerian (fix mesh in space) to some arbitrarily specified way. For a given problem, identifying which parameters to adjust, when and by how much is as much art as science. Often, finding the right ALE strategy requires many iterations (simulation fails, adjust parameters, and rollback) that can be disruptive and time consuming.

Currently, developing an ALE strategy is an ad-hoc process that relies mainly on the knowledge and experience of the user. There are approaches [10], [15], [3] for automatic remeshing or rezoning during the remap phase, but they are inadequate for multi-physics simulations, because they do not take into account the physics. Recent advances in numerical methods, such as using high-order curvilinear finite elements [5], [6], are more robust to simulation failures, but they still require user-driven parameter adjustments that can be disruptive.



(a) Mesh zone tangling (b) Anomalous hot spots

Fig. 1: Illustrations of ALE simulation failures.

For ALE simulations, there are two general ways that failures can occur: 1) mesh geometry becomes distorted or 2) physical quantities become distorted. A well-known example of mesh geometry distortion is *mesh zone tangling*, as illustrated in Figure 1(a), which corrupts the numerics around that tangled mesh zone and causes the simulation to halt. Simulations can also fail when physical quantities become distorted. One such example is *anomalous hot spots* (Figure 1(b)), which can occur when a physical quantity, such as energy density, spikes to unphysical levels within a mesh zone and causes the simulation to halt.

In this paper, we focus on predicting mesh tangling failures, which is a much more common problem plaguing current ALE codes. We propose a supervised learning framework for predicting conditions leading to these failures. Our framework uses the random forest learning algorithm and can be integrated into production quality ALE codes. In the following sections, we present a novel learning representation for ALE simulations, along with procedures for generating labeled data and training a predictive model. We present a set of experiments designed to analyze the predictability of failures in terms of the features, and evaluate the performance of our learning framework. Finally, we show how to incorporate the predictions into an actual relaxation strategy and demonstrate its usage on two well-known test problems.

II. Related Work

Machine learning has been applied to prediction [21], [16], [9] and avoidance [14], [11] of system faults in diverse problem domains, such as HPC hardware and industrial processes. However, these approaches are not readily applicable to simulation failures because: 1) faults are inherently stochastic, stemming from hardware, whereas simulation failures are deterministic and repeatable; 2) data sources differ (e.g., system logs vs. simulation state); and 3) avoiding faults generally means removing faulty hardware, whereas avoiding simulation failures involves parameter adjustments.

Fig. 2: A supervised learning framework for ALE simulations.

Currently, machine learning is applied to simulation data mainly for post-processing. In computational fluid dynamics, fluid features, such as vortices, are extracted automatically from simulation data using adaptive boosting [22], [2]. Machine learning has also been used to understand and mitigate certain kinds of model form uncertainty in Reynolds Averaged Navier Stokes (RANS) turbulence models [12], [13].

A common problem across scientific domains is the need for robust feature methods to encode high-dimensional data. In computational chemistry, machine learning is used to extract lower dimensional reaction coordinates from large simulation data [17], [19] and to learn the lower dimensional manifold that the higher dimensional molecular simulation data occupies [8]. In this work, we represent a complex simulation state with a small set of mesh quality metrics [20].

III. Methodology

In this paper, we present a machine learning framework for predicting conditions leading to ALE simulation failures. Our goal is to semi-automate the tedious development process for ALE strategies by exploiting *supervised learning*, which is well-suited to this problem since we can observe a running simulation and its end result. Figure 2 shows a high-level overview diagram of our proposed supervised learning framework. From an ALE simulation run, we collect labeled training data that can be used by a learning algorithm. We use the learning algorithm to derive statistical models for predicting conditions leading to simulation failures. For future simulation runs, we input these models into an inference algorithm that can be used to assist the development of ALE strategies.

A. Learning Representation

One of the main design choices for our framework is the *learning representation*, which maps the problem domain onto the building blocks of supervised learning: learning instances, features, and class labels. For ALE simulations, we have the current state-of-the-practice as a guide: how do users use ALE? When a simulation fails, the user asks *where* the failure occurred and *why*. In terms of *where*, users typically localize failures to a specific mesh zone. The *why* is typically expressed in terms of derived metrics using mesh quality and physical quantities for that zone. Users then adjust parameters based on these metrics to prevent similar failures in the future (e.g., relax zone if its metric is above a certain threshold).

Since mesh relaxation decisions can be made at the zone-level, it seems natural to represent a zone z as a learning instance i, the metrics calculated on z as features X of i, and the failure state of z as the class of i. Given that mesh zones evolve over the course of a simulation, we also need to represent the values of both zone metrics and failure states as they change over time. Therefore, we propose the following representation. Given a zone z at simulation time (or cycle) t:

1) a learning instance i is a pair (z,t)
2) features $X(i)$ are metrics of (z,t)
3) class of i is the failure state of z at cycle t

For mesh tangling failures, which are the focus of this paper, users typically use mesh quality metrics [20] to examine the failed zones. These metrics reveal different aspects of how *well shaped* a mesh zone is. Note that *poorly shaped* zones may not always indicate failures. For example, in coupled radiation transport and hydrodynamics simulations, long thin zones, which are typically considered to be poorly shaped, are necessary to capture the propagation of radiation through a medium. This is precisely what makes using ALE simulations so challenging. Table I lists the 16 metrics that we use for our experiments on 2D simulations with quadrilateral mesh zones.

Metric	Range	Metric	Range
Area	[0, MAX]	Scaled Jacobian	[-1, 1]
Aspect Ratio	[1, MAX]	Shape	[0, 1]
Condition	[1, MAX]	Shape and Size	[0, 1]
Distortion	[0, 1]	Shear	[0, 1]
Jacobian	[0, MAX]	Shear and Size	[0, 1]
Maximum Angle	[90, 360]	Skew	[0, 1]
Minimum Angle	[0, 90]	Stretch	[0, 1]
Oddy	[0, MAX]	Taper	[0, MAX]

TABLE I: Verdict mesh quality metrics used for features.

B. Generating Labeled Data

Unlike traditional machine learning applications where researchers have assembled databases of labeled data for training and testing, we do not have a database of ALE simulation failures. Assembling an initial set of failure examples associated with different ALE simulations is one of the contributions of this work. Our strategy for generating labeled data is to utilize well-known test problems used in developing ALE codes [5], [6]. Our goal is to generate a large set of diverse labeled data to train a predictive model.

Given a test problem, in order to generate many failure examples, we run an ALE code in the Lagrangian mode as much as possible. This ensures that we will encounter the most mesh tangling failures for that problem. Once we encounter a tangled mesh zone, we save its ID in a list of failed zones, rollback the simulation, and re-run. In the next run, we relax only zones on the failed zone list. This allows us to overcome all the previous tangled mesh zones, as well as ensure that other soon-to-fail zones will eventually achieve their tangling.

We identify tangled zones by checking side, corner and volume for negative values. Our *labeling* mechanism consists of

looping over all zones to check for these negative conditions. For each tangled zone, we output its metrics at the failure cycle to assemble our database of *positive class* examples.

For training data, we also need *negative class* examples, i.e., non-tangled zones. For these examples, in theory, we can use all the mesh zones from all the cycles prior to any tangling events. In practice, however, using all the zones from all the cycles will be computationally burdensome. For our experiments, we sub-sampled the cycles to generate the negative class examples (see Section IV for more details).

C. Predicting Failures

The learning instances, features, and class labels described above provide a basis for learning a *classifier* to discriminate failed zones from non-failed zones. However, a successful ALE strategy requires predicting failures before they occur so that mesh relaxation can be applied to avoid those failures. To predict failures, we must build a model that can identify the characteristics of soon-to-fail zones. Our approach is to train a *predictive* model using the *time history* of failed zones, by augmenting the positive class examples with metrics from those zones in the cycles prior to failure.

As for how to label these soon-to-fail zones, we want to assign values that can encode *time to failure* information for each failed zone. One option would be to use the number of cycles prior to failure as the label value; however, this option may not generalize well, due to the varying time scales of different multi-physics simulations. Instead, we designed a transition function $\phi(z_f,t)=1-\frac{1}{T}(t_f-t)$ to produce label values, such that T is the number of cycles in the time history and (z_f,t_f) is a failure instance. Essentially, ϕ maps the time interval $[t_f-T,t_f]$ to $[0,1]$ linearly to produce labels for soon-to-fail zones. In this fashion, ϕ can be viewed as an indicator for *risk of failure*, such that 0 means no risk and 1 means already failed. Now we can apply supervised learning to solve a regression problem where the response variable is ϕ and the regressors are the zone metrics.

D. Learning Algorithm

The learning framework described above is flexible enough to incorporate any supervised learning algorithm that supports regression. That said, there are practical reasons to prefer one algorithm over another. In particular, since the focus of this work is on formulating the ALE application as a learning problem, we want a learner that "just works" and does not require a lot of engineering of the algorithm itself. For this reason, we chose the random forest [4] algorithm.

A random forest is an ensemble approach, which aggregates the predictions of numerous "weak" decision-tree learners to produce a single "strong learner." Random forests provide an expressive representation, capable of modeling complex (i.e., non-linear) relationships between variables of different types (e.g., continuous and categorical). They are forgiving and robust to high-dimensional feature spaces, large training sets, and redundancy (i.e., correlations) in both features and training examples. Most importantly, random forests tend to perform well "out of the box" without requiring tuning of hyper-parameters or manual specification of model structure.

E. ALE Code Integration

In order to evaluate our proposed supervised learning framework for predicting ALE simulation failures, we need to integrate the learning and inference algorithms into an ALE code. We chose to use KULL [18], which is an ALE code for modeling high energy density physics, due to its software maturity and production usage. Since KULL provides a python interface, we decided to use the scikit-learn package for the random forest algorithm. KULL also links to the Verdict library [20], which we used to compute the mesh quality metrics. The software library that we developed for this integration is called LAGER (Learning Algorithm-Generated Empirical Relaxer).

One way to demonstrate the utility of our framework is to use the predicted value ϕ as part of a relaxation strategy. Mesh relaxation in KULL occurs on a node by node basis. In order to incorporate ϕ, we need to make the transition from the zone-based predictions of when failures occur to the node-based decisions of how much to relax. To do this, we developed a simple method that averages the predictions from the adjacent zones of a given node. Specifically, we assign a value $\Phi(n,t)=\frac{1}{N}\sum_{a=1}^{N}\phi(z_a,t)$ to each node n at cycle t, where N is the number of adjacent zones (z_a) of node n.

We take advantage of a functionality built into KULL that performs the Eulerian relaxation step for every node at every cycle. Thus KULL calculates a relaxation vector $\bar{v}$ for each node, which we can scale by its associated Φ value. Hence the relaxation for that node is scaled according to the predicted risk of tangling in its adjacent zones. In this way, LAGER can dynamically and smoothly control the amount of relaxation taking place at any node in the mesh during each cycle.

IV. Experimental Results

For our experimental results, we use two well-known test problems, implemented in KULL, that have been used for developing ALE codes. These problems are described below.

1) Bubble Shock: Models a planar shock traveling through a spherical helium bubble in air from the right side of the geometry. The geometry is idealized and defined on a cylindrically symmetric mesh in RZ coordinates.

2) Shock Tube: Models a planar shock traveling through an air/SF6 interface and generating a Richtmeyer-Meshkov instability. It is simulated on a 3D mesh "slab" that is only one zone thick in the Z dimension.

A. Framework Evaluation

In Section III, we presented various design choices for the supervised learning framework. To better understand the implications of these design choices, we seek to answer the following questions: Are the features (in III-A) useful for distinguishing failures from non-failures? How effective is our procedure (in III-B) for generating labeled data? Is time history of failures (in III-C) suitable for predicting failures?

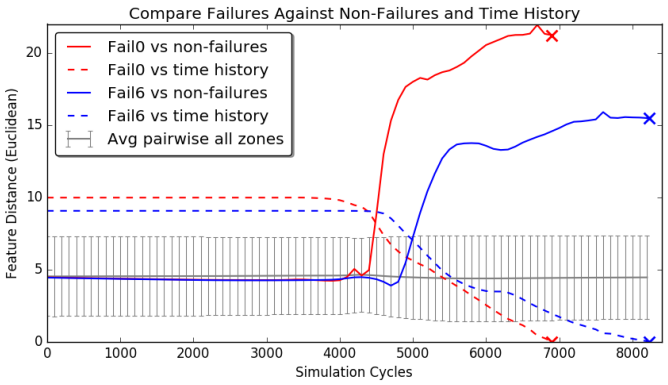
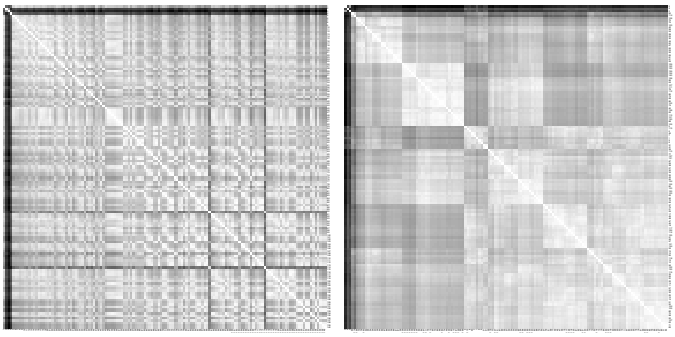


Fig. 3: Feature distance for failure analysis and comparison.



(a) Natual ordering of failures (b) Cluster re-ordering of failures

Fig. 4: Similarity of failures using pairwise feature distance.

To answer these questions, we conduct the following experiments to compare failures against 1) non-failures, 2) other failures, and 3) their time histories. We use the Bubble Shock test problem, for which our labeling procedure produced 140 unique failures. For the comparisons, we use a feature distance measure based on the the mesh quality metrics presented in Table I. For the distance measure, we first standardize features such that each has mean (μ) of 0 and standard deviation (σ) of 1 across all instances: $\hat{X} = (X - \mu)/\sigma$. To compute the standardized feature distance between two instances i and j, we use Euclidean distance: $\delta(i,j) = \|\hat{X}(i) - \hat{X}(j)\|^2$.

1) Non-Failures Comparison: For this experiment, we want to determine if failures are distinguishable from non-failures using feature distance. To facilitate this comparison, we define the average feature distance between a zone z and all other zones at cycle t as: $\Delta(z,t) = \frac{1}{M}\sum_{m=1}^{M}\delta((z,t),(z_m,t))$, where M is the number of mesh zones. We compute $\Delta(z,t)$ for every failed zone z_f and for every cycle t in $[0,t_f]$.

Figure 3 shows the $\Delta(z,t)$ curves for two representative failed zones as solid lines: fail0 (red) and fail6 (blue). For a baseline comparison, we plot the averaged $\Delta(z,t)$ curve (gray) for all 11,560 mesh zones, along with error bars to represent its σ. From cycles 0 to 4000, both fail0 and fail6 are near the averaged $\Delta(z,t)$ curve. After cycle 4000, both curves drastically increase in their feature distance, due to the deformation caused by the impinging shock wave. These results show that using our chosen features, failed zones exhibit characteristics that clearly distinguishes them from non-failed zones well before the failure occurs.

2) Other Failures Comparison: For this experiment, we want to determine if our labeling procedure can generate a diverse set of failures (i.e., failures with feature diversity). For this comparison, we compute the feature distance $\delta(i_f,j_f)$ between all pairs of failures. We plot the resulting distance matrix using a grayscale colormap, with 0 as white and 16.8 as black. Figure 4(a) shows the distance matrix in the natural order in which the failures were generated, and Figure 4(b) shows a re-ordering using hierarchical clustering (hclust function in R) to reveal block structures indicating similarities.

These results show that there are groups of similar failures, which is evident even in the natural ordering. More importantly, they show that the distance between some failures can be quite large, well-above the range of $[2.56, 4.62]$ for the averaged $\Delta(z,t)$ curve and its σ shown in Figure 3. This indicates that our labeling procedure can generate failures with feature diversity, even from a single test problem.

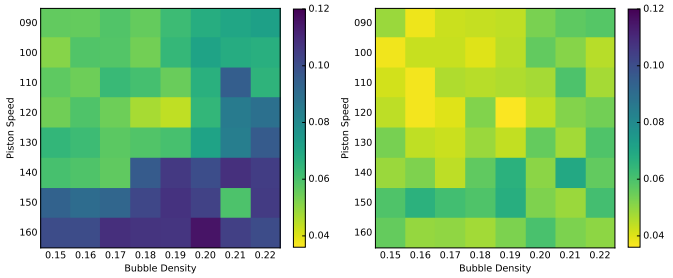
3) Time History Comparison: For this experiment, we want to determine if time history of failures can be used to predict risk of failure. Given that some of the features can change drastically during failure, we need to ascertain if the ϕ values assigned for soon-to-fail zones are suitable. For this comparsion, we compute the feature distance $\delta((z_f,t),(z_f,t_f))$ for every failed zone z_f and for every cycle t in $[0,t_f]$.

Figure 3 shows the time history curves for fail0 (red) and fail6 (blue) as dashed lines. At t_f, the distance is obviously 0. As we move away from t_f, the distance grows and reaches a plateau during the pre-shock phase of the simulation. These results show that the time history curves do not exhibit noticeable random fluctations, which would make predictions more difficult; in fact, the curve segments preceding t_f appear almost linear, which our ϕ function is designed to capture.

B. Learning Evaluation

Once we have generated labeled data, we run the random forest algorithm to train a model for predicting risk of failure. To better understand the performance of our supervised learning framework, we seek to answer the following questions: How well does the learnt model generalize to new simulation variations? What does the learning algorithm tell us about the features? How effective is our simple relaxation strategy at utilizing the predictions? To answer these questions, we run the following experiments using LAGER, which uses the RandomForestRegressor function in scikit-learn with the number of estimators (trees) set to 100.

For the learning experiments, we construct both training and testing sets by including all failures observed during the simulation, with time history (T) set to 100 cycles, and a temporal sample of non-failures (every 1000 cycles up to the first failure cycle). The reasons for this approach are: 1) it can cleanly separate failures from non-failures, 2) it provides uniform coverage of both classes throughout the simulation, and 3) it is a reasonable approximation to the set of relaxation decisions that would be encountered in an actual simulation



(a) Train on initial simulation (b) Train on 64 variations

Fig. 5: RMSE of predicting ϕ values for simulation variations.

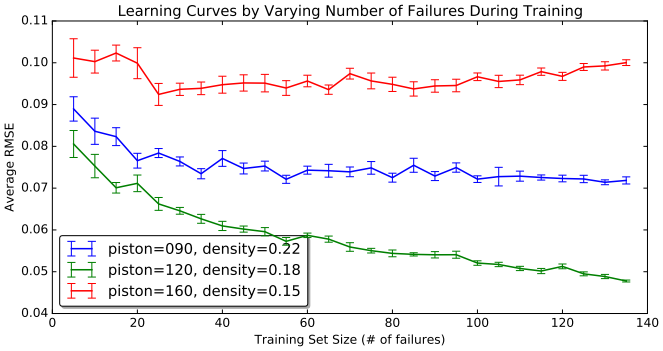


Fig. 6: Analyze model performance by varying training data.

run. We evaluate each experiment by measuring the root-mean-square error (RMSE) on the task of predicting the ϕ values.

1) Simulation Variations: Once we have a trained random forest model, we can evaluate it using data from other simulations. A simple way to produce simulations with similar physics is to vary the initial conditions of the original simulation. For Bubble Shock, we can vary the piston speed (shock strength) and the bubble density. We create a set of 8×8 simulations by varying piston speed (124.824) from 90–160 and bubble density (0.182) from 0.15–0.22.

We evaluate model performance using these 64 simulations. Figure 5(a) shows the resulting 8×8 RMSE matrix using the *viridis* colormap. We observe that: 1) there is significant variance in performance on the 64 test simulations and 2) performance generally decreases as distance between training and test simulations increases (i.e., as we move away from the center of the colormap). This indicates that *variance* is a significant source of error (i.e., we are overfitting the training data) and we would benefit from training on more diverse data.

To better understand how training data affects performance, we vary the number of failure examples (F) available at training time, from 5 to 135. The learning curves in Figure 6 show the mean RMSE over 10 trials for each F value and test set (error bars represent standard error). As F increases, we observe an initial decrease in RMSE. When train and test simulations are similar (e.g., piston=120/density=0.18) performance continues to improve with more data. When training and test simulations differ (e.g., piston=160), performance tops-out quickly and may even decrease with more data. This is consistent with the high RMSE we see in Figure 5(b) for piston=160 and again indicates important differences between data from individual simulation variations.

Figures 5(a) and 6 suggest that generalization performance may improve by training across multiple simulation variations. We evaluate this hypothesis by training on data from 63 of 64 simulations and testing on the one simulation not used for training. Repeating this *leave-one-out* cross-validation procedure for each of the 64 test simulations, we obtain the result in Figure 5(b). Note that Figures 5(a) and 5(b) use identical test sets and differ only in the amount and source of training data, which do not include any test data. These results show that training on data from a variety of simulations leads to improved and more consistent performance across simulations (RMSE mean: $0.08\rightarrow0.05$ and stdev: $0.02\rightarrow0.01$).

2) Feature Importance: Of the 16 mesh quality metrics chosen as features, we ask which are most informative (or important), since this knowledge will help ALE users design more effective relaxation strategies. One way to measure this is by using the *gini* importance, which is calculated during the training of the random forest. It is based on how often a feature is chosen for a split and how well that split separates the data into classes. Table II lists the 8 most important features calculated from the train-on-64-variations test, which produced 64 gini measures that we used to calculate the mean and stdev of the importance score per feature. For this test, the most informative features are: Scaled Jacobian and Maximum Angle. One thing to note about gini importance is that it can vary depending on the training setup.

Metric	Importance	Metric	Importance
Scaled Jacobian	$0.43\pm4.9e\text{-}2$	Aspect Ratio	$0.03\pm2.6e\text{-}4$
Maximum Angle	$0.35\pm4.9e\text{-}2$	Area	$0.02\pm2.6e\text{-}4$
Distortion	$0.06\pm8.0e\text{-}4$	Stretch	$0.02\pm2.9e\text{-}4$
Minimum Angle	$0.04\pm4.0e\text{-}4$	Taper	$0.02\pm1.8e\text{-}4$

TABLE II: Feature importance for Bubble Shock using gini.

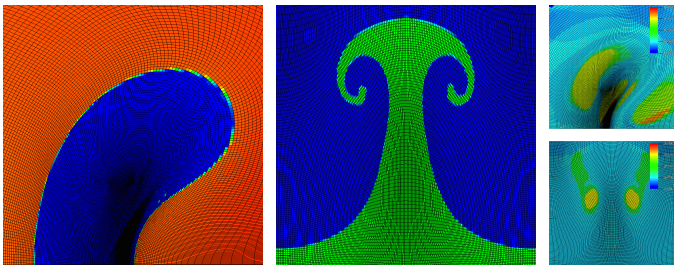
3) Relaxation Strategy: Once we have a learnt model, we can use it in similar simulations to predict risk of failure. One way to use these predictions is in a relaxation strategy as we described in Section III-E for LAGER. To evaluate that relaxation strategy, we trained models for both Bubble Shock and Shock Tube, and then applied those models to run those test problems using only LAGER and no other ALE parameters. Table III shows the comparisons between an ALE user developed strategy and the LAGER strategy, in terms of the minimum time step (ΔT) and the number of cycles required to reach goal time. For both problems, we were able to run the simulation to completion (goal time) without encountering any mesh tangling failures. Also, the simulation time step (ΔT) was kept appropriately large, allowing LAGER to reach goal time in fewer cycles than the ALE user strategy. Figure 7 shows the visualization of both simulations using the LAGER strategy, as well as the predicted ϕ values.

V. Conclusion

In this paper, we presented a supervised learning framework for predicting conditions leading to ALE simulation failures.

Test Problem	Approach	Minimum ΔT	# of Cycles
Bubble Shock	ALE User	$O(10^{-2})$ shakes	~252,000
	LAGER	$O(10^{-1})$ shakes	~71,000
Shock Tube	ALE User	$O(10^{-9})$ shakes	~58,000
	LAGER	$O(10^{-9})$ shakes	~36,000

TABLE III: Relaxation strategies from LAGER and ALE user.



(a) Bubble Shock LAGER (b) Shock Tube LAGER (c) ϕ values

Fig. 7: LAGER results on Bubble Shock and Shock Tube.

We presented a novel learning representation for ALE simulations, along with procedures for generating labeled data and training a predictive model. We integrated this framework into an ALE code for testing and evaluation. We designed a set of experiments to analyze the predictability of failures using feature distance, and to evaluate the performance of our learning framework. Finally, we showed how to incorporate the predictions into an actual relaxation strategy and demonstrated its usage for two well-known test problems.

Our results suggest a number directions for further research. First, Figure 3 shows that zone features provide quite a bit of early warning leading up to failure. This indicates that we may be able to improve predictive performance by optimizing our use of failure time history. Key questions include: 1) How much time history to consider? and 2) Which transition function to use (i.e., how quickly to discount historical data)?

Our approach to generating class labels and choosing training instances is conservative in the sense that the *negative* zones are drawn from a simulation time well before any failures occur. However, we may be able to fit a tighter decision boundary, and therefore improve performance, by training on non-failed zones that are spatio-temporal neighbors of failed zones. The difficulty is that we cannot know which zones may have failed had their neighbors not been relaxed. Questions here include: 1) Can we reduce noise in the *negative* labels (e.g., by clustering instances in feature space)? 2) Does including *negative* instances from failure periods in the simulation improve performance?

Finally, this work focused on guiding ALE simulations to completion. However, it is equally important that the simulated physics is accurate. Unfortunately, this is much harder to judge. The core question is: Given two ALE strategies that that produce different solutions, which is better? Among the approaches we plan to investigate are: 1) tracking total mesh relaxation (intuitively less is better) and 2) evaluating physics metrics as indicators for physical (in)accuracy.

ACKNOWLEDGMENT

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-698953), with funding provided by LDRD 16-ERD-036.

REFERENCES

[1] D. Benson, "Computational Methods in Lagrangian and Eulerian Hydrocodes," *Computer Methods in Applied Mechanics and Engineering*, vol. 99, no. 2-3, pp. 235–394, 1992.

[2] A. Biswas, W. He, Q. Deng, C.-M. Chen, H.-W. Shen, R. Machiraju, and A. Rangarajan, "An uncertainty-driven approach to vortex analysis using oracle consensus and spatial proximity," in *IEEE Pacific Vis*, 2015.

[3] W. Bo and M. Shashkov, "Adaptive reconnection-based arbitrary lagrangian eulerian method," *J. Comput. Phys.*, vol. 299, no. C, 2015.

[4] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, 2001.

[5] V. Dobrev, T. Ellis, T. Kolev, and R. Rieben, "Curvilinear Finite Elements for Lagrangian Hydrodynamics," *International Journal for Numerical Methods in Fluids*, vol. 65, no. 11-12, 2011.

[6] V. Dobrev, T. Kolev, and R. Rieben, "High-Order Curvilinear Finite Element Methods for Lagrangian Hydrodynamics," *SIAM Journal on Scientific Computing*, vol. 34, 2012.

[7] J. Donea, A. Huerta, J.-P. Ponthot, and A. Rodríguez-Ferran, "Arbitrary Lagrangian-Eulerian Methods," in *Encyclopedia of Computational Mechanics*. John Wiley & Sons, 2004.

[8] A. L. Ferguson, A. Z. Panagiotopoulos, I. G. Kevrekidis, and P. G. Debenedetti, "Nonlinear dimensionality reduction in molecular simulation: The diffusion map approach," *Chemical Physics Letters*, vol. 509, no. 13, pp. 1 – 11, 2011.

[9] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into hpc systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 77:1–77:11.

[10] P. Knupp, L. Margolin, and M. Shashkov, "Reference Jacobian Optimization-Based Rezone Strategies for Arbitrary Lagrangian Eulerian Methods," *Journal of Computational Physics*, vol. 176, 2002.

[11] Z. Lan and Y. Li, "Adaptive Fault Management of Parallel Applications for High-Performance Computing," *IEEE Transactions on Computers*, vol. 57, pp. 1647–1660, 2008.

[12] J. Ling, "Using machine learning to understand and mitigate model form uncertainty in turbulence models," in *IEEE International Conference on Machine Learning and Applications*, 2015, pp. 813–818.

[13] J. Ling and J. Templeton, "Evaluation of machine learning algorithms for prediction of regions of high reynolds averaged navier stokes uncertainty," *Physics of Fluids*, vol. 27, no. 085103, 2015.

[14] L. Lopes and L. Camarinha-Matos, "Learning Failure Recovery Knowledge for Mechanical Assembly," in *IEEE/RJS IROS Intelligent Robots and Systems*, 1996, pp. 712–719.

[15] R. Loubère, P.-H. Maire, M. Shashkov, J. Breil, and S. Galera, "ReALE: A Reconnection-based Arbitrary Lagrangian-Eulerian Method," *Journal of Computational Physics*, vol. 229, 2010.

[16] J. Murray, G. Hughes, and K. Kreutz-Delgado, "Machine Learning Methods for Predicting Failures in Hard Drives: A Multiple-Instance Application," *Journal of Machine Learning Research*, vol. 6, 2005.

[17] B. Peters, G. T. Beckham, and B. L. Trout, "Extensions to the likelihood maximization approach for finding reaction coordinates," *The Journal of Chemical Physics*, vol. 127, no. 3, 2007.

[18] J. Rathkopf *et al.*, "KULL: LLNL's ASCI Inertial Confinement Fusion Simulation Code," in *PHYSOR 2000 Advances in Reactor Physics and Mathematics and Computation into the Next Millennium*, 2000.

[19] E. E. Santiso and B. L. Trout, "A general set of order parameters for molecular crystals," *J. of Chemical Physics*, vol. 134, no. 6, 2011.

[20] C. Stimpson, C. Ernst, P. Knupp, P. Pébay, and D. Thompson, "The Verdict Geometric Quality Library," Sandia National Laboratories, Tech. Rep. SAND2007-1751, 2007.

[21] G. Xiang and J. Nan, "Supervised Methods for Fault Detection in Vehicles," Master's thesis, Halmstad University, 2010.

[22] L. Zhang, Q. Deng, R. Machiraju, A. Rangarajan, D. Thompson, D. K. Walters, and H.-W. Shen, "Boosting techniques for physics-based vortex detection," *Computer Graphics Forum*, vol. 33, no. 1, pp. 282–293, 2014.

[23] J. Zukas, *Introduction to Hydrocodes*. Elsevier, 2004.