

# Machine Learning Tools for Blast Load Prediction in Obstructed Environments

Adam A. Dennis MENG

*Thesis submitted to*



Department of Civil and Structural Engineering

*for the degree of*

Doctor of Philosophy

January 2024

# Abstract

The assessment of human injuries and structural damage following the detonation of a high explosive requires a comprehensive understanding of the blast load parameters. As explosive events are inherently unpredictable, and key details associated to the charge size, shape, location and material cannot be known a priori, obtaining these parameters often requires probabilistic approaches that feature large batches of numerical models with varying input conditions to embrace this uncertainty.

Machine learning (ML) methods have been shown to rapidly provide accurate predictions for many complex multi-parameter problems in a range of disciplines, including applications featuring blast wave coalescence. However, since ML tools develop their predictive accuracy through a training process that requires data from the problem being modelled, a dependency on potentially costly numerical solvers or physical experiments remains. Furthermore, ML tools are often provided with inputs relating to domain-specific parameters, preventing them from being used beyond the initial problem set, reducing their generality, and thus, requiring the tools to be re-trained when a new scenario is generated.

This thesis introduces two novel methods that independently reduce the impact of costly data collection processes, and prevent the development of tools with limited potential uses. Firstly, when a batch of numerical models are required for probabilistic assessments or training ML tools, any given model can share a number of solution steps with the others. Hence, simulating all domains from birth to termination may result in large amounts of calculation repetition that needlessly increases the overall computation time. The Branching Algorithm (BA) is therefore introduced as a means of mapping data between domains to ensure that calculation steps are only computed once, by identifying when the parameter fields of each model in the batch becomes unique.

Following this, a Direction-encoded Framework for ML tools is developed to enable predictions of blast loading parameters that are based on the surroundings of each point of interest and its position relative to the charge. Through comparisons to a traditional Artificial Neural Network (ANN), provided with global domain inputs, the framework is applied as the Direction-encoded Neural Network (DeNN) to show that the adapted approach enables predictions to be generated in domains with variable sizes and movable obstacles without requiring additional task-specific training.

The computational benefits of BA and the DeNN are then leveraged in a combined analysis, whereby a dataset is incrementally generated using a numerical solver and the BA, and simultaneously used to train the DeNN until a prescribed performance threshold is met. The DeNN then replaces the solver to generate results for any remaining scenarios being evaluated to further reduce the computation time without a detrimental loss of accuracy.

By reducing the time required to conduct a batch analysis, and developing versatile, robust ML based tools, this thesis has shown that complex obstructed environments can be rapidly modelled with consideration of varied geometrical and charge conditions. Results presented throughout the study therefore contribute to the goal of being able to effectively assess the risk posed by a given threat in a probabilistic manner.

# Acknowledgements

My journey to completing this thesis never felt like a solo expedition thanks to the support and guidance given by so many colleagues that I now call friends. As I write this section it's hard not to think back to all the times when people have helped me because they wanted to, not because they had to.

Firstly, I would like to thank my supervisor, Dr. Sam Rigby. Ever since the first year of my undergraduate degree, when I joined you on a trip out to the blast site, you've supported me and made sure that I continually challenged myself. Thank you for letting me have space to grow as a researcher and for always believing in me.

To Dr. Chris Stirling, thank you for your unofficial supervision and guidance throughout this project. I simply wouldn't have been able to explore all of my ideas without your help. I'm excited to see where you and the team can take Viper in the years to come.

To Dr. Danny Smyl, Prof. Sam Clarke, Dr. Matteo Di Benedetti, Dr. Harry Day, Jonathan Wood, and all the academics in the Blast group, thank for your advice and willingness to help with my professional development. The confidence that I've gained through the teaching opportunities and feedback that you've given me will stick with me no matter which career path I follow.

To Adrien, Arthur, Charles, Dain, Ella, Emma, Isuru, Jay, Jordan, Leanna, Lewis, Lucy, Oswald, Rowena, Ross, Tommy, and Tom, some of my favourite memories come from the chats we've had in various labs and lunch breaks, over table football games and desk dividers. Thank you for continually motivating me to produce my best work and for making the experience so enjoyable.

To Holly and Rose, thank you for always being happy to help with any of my questions and for helping with the admin of my project. Similarly, thank you to the building staff that provided a comfortable space for me to work in over the last 3 years.

Finally, Mum, Dad, Charlie, Sophie, you're the reason I want to be the best possible version of myself. Thank you for everything.

Adam A Dennis,  
Wednesday 3<sup>rd</sup> January, 2024

# Contents

Abstract . . . . .	i
Acknowledgements . . . . .	ii
List of Figures . . . . .	xiv
List of Tables . . . . .	xvii
Nomenclature . . . . .	xviii
<b>1 Introduction</b>	<b>2</b>
1.1 Background and motivation . . . . .	2
1.2 Aims and objectives . . . . .	4
1.3 Thesis outline . . . . .	5
1.4 Published work . . . . .	6
<b>2 Theory background and literature review</b>	<b>7</b>
2.1 Introduction . . . . .	7
2.2 Blast wave mechanics . . . . .	7
2.2.1 Shock wave formation and detonation in free air . . . . .	7
2.2.2 Pressure-Time histories . . . . .	8
2.2.3 TNT equivalence . . . . .	9
2.2.4 Blast scaling . . . . .	9
2.2.5 Blast wave reflection . . . . .	10
2.2.6 Modelling methods . . . . .	11
2.2.7 Explosions in confined environments . . . . .	13
2.2.8 Explosions in obstructed environments . . . . .	14
2.2.9 Computational modelling approaches . . . . .	16
2.3 Fast Running Engineering Models (FREMs) . . . . .	18
2.3.1 Surrogate and Reduced Order Modelling . . . . .	18
2.3.2 FREMs in Blast . . . . .	19
2.4 An introduction to Machine Learning . . . . .	21
2.4.1 Introduction . . . . .	21

2.4.2	Applications . . . . .	22
2.4.3	Development approaches . . . . .	23
2.4.4	Normalisation . . . . .	23
2.4.5	Feature engineering . . . . .	24
2.4.6	Algorithms . . . . .	25
2.4.7	Multi-layer Perceptrons (MLPs) . . . . .	27
2.5	Machine Learning in wider engineering . . . . .	30
2.6	Machine Learning in blast . . . . .	32
2.7	Summary . . . . .	35
<b>3</b>	<b>Numerical modelling</b>	<b>36</b>
3.1	Introduction . . . . .	36
3.2	Viper::Blast . . . . .	36
3.2.1	Introduction . . . . .	36
3.2.2	Simulations methods . . . . .	37
3.2.3	Domain creation and mapping . . . . .	37
3.2.4	Explosive types . . . . .	38
3.2.5	Mesh sensitivity . . . . .	38
3.2.6	Cell size in 3D . . . . .	40
3.2.7	Far-field experimental validation . . . . .	42
3.2.8	Summary . . . . .	45
3.3	LS-DYNA for blast analyses . . . . .	45
3.3.1	Introduction . . . . .	45
3.3.2	ALE mapping . . . . .	45
3.3.3	Experimental validation . . . . .	46
3.3.4	Summary . . . . .	48

<b>4</b>	<b>Informed data mapping</b>	<b>49</b>
4.1	Introduction . . . . .	49
4.2	The Branching Algorithm . . . . .	50
4.2.1	Concept . . . . .	50
4.2.2	Inspiration . . . . .	50
4.2.3	Principles and terminology . . . . .	52
4.2.4	The generalised Branching Algorithm . . . . .	54
4.2.5	Context and potential applications . . . . .	56
4.2.6	The Branching Algorithm for blast analysis . . . . .	57
4.3	Proof of concept for blast analyses in 2D . . . . .	60
4.3.1	Problem scenario . . . . .	60
4.3.2	Model specification . . . . .	61
4.3.3	Algorithm walk-through and output . . . . .	63
4.3.4	Computation times . . . . .	66
4.3.5	Mapping results . . . . .	70
4.4	Summary . . . . .	71
<b>5</b>	<b>The Branching Algorithm in 3D</b>	<b>72</b>
5.1	Introduction . . . . .	72
5.2	Domain discretisation and influence comparison . . . . .	72
5.3	Obstacle influences . . . . .	74
5.3.1	Surface influences . . . . .	75
5.3.2	Edge influences . . . . .	79
5.3.3	Vertex influences . . . . .	80
5.4	Boundary influences . . . . .	80
5.5	Comparison metric . . . . .	81
5.5.1	Shortest path analysis . . . . .	82
5.6	Node connections . . . . .	84
5.6.1	Nearest neighbour analysis . . . . .	84
5.6.2	Charge connections . . . . .	86
5.6.3	Generating free nodes . . . . .	86

5.6.4	Connection accuracy: Clipping . . . . .	89
5.6.5	Path optimisation . . . . .	90
5.7	Alternative methods to represent a domain . . . . .	93
5.7.1	Comparison metric: Ray tracing . . . . .	93
5.7.2	Domain representation: Voxelisation and skewering . . . . .	94
5.8	Interfacing with the Branching Algorithm . . . . .	97
5.9	Additional features . . . . .	97
5.9.1	Probabilistic grouping . . . . .	97
5.9.2	Domain scaling . . . . .	98
5.9.3	Viper::Blast integration . . . . .	98
5.10	Application: Containment structure analysis . . . . .	98
5.10.1	Problem scenario and model specification . . . . .	98
5.10.2	Algorithm output . . . . .	100
5.10.3	Simulation times . . . . .	100
5.10.4	Mapping results . . . . .	103
5.11	Summary . . . . .	104
<b>6</b>	<b>A Direction-encoded framework for Machine Learning tools</b>	<b>106</b>
6.1	Introduction . . . . .	106
6.2	Introduction to the Direction-encoded Framework . . . . .	106
6.2.1	Adapted wave travel distance calculation . . . . .	108
6.3	Dataset development and training approach . . . . .	112
6.3.1	Initial model architecture . . . . .	112
6.3.2	Training dataset . . . . .	113
6.3.3	Testing models . . . . .	114
6.3.4	Analysis method . . . . .	116
6.3.5	Viper::Blast modelling . . . . .	116
6.3.6	Performance metrics . . . . .	117
6.4	Initial performance assessment . . . . .	118
6.5	Feature engineering and development . . . . .	121
6.5.1	Introduction . . . . .	121

6.5.2	Rotating laser directions and a mirrored dataset . . . . .	121
6.5.3	Wave reflections with superposition . . . . .	125
6.5.4	Multiple neural networks . . . . .	128
6.5.5	Expanded input patterns . . . . .	130
6.5.6	Kingery and Bulmash pressure input . . . . .	132
6.5.7	Input and output normalisation . . . . .	133
6.5.8	Summary of performance improvements . . . . .	136
6.6	Hyperparameter tuning . . . . .	138
6.7	Developed performance assessment . . . . .	139
6.7.1	Training analysis . . . . .	139
6.7.2	Testing analysis . . . . .	140
6.8	Application: Ear drum rupture . . . . .	145
6.9	Alternative parameter predictions . . . . .	147
6.10	Summary . . . . .	149
<b>7</b>	<b>Direction-encoded Neural Network in Series</b>	<b>152</b>
7.1	Introduction . . . . .	152
7.2	Incremental training framework . . . . .	152
7.3	Analysis options . . . . .	155
7.4	Problem scenario . . . . .	157
7.4.1	Viper::Blast modelling . . . . .	158
7.4.2	DeNN setup conditions . . . . .	159
7.5	Initialisation trials . . . . .	159
7.5.1	Ultimate validation performance . . . . .	159
7.5.2	Defined performance target . . . . .	162
7.6	Prediction variations . . . . .	163
7.7	Analysis method comparison . . . . .	164
7.8	DeNN performance review . . . . .	164
7.9	Summary . . . . .	166

<b>8 Summary and Conclusions</b>	<b>167</b>
8.1 Summary . . . . .	167
8.2 Conclusions . . . . .	169
8.3 Future work opportunities and outlook . . . . .	169
8.3.1 The Branching Algorithm (BA) . . . . .	170
8.3.2 The Direction-encoded Framework and Neural Network . . . . .	170
8.3.3 Outlook . . . . .	171
<b>A BA in 3D: Example read-in file</b>	<b>185</b>

# List of Figures

1.1	A comparison between physics-based modelling tools and quick-running methods in terms of computation time and solution accuracy. . . . .	3
1.2	Comparison of modelling approaches considering computation time, solution accuracy and tool versatility. . . . .	4
2.1	Friedlander Waveform. . . . .	8
2.2	Representation of Hopkinson-Cranz blast wave scaling that can be applied to reflected and incident blast wave parameters (Hopkinson 1915, Cranz 1926). . . . .	10
2.3	Example pressure-time history for a confined explosion. . . . .	13
2.4	Example pressure-time histories throughout an obstructed environment. . .	15
2.5	Example progression through a deterministic model for blast analysis. . . .	17
2.6	Example probabilistic model using Monte-Carlo sampling for a blast analysis, where ‘n’ is the number of randomly selected input combinations adapted from Netherton & Stewart (2016). . . . .	17
2.7	Example ANN structure. . . . .	28
2.8	Example mathematical procedure for a single hidden neuron. . . . .	28
2.9	ReLU and Sigmoid activation functions. . . . .	29
2.10	Previous approach to blast wave analysis using ANNs (Dennis et al. 2021). Reference to the prediction point and charge are made to a user-defined origin. . . . .	33
2.11	Comparison of modelling approaches in relation to the output from Dennis et al. (2021) considering computation time, solution accuracy and tool versatility. . . . .	34
3.1	Overpressure and specific impulse from mesh sensitivity analysis and evaluation of IG and JWL detonation models for PE4. . . . .	39
3.2	Experimental arrangement and gauge positions (Rigby et al. 2015) . . . . .	40
3.3	PE4 validation results for 250g hemispherical charges and gauges placed at 4 m and 8 m stand-offs. . . . .	41
3.4	Overpressure–time histories for 250 g/2 m, 180 g/2 m and 250 g/4 m, PE4 hemispherical charge size/stand-off pairings. Viper arrival times matched to experiments. . . . .	42

3.5	Overpressure–time histories for 350 g/4 m, 250 g/6 m and 290 g/6 m, PE4 hemispherical charge size/stand-off pairings. Viper arrival times matched to experiments. . . . .	43
3.6	Overpressure–time histories for 350 g/6 m, 250 g/8 m and 250 g/10 m, PE4 hemispherical charge size/stand-off pairings. Viper arrival times matched to experiments. . . . .	44
3.7	Pressure-time and specific impulse-time histories for the detonation of a 100 g PE4 charge placed 80 mm from a rigid plate for three distances from the centre of the plate. . . . .	47
4.1	Initial idea for a method that saves computation time with removed simulation steps shaded in grey. Blast waves shown in red, rigid panel in dark grey. . . . .	49
4.2	Example deviation tree obtained using the Branching Algorithm. . . . .	59
4.3	Explosive test arrangements to be used with the BA shown in 2D. Charge is given as a red circle and the impacted panel is shown in grey. Boundaries are ambient non-reflecting. Model numbers are provided in the lower right corner of each image. All dimensions are in millimetres. . . . .	61
4.4	Numerical models showing the various stand-off distances, blast panel diameters (grey rectangle) and the 49.2 mm diameter charge (red semicircle). (a) Models 1, 2 and 3 (b) Models 4, 5 and 6 (c) Models 7, 8 and 9. . . . .	62
4.5	Algorithm output (a) flow chart, (b) deviation tree. Coordinates given in (a) are given in millimetres to indicate where the model will deviate relative to the charge centre in each model. . . . .	67
4.6	Progression of the shock front from the LS-DYNA models showing where data mapping occurs. Faded cells show the simulation steps that are not required if the BA is used. Data mapping is shown with black arrows. . . .	68
4.7	Continuation of Figure 4.6. Progression of the shock front from the LS-DYNA models showing where data mapping occurs. Faded cells show the simulation steps that are not required if the BA is used. Data mapping is shown with black arrows. . . . .	69
4.8	Pressure time histories for model 1 generated from simulations with and without use of the BA. . . . .	70
5.1	Comparison of how a blast wave may interact with a rigid obstacle in 2D and 3D. Red arrows indicate the path of the blast wave emanating from the charge, shown as a red circle. . . . .	72
5.2	Example of how the adopted meshing strategy, using a mesh spacing of 0.1, enables influences to be defined relative to the charge centre, (0,0), in comparable terms regardless of the orientation of the obstacles in each model. . . . .	73

5.3	Variation of values stored by the algorithm enabling comparisons to be made between each identified influence. Blue dots correspond to each node in the discretisation of an arbitrary obstacle. . . . .	74
5.4	Surface identification from an STL file required for defining global influence locations. Between steps i and ii the surface count decreased from 84 to 48 as various triangles were combined to form polygons with varying vertex counts. . . . .	75
5.5	Process of defining influence locations on the surface of an obstacle. . . . .	76
5.6	Visual representation of using area ratios with barycentric coordinates to identify if a point lies within a triangle. Plot ii shows how the value of $A_\mu$ only depends on $\alpha$ . This also applies to $A_\nu$ and $A_\tau$ , with $\beta$ and $\gamma$ respectively. 78	78
5.7	Defining influence locations on the edges of an obstacle. . . . .	79
5.8	Generating boundary influences with the removal of obstacle surface influences on the boundary. . . . .	81
5.9	Example use of Dijkstra's algorithm for shortest path analysis in 2D. . . . .	83
5.10	Visual of the 26 potential connections that an influence or free node can make in the connected graph of the domain. Red nodes are connected to the central node, black dots are not connected. . . . .	85
5.11	Example of how direct connections from the charge may lead to obstacles with no traceable wave path. . . . .	86
5.12	Generating free nodes using a 3D line intersection check approach. . . . .	87
5.13	Example of how a point can be classified as internal or external by projecting a line with sufficient length in any direction, checking the number of obstacle intersections, and determining if the count is odd or even. An odd number of intersections relates to internal points (left), even for external (right). This rule applies in 2D and 3D. . . . .	88
5.14	Method of preventing connections passing through a rigid obstacle. . . . .	89
5.15	Unoptimised path obtained using Dijkstra's algorithm. Charge shown as red circle. Influences shown as grey dots. Path shown as red line. . . . .	90
5.16	Unoptimised path influence types. . . . .	91
5.17	Optimised path influence types. . . . .	91
5.18	Output path obtained using Dijkstra's algorithm and type-run optimisation. 92	92
5.19	Example of a shortest path for a visited influence that is identified when assessing the optimised path of a different influence. . . . .	92
5.20	Example of how ray tracing in 2D can be used to identify wave travel paths. 93	93

5.21	Voxelisation using the skewering approach discussed in Section 5.6.3. Plot ii utilises the internal/external point check shown in Figure 5.13 to define free voxels. . . . .	95
5.22	Voxel spacing results in different wave travel distances even through the models have the same stand-off in the un-voxelised form. Leads to a deviation at the incorrect time, reducing the BA efficiency. Free voxels omitted for clarity. . . . .	96
5.23	Plan view of the five containment structures included in the example analysis.	99
5.24	Plan view of the modelling domain, showing the origin point of the containment structures in Viper::Blast, four potential charge locations, and five gauges used to record pressure-time profiles. . . . .	99
5.25	Generated deviation trees for each group and visual representations of the deviation points for each model. Red lines show the blast wave travel path to the first influence that results in a parameter field that is no longer identical to the parent model. . . . .	101
5.26	Pressure-time history recorded by gauge 2 when simulating model 9 entirely in its own domain (left) compared to when informed data mapping is used (right). Each line type shows which domain the data was record in. . . . .	103
6.1	Proposed approach for generating blast load predictions using an ANN. Reference to the prediction point is made relative to the surroundings and the wave's journey. . . . .	107
6.2	Example of the variation in wave travel distance obtained when using Dijkstra's shortest path algorithm with type-run optimisation (introduced in Section 5.6.5). Coordinates given in (x,y) to show relative distance between points A and B for a charge positioned at (7.5,5). . . . .	108
6.3	Distribution of shortest wave travel paths from the charge to each POI, showing how type-run optimisation removes most inconsistencies in the domain caused by Dijkstra's algorithm. Charge centre positioned at $x = 5, y = 3.5$ , white voids are rigid obstacles. . . . .	110
6.4	Additional path reduction caused by smoothing the output from type-run optimisation. Charge centre positioned at $x = 5, y = 3.5$ , white voids are rigid obstacles. . . . .	111
6.5	Randomly generated training models used to develop the DeNN. . . . .	113
6.6	Selected testing models to be used for unseen ANN performance assessment.	115
6.7	Representation of how data is split for K-fold cross validation during training, showing that this is independent of the two additional models that have been devised for testing the trained DeNN. Adapted from Pannell et al. (2022). . . . .	116

6.8	T1 peak overpressure targets generated by Viper::Blast, predictions from the DeNN, and the resulting absolute errors. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone. . . . .	119
6.9	T2 peak overpressure targets generated by Viper::Blast, predictions from the DeNN, and the resulting absolute errors. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone. . . . .	120
6.10	Example of how the network could be provided with differing inputs for the same expected output prediction. Use of rotating laser directions ensures identical input patterns to maintain consistency in the predictions. . . . .	122
6.11	Proof of how mirroring the dataset can expand the number of points in training without duplicating input patterns. . . . .	123
6.12	DeNN predictions showing when the directional inputs are fixed in position relative to the domain compared to when a mirrored dataset is used in addition to allowing the directional inputs to rotate such that direction 1 points towards the charge centre. . . . .	124
6.13	Example of how wave reflections are represented by the input pattern. Larger values for directional inputs 4, 5, 6, 7 and 8 imply a larger reflection effects. . . . .	125
6.14	Zone of influence examples showing the regions where directional inputs would be treated as ambient at stand-off distances greater than the wave travel distance in accordance to Equation 6.4. . . . .	126
6.15	Additional example directional inputs for various points. Thick black line in the directional rosette indicates direction 1. . . . .	127
6.16	Flowchart showing how all data points are distributed into each ANN. . . . .	128
6.17	Distribution of points predicted by each ANN for T1. Includes 1.5 m un-hatched region around the charge that is not predicted. . . . .	129
6.18	Expanded directional input options. . . . .	131
6.19	Comparison between the peak incident overpressure from a spherical, free air, 1 kg TNT charge calculated for each point using the shortest wave travel distance as the stand-off in the KB method, and Viper::Blast, where the domain geometry is included to account for wave coalescence and reflections. . . . .	133
6.20	Histograms showing the data distributions for three inputs and the peak overpressure output, before and after a log transformation. . . . .	135
6.21	A comparison of the input patterns that are generated for the prediction of peak overpressure when using the DeNN and a Cartesian based network. . . . .	137

6.22	T1 peak overpressure targets generated by Viper::Blast, predictions from the DeNN, and the resulting absolute errors. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone. . . . .	142
6.23	T2 peak overpressure targets generated by Viper::Blast, predictions from the DeNN, and the resulting absolute errors. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone. . . . .	143
6.24	Eardrum rupture levels for T1 calculated by the DeNN and Viper. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone. . . . .	146
6.25	Eardrum rupture levels for T2 calculated by the DeNN and Viper. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone. . . . .	147
6.26	Improved versatility of the Direction-encoded Neural Network shown in relation to its computation time and solution accuracy when compared to the Cartesian approach taken by Dennis et al. (2021). . . . .	150
7.1	Comparison between the development and use of an ANN that uses Cartesian inputs and the Direction-encoded Neural Network, using inputs that are relative to the POI and the charge. . . . .	153
7.2	DeNNIS procedure allowing for incremental training. . . . .	154
7.3	Simulation order options based on the output from the Branching Algorithm considering required diversity in training data points in relation to charge location and domain geometry. . . . .	156
7.4	Four geometries and five independent charge locations creating 20 models in the batch. Model numbers are as follows: domain A (1–5), domain B (6–10), domain C (11–15), domain D (16–20). Obstacle height of 2 m. . . .	157
7.5	Branching Algorithm mapping trees for the batch of domains shown in Figure 7.4. . . . .	158
7.6	DeNNIS method comparison considering MAE of ANN–1 and ANN–2 when evaluating the validation datasets associated with each initialisation option that uses an unstructured Branching Algorithm training framework. . . . .	160
7.7	DeNNIS method comparison considering MAE of ANN–1 and ANN–2 when evaluating the validation datasets associated with each initialisation option that uses a pre-trained DeNN. . . . .	161
A.1	Branching Algorithm read-in file example. . . . .	185

# List of Tables

3.1	Viper::Blast model parameters for PE4 detonations. . . . .	38
3.2	Far field validation with PE4 hemispheres, model details. . . . .	43
3.3	LS-DYNA keyword values. . . . .	46
3.4	Peak reflected pressure and specific impulse values from the experiment and simulation profiles shown in Figure 3.7. . . . .	48
4.1	Model setup parameters. . . . .	60
4.2	Individual influences for each model being considered are formalised in the influence table below. Rear nodes of the panels are omitted for brevity. Influence defining the trunk model is shown in italics. . . . .	63
4.3	Combined influence table is formed and sorted according to the time when each entry is encountered by the blast wave. Entries included in the trunk model are removed. . . . .	64
4.4	Combined influence table for the first sub-trunk. Model 3 is deemed to be the sub-trunk model. . . . .	65
4.5	Required computation time with and without use of the BA method. . . . .	68
5.1	Example influence table structure for 3D analyses. . . . .	74
5.2	Example output mapping table from the BA for use with the numerical solver. . . . .	97
5.3	Comparison of model computation times. . . . .	102
5.4	Comparison of model computation times for the batch. . . . .	102
5.5	Comparison of the peak overpressure recorded at each gauge for each containment structure considering all charge locations. . . . .	104
6.1	Fixed network parameters. . . . .	112
6.2	Training dataset variable statistics. . . . .	114
6.3	Viper::Blast training model parameters. . . . .	117
6.4	Mean performance metrics from 4-fold cross validation of the DeNN. . . . .	118
6.5	Performance metrics for both testing domains, simulated using the DeNN, compared to a Viper model. . . . .	118

6.6	DeNN performance comparison between the initial feature set, presented in Section 6.4, to when rotating laser directions and a mirrored dataset is incorporated into input generation. . . . .	124
6.7	DeNN performance comparison between the current best performing feature set, presented in Section 6.5.2, with when the superposition equation is also incorporated in input generation. . . . .	128
6.8	DeNN performance comparison between the current best performing feature set, presented in Section 6.5.3, with when the multiple neural networks are used. . . . .	129
6.9	Performance comparison between when 8, 12 and 16 directional inputs are provided in the input pattern of the DeNN. Rotating laser directions, the superposition equation, mirrored dataset and multiple ANNs are included in input generation. . . . .	130
6.10	DeNN performance comparison between using the wave travel distance as an input and when this is translated to an equivalent free air incident overpressure generated by the KB method. Rotating laser directions, the superposition equation, mirrored dataset and multiple ANNs are included in input generation. Best statistics shown in bold. . . . .	132
6.11	DeNN performance statistics comparing when no normalisation is applied to inputs or outputs versus when log based normalisation is used. Rotating laser directions, the superposition equation, mirrored dataset, multiple ANNs and 16 directional lasers are included in input generation. Best statistics shown in bold. . . . .	136
6.12	Performance progression with testing models throughout the feature engineering process for the DeNN. Predictions of peak overpressure. . . . .	136
6.13	Fixed network parameters. . . . .	138
6.14	Tuned hyperparameter options, ranges and sampling methods. . . . .	139
6.15	Tuned hyperparameters for the developed DeNNs. . . . .	139
6.16	Mean performance metrics from 4-fold cross validation of the tuned, developed DeNN. . . . .	139
6.17	Variation of tuned network predictive performance during validation relative to the target peak overpressure magnitude. . . . .	140
6.18	Performance metrics for both testing domains, simulated using the DeNN, compared to a Viper model. . . . .	141
6.19	Variation of network predictive performance for both testing models relative to the target peak overpressure magnitude. . . . .	144
6.20	Overpressure eardrum rupture limits (Denny et al. 2021 <i>a</i> ). . . . .	145
6.21	Training dataset target variable statistics. . . . .	148

6.22	Mean performance metrics from 4-fold cross validation of the DeNN when predicting peak specific impulse. . . . .	148
6.23	Mean performance metrics from 4-fold cross validation of the DeNN when predicting time of arrival. . . . .	149
7.1	DeNNIS analysis options. . . . .	155
7.2	Viper::Blast training model parameters. . . . .	158
7.3	Direction-encoded Neural Network variables. . . . .	159
7.4	Average batch performance following DeNNIS training using a performance target of 2.5 kPa MAE for both ANN-1 and ANN-2. . . . .	162
7.5	Comparison of DeNN performance variation (ANN-1 and ANN-2) in the batch considering each geometry for pre-trained networks developed with a performance limit of 2.5 kPa MAE and a structured BA training framework. . . . .	163
7.6	Performance comparison of various approaches for analysing the batch of domains. DeNNIS method implemented using a pre-trained DeNN, structured training process and one validation domain selected per geometry. . . . .	164
7.7	Performance metrics for both testing domains, simulated using the DeNN and DeNNIS methods, compared to equivalent Viper models. Bold statistics indicate best performance for each testing model. . . . .	165

# Nomenclature

$A$	area
$A_{1,2}$	coefficients in JWL equation of state
$a$	arbitrary scalar
$B$	coefficient in JWL equation of state
$b$	neuron bias, arbitrary scalar
$C_r$	reflection coefficient
$C_{0:6}$	coefficients in linear polynomial equation of state
$c$	arbitrary scalar
$D$	detonation velocity
$d$	arbitrary scalar
$D_{CJ}$	Chapman-Jouguet detonation velocity
$E_0$	detonation energy per unit volume
$h$	hidden neuron,
$i$	impulse, influence number
$i_r$	reflected positive phase impulse
$i_{so}$	incident positive phase impulse
$i_r^-$	reflected negative phase impulse
$i_{so}^-$	incident negative phase impulse
$K$	scale factor, activation function
$L$	Edge length
$l$	Connection length
$m$	mesh or voxel spacing
$m_n$	measured value
$N$	total number of data points / models
$n$	data point, model number
$o_n$	observed value
$P$	Cartesian coordinate point
$P_{CJ}$	Chapman-Jouguet pressure
$p$	pressure
$p_0$	ambient air pressure
$p_r$	reflected pressure
$p_{so}$	incident pressure
$p_{QSP}$	quasi-static pressure
$R$	distance from charge centre (stand-off)
$r$	length ratio
$R_{1,2}$	coefficients in JWL equation of state
$R_t^2$	Young's correlation coefficient
$s$	skew
$T$	trunk model number

$t$	time / time step
$t_a$	time of arrival
$t_d$	positive phase time duration
$t_d^-$	negative phase time duration
$v$	vector
$W$	explosive mass
$w$	connection weight
$x$	input, Cartesian coordinate
$y$	output, Cartesian coordinate
$Z$	scaled distance
$z$	Cartesian coordinate
$\alpha$	barycentric coordinates area ratio
$\beta$	barycentric coordinates area ratio, greatest unique initial influence
$\gamma$	barycentric coordinates area ratio, comparison measure for when an influence becomes active
$\Delta$	domain
$\epsilon$	deviation conditions
$\theta$	parameter field
$\Lambda$	influence table
$\lambda$	influence(s)
$\mu$	surface vertex
$\nu$	surface vertex
$\rho$	density
$\tau$	surface vertex
$\chi$	subset of inputs
$\Omega$	inputs / initial conditions
$\omega$	coefficient in JWL equation of state

# Chapter 1

## Introduction

### 1.1 Background and motivation

On May 22<sup>nd</sup> 2017, a suicide bomber detonated an improvised explosive device (IED) in the foyer of Manchester Arena, killing 22 people and injuring at least 160 more (Pringle et al. 2020). More recently on August 4<sup>th</sup> 2020, over 181 deaths and 6000 injuries were reported in the city of Beirut following the accidental detonation of an approximately 2750 tonnes of ammonium nitrate (Rigby et al. 2020a).

With the continual presence of terrorist attacks, industrial accidents and conflict taking places all over the world, understanding the risk associated to the detonation of explosive materials is vital for designing and developing protective structures and procedures that can reduce any detrimental impact on human life. A key component of this involves understanding how the blast wave that emanates from an explosive compound propagates and interacts with its surroundings.

Historically, this understanding has been developed using physical experiments in controlled environments where the number of trials, extractable data points and variety of test scenarios is limited by cost, safety and expertise. However, with the widespread availability of computing power, validated numerical methods have become increasingly popular as a means of understanding various phenomena without the aforementioned drawbacks associated to data extraction or health and safety.

Semi-empirical tools, such as the Kingery and Bulmash method (Kingery & Bulmash 1984), have been derived from experimental trials that enable the relationships between variables to be defined by simplified equations and charts. They can therefore be evaluated rapidly with a reduced number of inputs, yet this also restricts their use to a limited range of modelling scenarios. Conversely, validated Computational Fluid Dynamics (CFD) or Finite Element (FE) numerical models, such as Viper::Blast (Stirling 2023) and LS-DYNA (Livermore Software Technology Company 2015), obey conservation laws in a discretisation of space and time to accurately model the physics of a detonation, and the subsequent wave propagation or structural response. This makes them well suited to evaluating diverse problems, but, computation times can last many hours, or days, depending on the complexity of the problem and the desired level of predictive accuracy.

At present, regardless of the tool being selected to analyse a given problem, the analysis is often conducted deterministically, meaning a single output is provided for a well-defined problem. However, this approach disregards the inherent uncertainty associated to the charge size, shape, location and material, hence, relying on a number of assumptions that cannot be known before an explosion occurs. Probabilistic approaches are therefore becoming more common so that the risk associated to a specific outcome can be evaluated.

Despite this, the move towards this approach still relies on a suitable deterministic model that can accurately represent the physics of the various scenarios being simulated. Moreover, the chosen model must be rapid in its execution of each unique scenario so that the time required to develop a comprehensive understanding of the threat is not unfeasible. Figure 1.1 presents this challenge with a zone that is often thought to be the goal for the development of new Fast Running Engineering Models (FREMs).

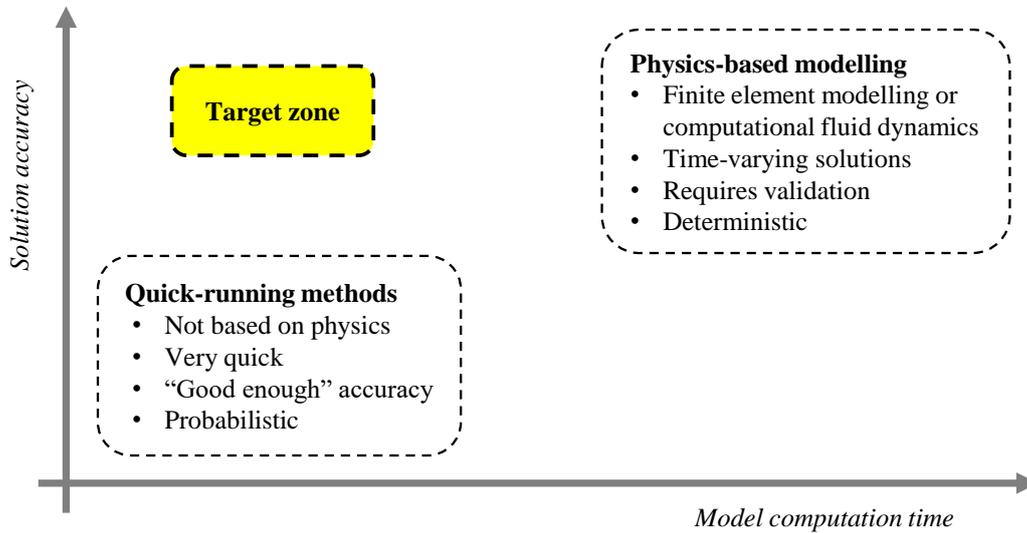


Figure 1.1: A comparison between physics-based modelling tools and quick-running methods in terms of computation time and solution accuracy.

Machine Learning (ML) methods present a means of achieving this goal due to their rapid execution times and simplified mathematical operations that have been shown to provide low percentage errors and high correlation coefficients between targets and predictions in a wide range of disciplines. Through a process of training, ML tools learn the relationships between a series of input variables to optimise various parameters and improve the predictive accuracy. This enables a developed model to generalise highly complex, multi-parameter problems to generate predictions for unseen input combinations, provided that they fall within the bounds of the training variables (Dennis et al. 2021).

There are many instances of ML tools being used to solve problems related to blast wave propagation, including applications for Boiling Liquid Expanding Vapour Explosions (BLEVEs) and the assessment of localised blast on naval structures (Li, Wang, Shao, Li & Hao 2023, Neto et al. 2017). Each study presents notable developments related to how ML based tools could be created, yet, the input parameters are intrinsically linked to the scenario being modelled. In some cases this means that a change to the basic domain arrangement would require a new model to be developed, and since it is also common for training datasets to be formed through a data collection process that requires the simulation of tens, or hundreds, of physics-based numerical models, the computation time associated to development can therefore become comparable to exclusively using numerical models in the analysis of a given problem.

Consequently, Figure 1.2 provides an updated view of the target zone for FREM development considering a new set of axes that includes model versatility. This provides a distinction between ML tools that are developed with limited scope for reuse in alterna-

tive studies and tools that are able to be applied more generally to blast loading based problems in probabilistic frameworks. Furthermore, it shows the target zone of this thesis, which discusses the development of a computationally efficient data collection approach that reduces the time required to develop tools directly, in addition to a novel ML model that can be applied to a wide variety of explosive scenarios.

It is hoped that by approaching the development of ML tools with a focus on versatility and development time, future research will expand upon these ideas and continue to work towards creating FREMs that allow for emergency response efforts and protective structure designs to be optimised and improved.

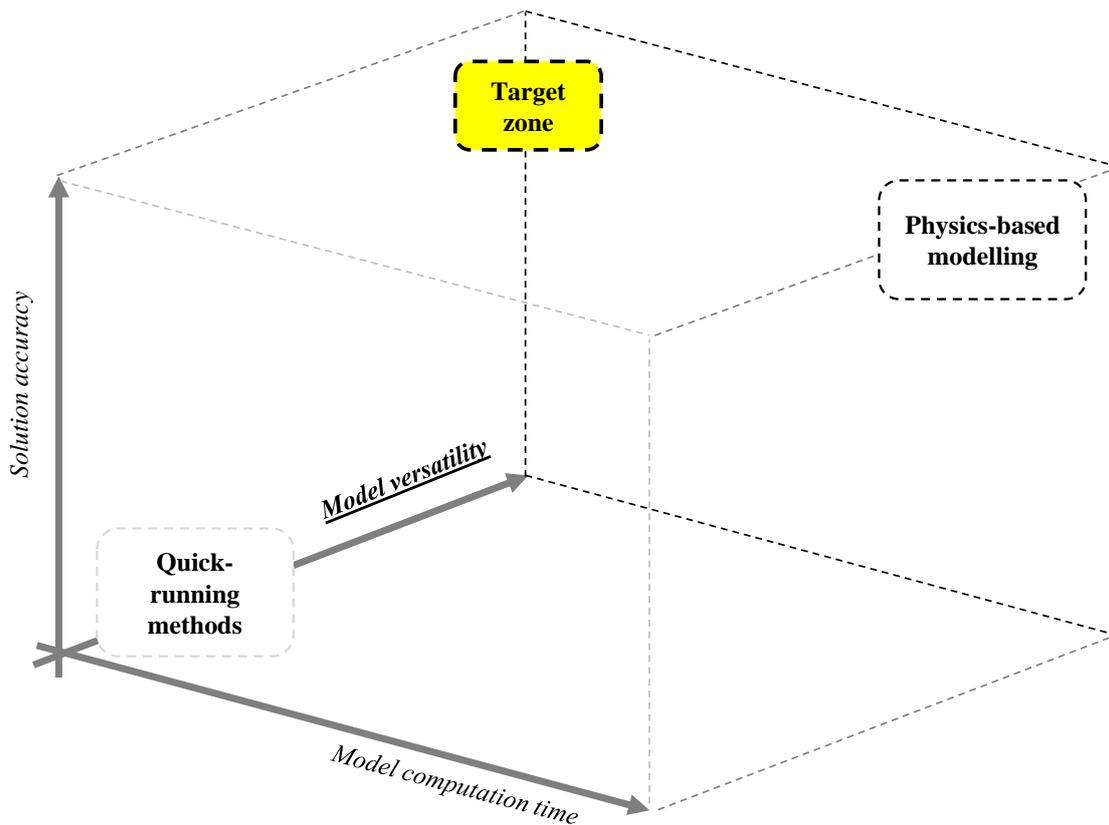


Figure 1.2: Comparison of modelling approaches considering computation time, solution accuracy and tool versatility.

## 1.2 Aims and objectives

The aim of this thesis is to produce the next generation of predictive tools that leverage novel machine learning techniques to evaluate blast loading in complex environments. In particular relating to the computational efficiency of CFD modelling for batch analyses, and the development of versatile ML tools themselves. In order to meet this aim, this project therefore has the following objectives:

1. Evaluate current literature to identify the limitations and opportunities associated to Machine Learning tools used in Fast Running Engineering Model development.
2. Establish a pipeline for generating experimentally validated data from numerical models.
3. Develop an approach for reducing the computation time required to generate training datasets featuring many models that share comparable setup characteristics.
4. Engineer an input set for Machine Learning models that allows for the tool to be used with movable obstacles, and changing domain sizes.
5. Produce a framework for probabilistically modelling variable explosive scenarios that incorporates the training phase of Machine Learning development with its use, removing the need for batches of tests to be evaluated exclusively with numerical models.

### 1.3 Thesis outline

To meet the aforementioned objectives, the following chapters of this thesis are organised as follows:

#### **Chapter 2:** Theory background and literature review

Background information concerning blast wave mechanics and Machine Learning is presented alongside a critical evaluation of current literature related to domains featuring various obstacles, computational blast analysis approaches, and existing Fast Running Engineering Models (FREMs).

*Objective 1 met.*

#### **Chapter 3:** Numerical modelling

Experimental validation of LS-DYNA and Viper::Blast, establishing parameters used throughout this study to generate data used in the development of various tools.

*Objective 2 met.*

#### **Chapter 4:** Informed data mapping to reduce computation times

Development of the Branching Algorithm, an approach that ensures calculation steps in batches of models are only computed once, thus enabling informed data mapping to reduce the required computation time.

*Objective 3 met.*

#### **Chapter 5:** The Branching Algorithm in 3D

Adaptation of the Branching Algorithm in three dimensions, including results from its application with a batch of models.

**Chapter 6:** A Direction-encoded Framework for Machine Learning Tools

Introduction to the Direction-encoded Framework for Machine Learning Tools, a novel approach to structuring the inputs of a model to predict blast loading in obstructed environments with varying charge locations and structural arrangements. Applied as the Direction-encoded Neural Network (DeNN).

*Objective 4 met.*

**Chapter 7:** Direction-encoded Neural Network in series (DeNNIS)

Integration and application of the Direction-encoded Neural Network with the Branching Algorithm. Highlighting the cumulative benefit in terms of computation time.

*Objective 5 met.*

**Chapter 8:** Summary and Conclusions

Conclusions from this thesis with suggestions for future work in this subject area.

## 1.4 Published work

Various works from this thesis have been published in peer-reviewed journals or presented at conferences. This following list provides each publication in order of submission:

- Dennis, A. A., Smyl, D. J., Stirling, C. G., & Rigby, S. E. (2023). ‘A branching algorithm to reduce computational time of batch models: Application for blast analyses’, *International Journal of Protective Structures* **14**(2), 135–167.
- Dennis, A. A. & Rigby, S. E. (2023b) ‘The Direction-encoded Neural Network: A machine learning approach to rapidly predict blast loading in obstructed environments’, *International Journal of Protective Structures*.
- Dennis, A. A., Stirling, C. G. & Rigby, S. E. (2023), Towards the development of Machine Learning tools for blast load prediction, *in* ‘6th International Conference on Protective Structures (ICPS6)’, 14–17 May, Auburn, AL.
- Kirchner, M. R., Kirchner, S. R., Dennis, A. A. & Rigby, S. E. (2023) ‘Non-parametric characterization of blast loads’, *International Journal of Protective Structures*.
- Dennis, A. A. & Rigby, S. E. (2023a), Prediction of Blast Loads using Machine Learning Approaches, *in* ‘SECED 2023 Conference, Earthquake Engineering & Dynamics for a Sustainable Future’, 14–15 September 2023. Cambridge, UK.

# Chapter 2

## Theory background and literature review

### 2.1 Introduction

This chapter provides the background information required to understand the theory and challenges associated with blast wave mechanics in addition to a review of current literature concerning the computational analysis of explosions and various modelling approaches. Included is an introduction to Machine Learning (ML), with a particular focus on the implementation of Artificial Neural Networks (ANNs) for blast applications. This aims to identify ways in which this thesis can improve on the current practice for blast load prediction.

### 2.2 Blast wave mechanics

#### 2.2.1 Shock wave formation and detonation in free air

The development of a shock wave and the release of a high amount of energy can be observed from physical, chemical, or nuclear explosions. In each case, significant loads can be experienced by an impacted body or obstacle as a result of a disturbance of the air molecules in environment surrounding the detonation point.

Physical and nuclear explosions are beyond the scope of this study as they are not commonly associated to blast protection design. Nevertheless, in a chemical explosion in free air, rapid oxidation of carbon and hydrogen atoms breaks intermolecular bonds within the material, releasing the energy that contributes to the devastating effects seen in many of the attacks and accidents discussed in Chapter 1. This is instigated by a detonator that creates a shock wave of energy to start a chemical reaction within the charge mass (Wilkinson & Anderson 2003). As chain reaction process continues, oxygen is used to facilitate oxidation of the fuel elements and the mass is converted into a dense gas that expands rapidly outwards from the initiation point. Any mass of explosive behind the wave front is forced to react, driving the expansion as the air increases in pressure, density and temperature, which in turn displaces surrounding air molecules outside of the charge boundaries. The gases involved in this detonation process can reach pressures of up to 30 GPa and temperatures of 4000°C (Cormie et al. 2009).

Following detonation, the greatest proportion of energy from the explosion is contained as pressure energy in the air molecules that are displaced and compressed by the shock front (Cormie et al. 2009). Higher wave velocities are linked to greater amounts of pressure energy and so a velocity gradient will be present across the shock wave as it emerges from the explosive. This propagates outwards from the charge until the high pressure regions overtake the front of the disturbance, ultimately resulting in a discontinuous step change

in pressure, density and temperature that is characteristic of identifying when the shock front arrives at a target (Baker 1973).

As the blast wave continues to expand, its energy dissipates to the surrounding air, maintaining constant energy within the system. Yet, this causes a reduction in the wave's pressure and density, meaning that at larger stand-off distances from the charge, the impact of the blast wave is also reduced.

### 2.2.2 Pressure-Time histories

Figure 2.1 displays an example of the Friedlander waveform, representing an idealised pressure-time history that corresponds to a passing shock front at a given stand-off distance in free air.

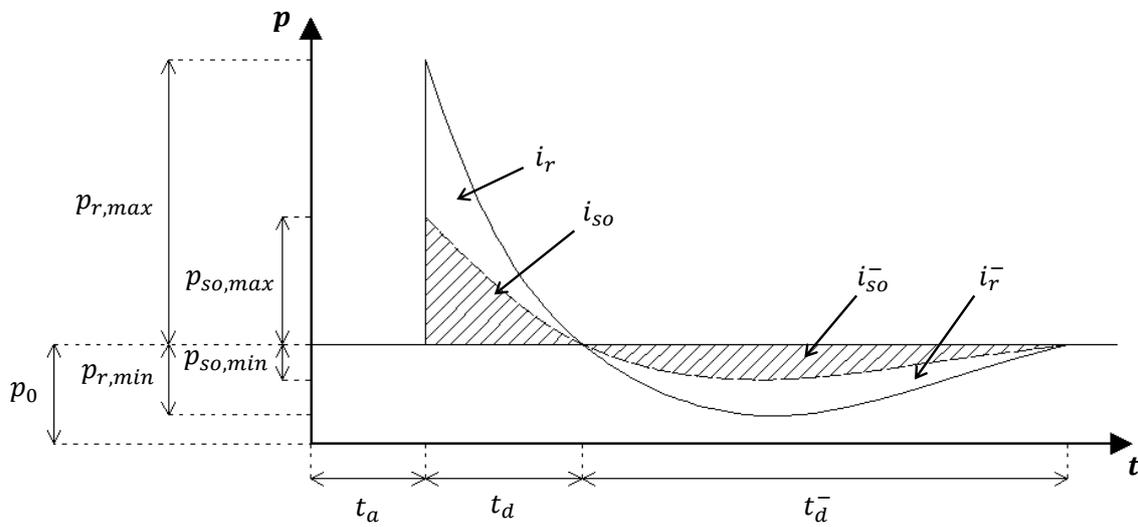


Figure 2.1: Friedlander Waveform.

It is shown that initially atmospheric pressure,  $p_0$ , is maintained, but, as the shock front arrives at time  $t_a$ , the pressure suddenly increases to a peak overpressure. This then decays back to the ambient level over a period of time known as the ‘positive phase’, with a duration  $t_d$ . As the wave continues to propagate, the decay continues into a period of negative overpressure known as the ‘negative phase’. This phenomena is caused by the air that expands after the shock wave has passed, lasting for a duration  $t_d^-$ . Finally, atmospheric pressure is restored after the blast wave has propagated beyond the point of interest, and its energy has been dissipated.

It should be noted that Figure 2.1 shows two different profiles of pressure. The first relating to incident, or ‘side on’ pressure, and the other corresponding to reflected pressure. The reason for the maximum and minimum reflected overpressures,  $p_{r,max}$  and  $p_{r,min}$ , being higher in magnitude than the incident overpressures,  $p_{so,max}$  and  $p_{so,min}$  will be explained in the following section.

Another key parameter linked to blast loading is the specific impulse, found through integrating the pressure with respect to time. In Figure 2.1 these values are denoted by  $i_r$  &  $i_{so}$  for the positive phase and  $i_r^-$  &  $i_{so}^-$  for the negative phase of the reflected and incident profiles respectively.

### 2.2.3 TNT equivalence

The chemical compound Trinitrotoluene (TNT) was a commonly used as an explosive during the mid 1900s when many of the foundational semi-empirical models and equations were derived for use in blast engineering (Kingery & Bulmash 1984). The development of these tools was therefore reliant on the properties associated to this specific material, limiting them from being applied with any other explosive unless the mass of a new compound is converted to an equivalent in TNT.

Termed the TNT equivalence,  $TNT_e$ , Equation 2.1 shows how this process is applied, using a conversion factor to scale the mass of a given explosive,  $W$ , to an equivalent mass of TNT,  $W_{TNT}$ , that would maintain the same energy release.

$$W_{TNT} = TNT_e \times W \quad (2.1)$$

For each explosive compound, it is common for differing factors to be used for pressure and impulse conversions. Similarly, variations in experimental work that aims to compare the energy release from multiple materials has also resulted in many different equivalency values being reported in literature. It is therefore essential to understand the potential inaccuracies of predictions formed using this approach, despite recent studies attempting to refine these factors for compounds such as PE4 and C4 (Rigby & Sielicki 2014, Bogosian et al. 2016).

### 2.2.4 Blast scaling

Hopkinson-Cranz blast scaling laws dictate that there is similarity between blast waves produced by two different charge masses at equal scaled distances (Hopkinson 1915, Cranx 1926). This allows for the comparison of multiple blast scenarios through the use of scaled parameters since the pressure generated at a stand-off distance,  $R$ , from an explosive mass,  $W$ , will be similar to that produced by a charge mass,  $K^3W$ , at a distance  $KR$ . This is formalised in Equation 2.2 and visualised by Figure 2.2.

$$Z = \frac{R}{W^{1/3}} \quad (2.2)$$

Where  $Z$  is the scaled distance in units of  $\text{m}/\text{kg}^{1/3}$ . Here the charge mass is in kilograms of TNT and so equivalency conversions will be required. This law does however only apply to the effects from the same charge material and shape (Wilkinson and Anderson, 2003).

Time dependant parameters are scaled by the same value as the stand-off distance, whereas the pressures and velocities are not factored at all. The factor that is used to perform the adjustment on the time dependant values is given by the cube-root of the charge mass as shown in the following set of equations:

$$p_{actual} = p_{scaled} \quad (2.3)$$

$$t_{actual} = t_{scaled}W^{1/3} \quad (2.4)$$

$$\dot{i}_{actual} = \dot{i}_{scaled}W^{1/3} \quad (2.5)$$

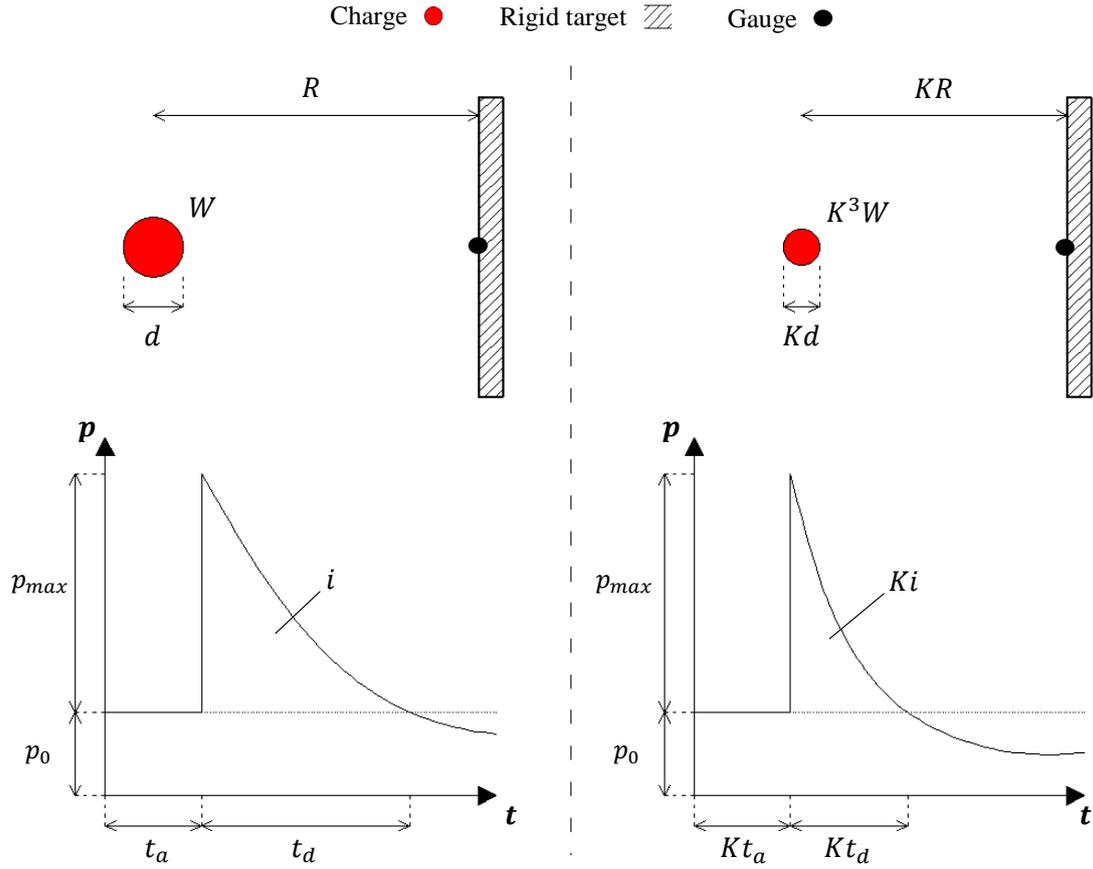


Figure 2.2: Representation of Hopkinson-Cranz blast wave scaling that can be applied to reflected and incident blast wave parameters (Hopkinson 1915, Cranz 1926).

### 2.2.5 Blast wave reflection

As an incident blast wave impacts a target, the laws of conservation of mass, momentum and energy are preserved through an increase in pressure, density and temperature of the blast wave itself. This is because, at the collision interface, the reflected wave propagates back out to the free air, whilst another wave translates through the impacted medium.

To understand this effect, conservation of mass, momentum and energy across discontinuity of compressed air on both sides of the shock front can be considered with the Rankine-Hugoniot Jump conditions to derive Equation 2.6 (Anderson 2011).

$$p_r = 2p_{so} \frac{7p_o + 4p_{so}}{7p_o + p_{so}} \quad (2.6)$$

Where  $p_r$  is the reflected pressure,  $p_{so}$  is the incident pressure, and  $p_o$  is the ambient pressure. Furthermore, a reflection coefficient,  $C$ , given by Equation 2.7, can be defined to represent the ratio of reflected pressure to the incident pressure.

$$C_r = \frac{p_r}{p_{so}} \quad (2.7)$$

Disregarding the assumption that air behaves as an ideal gas at extremely high pressures and temperatures, this coefficient has been shown to reach values as high as 20 (US Army Material Command 1974). Reflected parameters, typically given the subscript ‘r’, are therefore critical in the assessment of blast loads. This also highlights how complex loading topologies can be created when multiple reflecting surfaces are present in a domain, as waves will reflect and coalesce numerous times before the energy has dissipated.

### 2.2.6 Modelling methods

Traditionally, experiments have been widely employed as a means of gaining valuable insights into various aspects of blast loading, including structural response and human injury. With test arrangements that are designed to simulate real-world scenarios, the effects caused by the detonation of an explosive can be directly measured and used to understand the relevant physical phenomena. This includes the derivation of various TNT equivalency values, as demonstrated by Rigby & Sielicki (2014) where hemispherical PE4 charges are found to have an equivalency of 1.2 at any far-field scaled distance, and the characterisation of detonation mechanisms, such as in a study by Tyas et al. (2016) where afterburn is shown to have a large influence on the reflected shock parameters in the near-field. Whilst there is no exact demarcation between the near-field and far-field, it is commonly accepted that the near-field region relates to when the shock wave is still attached to, and driven by, the detonation product cloud (Rigby et al. 2020b). This finding was also enabled by physical experiments and generally persists to a scaled distance of around  $1 \text{ m/kg}^{1/3}$ .

Despite these benefits, requirements for specialised testing facilities, expertise, and financial resources often restricts the number of trials that can be conducted. This therefore limits the ability for complex, large domains to be investigated, particularly because the construction costs associated to each full scale arrangement are likely to be prohibitively high. As a result, Sauvan et al. (2012) suggests that the findings from scaled models show good agreement with the full-sized counterparts, making laboratory-scale testing a cost-effective and safe option. Fouchier et al. (2017) utilises a similar approach to present results from a study using five 1:200 scale models representing different urban layouts. This facilitates the analysis of varied street widths and building heights on key blast parameters.

In situations where scaled models are not required, the location and limited number of extractable data points can also limit the depth of insights that can be obtained from each trial. For each pressure trace that needs to be recorded, an individual gauge must to be calibrated, powered and fixed to a rigid surface at a precise location. In a study concerning the propagation of a blast wave around a corner, Gajewski & Sielicki (2020) contend with this limitation by distributing four gauges along the side of a rigid wall. Findings are used to assess the variability of the blast and the human injury risk for civilians or members of special forces that hide from an explosion. This enables precise wave superposition effects to be captured in each pressure profile, allowing for damage/safety criteria to be assessed. However, gaps in the spatial distribution of recorded values could prevent key conclusions from being derived.

As a means of addressing the aforementioned limitations associated with experimental approaches, numerical analysis provides an alternative modelling method that allows diverse phenomena to be investigated with significant advantages related to cost, safety and efficiency. They can also be used in situations where experimental trials would be impractical or unfeasible given current technological and resource constraints. With a focus on blast wave loading, and not structural response, tools capable of predicting the key variables associated with blast waves are often compared using two performance metrics, accuracy and computational expense. With this, the majority of widely used methods fall into two broad categories, fast running and reasonably accurate, or slow running and highly accurate.

An example from the former is known as the Kingery and Bulmash (KB) method (Kingery & Bulmash 1984). This semi-empirical solver operates with curves that are fit to a dataset generated partly by field tests and partly by computer analyses. It can provide a user with pressure, impulse, duration and the arrival time of explosions with scaled distances between 0.067 and 39.67 m/kg<sup>1/3</sup>. An explosive can also be characterised as either a spherical free air burst, or a hemispherical surface burst. The former being associated to scenarios where the charge is detonated at a sufficient distance from the ground surface such that the wave duration time has elapsed before any reflections arrive at the initiation point. Conversely, a surface burst requires consideration of the reflecting surface that is assumed to be perfectly rigid. The KB approach therefore provides rapid analyses with a good accuracy for simple scenarios that do not require any physical conservation laws. Its simplicity also means that it underpins many widely used tools including the computer programme ConWep and the United States Department of Defence Design Manual UFC-3-340-02 (Hyde 1991, US DoD 2008).

Conversely, computational fluid dynamics (CFD) software packages such as APOLLO Blastsimulator and Viper::Blast are purpose written to solve complex blast scenarios, accounting for conservation laws and wave interactions (Fraunhofer EMI 2018, Stirling 2023). They directly solve the physics of a given problem in a time stepped, discretised calculation procedure that provides the user with a detailed parameter space at the expense of increased computation times that can last many hours, or even days. Despite this, varied charge compositions, sizes and locations can be positioned in domains featuring hundreds or thousands of gauges.

As a result of the large variation in modelling performance, the use of each style of solver will depend on the scenario being modelled, the constraints of the project, and the expertise of the users of each type of tool. It should be noted that while computational modelling and simulation techniques have gained significant popularity in recent years, experiments remain an essential capability in blast research, offering direct observations and validation for theoretical models (e.g. Wu et al. 2013). The combination of experiments and simulations therefore enables a comprehensive understanding of blast effects, leading to improved safety measures and informed design practices.

### 2.2.7 Explosions in confined environments

In Section 2.2.5, the effect of reflecting a blast wave was shown to be critical for the evaluation of structural damage or human injury due to the possibility for increased loading. When analysing internal or confined environments, the inability of the waves to dissipate outside of the domain results in multiple reflections that can lead to far more complex loading profiles than those seen in free-air. This causes regions of intense pressure build-up, particularly in corners where adjoining rigid surfaces meet.

The ‘venting’ capability of a confined environment defines its ability to allow gas to escape, thus alleviating the energy and pressure build-up to allow the blast waves to decay. If a structure has little to no venting capabilities (i.e. no openings or window glazing that is allowed to break upon impact), a Quasi-Static Pressure (QSP) will remain after the initial blast and subsequent reflections have occurred.

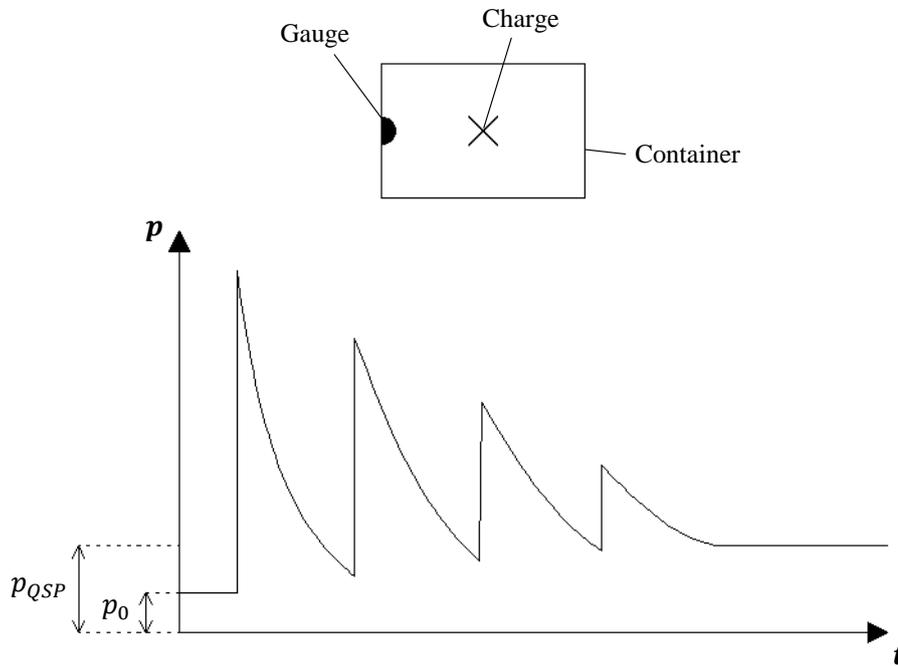


Figure 2.3: Example pressure-time history for a confined explosion.

Figure 2.3 shows how the energy released from the charge is irreversibly transferred to the surrounding air, leading to an increase in the temperature and ambient pressure. In this case, the QSP is shown to remain constant after the event, yet there will always be a mechanism to allow for gradual dissipation that may last for a number of seconds.

The importance of being able to model the effect of confinement is shown by Shehu et al. (2023), where the presence of a QSP and an increased number of reflections is shown to cause larger deformations on plates in confined environments when compared to detonations in free air. Similarly, Anthistle et al. (2016) conduct a series of quarter symmetry experimental trials to assess the influence of adding rigid baffles to an enclosed space that could replicate a train carriage. It is shown that the baffles can have a detrimental impact on the peak pressures and cumulative impulses along the walls where they are placed, due to the increased number of reflections compared to when no baffles are included. Furthermore, the number and spacing of the baffles had a minimal effect on the pressures along

the symmetry plane, highlighting that adding local mitigation measures may not have the desired effect when considering the global domain.

As the complexity of the test space increases, it may be impractical to experimentally model confined explosions, thus necessitating the use of numerical models or simplified calculation procedures. For this, a comparison between empirically derived loading profiles from the UFC-3-340 design manual and a series of recordings from a full scale tests of a partially vented room is made by Codina & Ambrosini (2018). It is shown that the simplified approach is able to accurately predict the peak reflected overpressure both inside and outside the room, however, underestimations of impulse are possible and only a select number of locations could be compared due to experimental limitations.

On the other hand, Remennikov et al. (2022) conducts a series of scaled experimental trials of underground coal mine openings to calibrate the CFD solver, Viper::Blast. The complex loading profiles could not be predicted throughout the domain with any existing simplified approach, and so the CFD tool was shown to be critical for validating that the results could be scaled up to full-scale dimensions. Similarly, in the assessment of train carriages, Larcher et al. (2011) and Larcher et al. (2016) use the FE solver, EUROPLEXUS to conduct analyses of the venting capabilities, structural damage, and human injuries. In Larcher et al. (2016), comparisons are made to experimental data with the authors noting that the numerical model displays good agreement for carriage displacements, but the critical considerations associated to risk of death and ear drum rupture are over predicted.

### 2.2.8 Explosions in obstructed environments

Confined explosions are not considered throughout the remainder of this thesis, however, the previous section aims to highlight how understanding the complexities associated with blast wave coalescence can be both challenging and critical for various applications. In the remaining chapters, domains featuring structural arrangements that will cause the blast wave to undergo reflection, clearing and channelling, with only one rigid reflecting boundary at the ground plane are considered. They are herein defined as ‘obstructed environments’.

Figure 2.4 displays an example of an obstructed environment where a range of these effects can be observed. A 1 kg, hemispherical, TNT charge is detonated on a rigid ground plane and pressure traces are compared to equivalent profiles from a free air burst with the same charge conditions. These illustrative results were generated in the CFD solver, Viper::Blast, according to the methodology detailed in Chapter 3. A gauge focussed on shielding is also included to highlight how the common mitigation measure of adding a barrier between the target and charge can significantly reduce the loading whilst also delaying the pressure peak (plot A). It should be noted that the domain is 2 m tall, with obstacles that are also 2 m in height. This prevents the blast wave from passing over the obstacles to be representative of environments with very tall structures, such as city streets.

The process of clearing occurs when an impacted obstacle has finite dimensions perpendicular to the direction of travel of the blast wave (Rigby et al. 2013). As the blast wave reaches the free edge of the reflecting surface, a reflected shock front propagates away from the target at the same time that the incident shock continues beyond the edge. The pres-

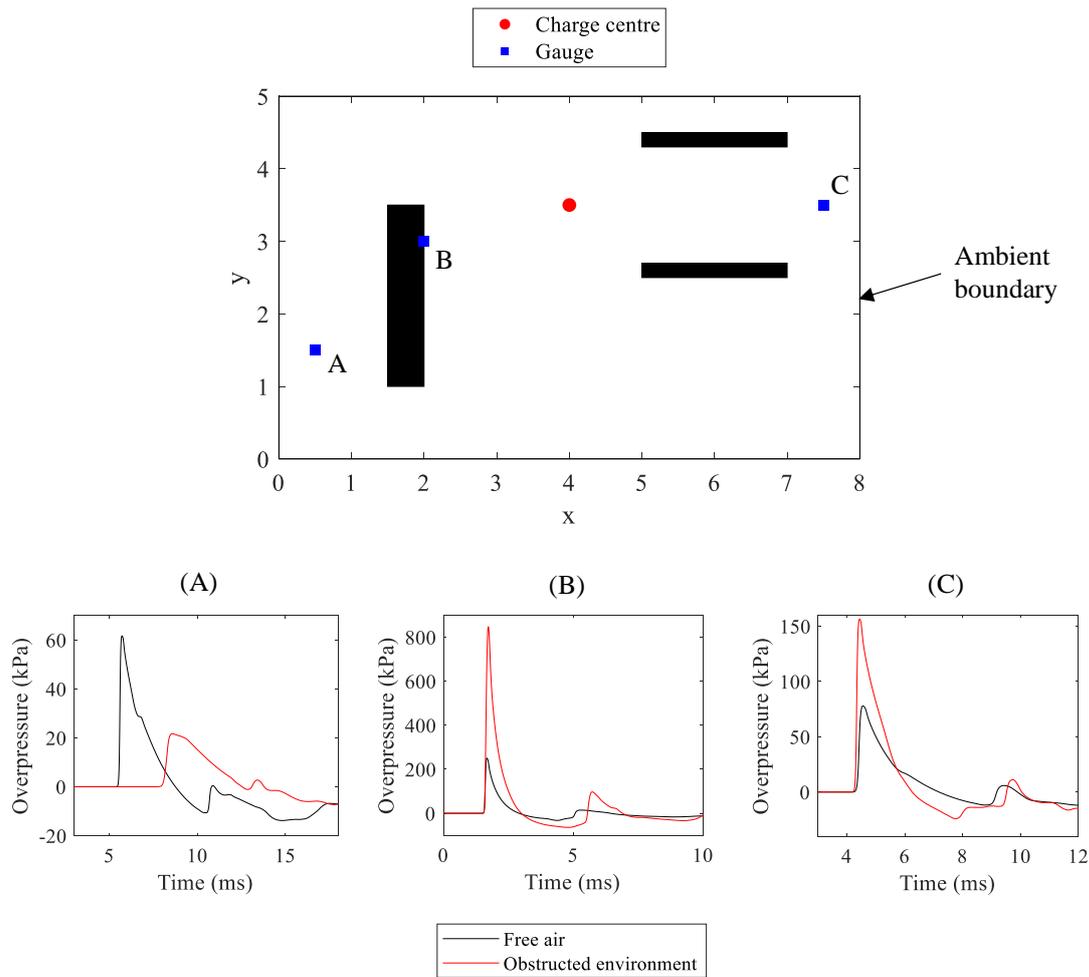


Figure 2.4: Example pressure-time histories throughout an obstructed environment.

sure imbalance causes diffraction around the free edge and the generation of a low pressure rarefaction wave that travels along the loaded surface towards the centre of the target. This reduces the positive phase pressure and impulse acting on the loaded face, making clearing an important mechanism to consider when modelling blast-structure interaction (e.g. Alshammari et al. 2022). However, simplified methods such as the KB approach assumes infinite lateral dimensions, making no adjustment for this effect.

Similarly, channelling (plot C) occurs when the geometry of the surrounding environment influences the propagation of the blast wave, causing multiple reflections to concentrate in certain areas (Isaac et al. 2022a). Simplified methods are therefore unable to calculate the amplification that would be expected in a recorded pressure trace. However, in a study by Codina et al. (2013), use of a numerical model shows that channelling can increase pressure and impulse by a factor of 8 when comparing a confined city street to an unconfined control domain. Remennikov & Rose (2005) also reached a similar conclusion when assessing the validity of the assumption made by simplified tools that blast effects on buildings occur in an isolated open space. It is shown that ConWep is unable to account for multiple wave reflections, in particular when a 300% increase in pressure should be generated as a result of channelling between two buildings.

The drastic amplifications and reductions in pressure and impulse show how the wave interaction processes should be considered in the evaluation of structural response and human safety, particularly in obstructed environments where combinations of each effect can contribute to a single pressure trace. This presents a need for analysis approaches that can accurately model loading variations throughout a given domain, with respect to the specific geometry. Typically, due to the real-world nature of obstructed environments, this is achieved using numerical models or scale experiments.

In the assessment of city-scale explosions, Valsamos et al. (2021) developed an approach that utilises a 3D FE solver with geospatial data from the open-source geographic database, OpenStreetMap. This allows the structural geometry of a city to be modelled, with the subsequent simulation taking into account blast wave interactions at building surfaces throughout the entire obstructed environment. Results from a performance review of the method are presented for a model that aims to replicate the detonation of 2,750 tonnes of ammonium nitrate at the port of Beirut in Lebanon which took place on August 4th 2020. It shows that human injury assessments and structural damage estimations can be generated with an appreciation for the city's unique layout, ultimately allowing for better protective measures to be implemented when considering explosions of this scale.

Furthermore, studies focussed on identifying optimal geometrical shapes that could be used to improve protective designs rely on accurate characterisation of blast effects. For example, in an experimental study of pre-fractal obstacles, Isaac et al. (2022b) observe that reductions in peak overpressure and specific impulse of up to 26% and 19% respectively can occur due to 'trapping'. This is a mechanism that involves a blast wave losing energy through multiple reflections inside a shape with an array of smaller obstacles. Additionally, Gautier et al. (2020) use scaled experiments to investigate the influence of the distribution and areal density of cylinders on overpressure and impulse throughout a domain. The cylinders themselves are intended to approximately represent poles, trees, traffic lights or people, thus informing urban planning for blast mitigation.

Clearly, the combination of various wave interaction effects results in a highly non-linear and chaotic process that is dependent upon many factors. Thus making explosions in confined and obstructed environments challenging to model, and predict, even with current state of the art numerical tools. Nevertheless, gaining an understanding of unique geometrical layouts can provide more complete information on the threats posed by explosive events in urban settings, it is therefore key for understanding many accidental and terrorist explosions in populated areas.

### 2.2.9 Computational modelling approaches

The need to understand the effects of an explosion on various targets is underpinned by the devastating impact that an attack or accidental detonation can have, both economically and in terms of loss of life. The inherent uncertainty associated with an explosive event is made evident by how the specific explosive material, its location, and its mass/shape cannot be known *a priori*. As a result, there is a shifting focus in the sector to move away from solely using deterministic analysis methods (providing a definitive output for a well-defined set of input conditions, as shown in Figure 2.5), as these assume that the given scenario replicates the exact, and only, explosion that the target should be designed to resist.

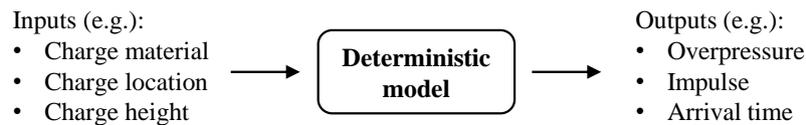


Figure 2.5: Example progression through a deterministic model for blast analysis.

Probabilistic approaches are therefore becoming more prevalent in blast protection research and design as they combat this uncertainty by enabling the analysis of a large number of randomly generated model inputs with additional consideration for modelling uncertainties (Stewart et al. 2006, Stewart & Netherton 2015). Use of various sampling techniques, such as Monte-Carlo (MC) sampling, of the input variables means that randomly selected explosive events are simulated based on the predefined possibilities of each parameter (Murmu et al. 2018). Consequently, instead of being provided with a single output, the user obtains probability distributions of each tracked variable that can be used to inform cost-benefit analyses in decision making processes that consider the risk to a higher fidelity (Netherton & Stewart 2010, 2016). A visual representation of this approach is given in Figure 2.6.

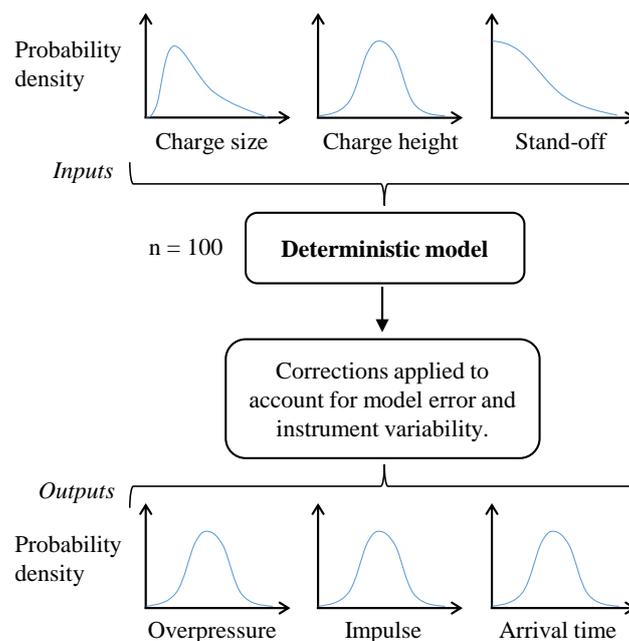


Figure 2.6: Example probabilistic model using Monte-Carlo sampling for a blast analysis, where ‘n’ is the number of randomly selected input combinations adapted from Netherton & Stewart (2016).

The nature of the risk based output from probabilistic analyses is well suited to the determination of human injuries, where the variability of the explosive event can significantly alter the regions where specific injuries could be expected to occur. For example, a study by Gan et al. (2022) utilises this approach to quantify the influence of obstacle density in an internal environment with a focus on the influence of channelling and shielding. However, it has also been shown that it is equally useful to vary key model inputs when optimising structural assessments, such as with the design of sacrificial cladding, or domed structures (Qi et al. 2020, Rebelo & Cismasiu 2021).

In a study by Marks et al. (2021), the authors used MC sampling of select input parameters to conduct a fatality risk assessment along a city street layout with an explosive charge positioned behind a row of bollards. Two significant findings emerged from this analysis, highlighting the efficacy of numerical modelling in this context. Firstly, it was observed that by strategically positioning bollards at a distance of 10 m from the main street within the simulated layout, the fatality risk for individuals in proximity to that street decreased by more than 90%. Secondly, the fatality risks are 10–60% higher if deterministic modelling is used when compared to a probabilistic approach. Both conclusions show how the use of a large number of batch-run numerical models ensures that safer design practices can be identified with a greater level of optimisation.

Despite these benefits, probabilistic studies still require a deterministic model to provide the results for every input combination. Accordingly, if a complex CFD solver were to be used in a probabilistic framework, as with a study by Alterman et al. (2019), the time taken to model all scenarios could be beyond what is acceptable. Here, MC sampling was used to estimate blast loads resulting from the detonation of an improvised explosive device (IED) within a typical ground floor foyer of a commercial or government building. In forming the risk assessment, a computation time of 1.55 hours was required for each of the 100 unique models. Use of a simplified tool such as the KB method would have reduced this computational requirement, however, this would have also limited the depth of the author’s conclusions considering the complexity of the investigated domain. Ideally, a balance between these options would be available to provide rapid, accurate probabilistic results for explosive arrangements of any complexity.

## 2.3 Fast Running Engineering Models (FREMs)

### 2.3.1 Surrogate and Reduced Order Modelling

Surrogate Models (SMs) and Reduced Order Models (ROMs) play a crucial role in Fast Running Engineering Model (FREM) development as they provide methodologies for the creation of computationally efficient representations of complex systems.

A SM, sometimes referred to as a meta-model, serves as an approximation of a system that has a resource-intensive data collection process. They can be developed using various techniques, response surfaces, linear and polynomial regression and Gaussian process regression (Kudela & Matousek 2022). For example, in blast engineering, Pannell et al. (2021) derives a Gaussian function to approximate the specific impulse distribution on a plate in extreme near-field scenarios. This acts as a simplified analysis approach that enables loading to be determined for a wide range of plate widths and explosive stand-off distances without the need for experimentation or numerical analysis.

Additional benefits are also noted by Westermann & Evins (2019), where the use of SMs in sustainable building design covers conceptual design, sensitivity analysis, uncertainty analysis, and optimisation. Here, the authors note that rapid analysis times prevent creativity interruptions, maximise the potential for design exploration and enable a greater appreciation for the impact of varied building parameters to be understood.

Conversely, ROMs strategically reduce the dimensionality of a high-dimensional system while preserving essential features. This is achieved through the identification of dominant modes or patterns in the system's behaviour, and the subsequent construction of a lower-dimensional model (Guo & Hesthaven 2019). A range of methods can be used to construct a ROM, including Proper Orthogonal Decomposition (POD), Dynamic Mode Decomposition (DMD), and Galerkin projection (Hijazi et al. 2020). These approaches combine multiple basis functions that relate the dominant modes, or patterns, in the data to create the required compact representation of a given system. For instance, Xiao et al. (2015) uses POD to solve the Navier-Stokes equations for fluid dynamics, analytically determining the POD coefficients using two novel methods. Results are shown to be highly accurate for flow past a cylinder and the movement of circulating air in domain. As with SMs, the authors also note that this is observed alongside a significant computational speed-up, equal to two orders of magnitude in this case.

The development of a model using either of these methodologies can incorporate data-driven or model-based approaches, with the former utilising a comprehensive dataset of example inputs and outputs to learn the system's behaviour, and the latter producing an analytical representation of the known processes. Despite this, data-driven techniques, such as Machine Learning (ML), are often used for SMs, where underlying equations may not be known or established (e.g. Pannell et al. 2022). Whereas, ROMs are typically developed for systems where governing equations are known, and the improvement in computational efficiency is required due to complex full-order calculation processes.

Nevertheless, whilst both methodologies serve as a means of developing accurate approximations for complex and computationally demanding processes, ensuring the reliability of a developed model will always require appropriate validation that compares the achieved accuracy to the associated full-order system.

### 2.3.2 FREMs in Blast

In Section 2.2.9, probabilistic analyses were introduced as a means of developing a comprehensive understanding of the threat posed by given explosive with an appreciation for the uncertainty of the event. These methods commonly require deterministic numerical models, however, the solution of complex problems using physics-based modelling approaches typically results in unfeasible computation times that restrict the number of scenarios that can be evaluated. Accordingly, there is a need for the development of analysis tools that yield significant improvements in computational time, with a minimal reduction in accuracy of the results.

The benefits of using a FREM in a probabilistic analysis are shown by Seisson et al. (2020), where structural response assessments of masonry panels are performed using a surrogate model based on a single degree of freedom (SDoF) and simplified blast loads that are represented by triangular pressure profiles. The availability of these methods enabled 10000 unique scenarios to be simulated with limited computational expense, ultimately allowing for the influence of changing material properties on failure probability to be assessed. Despite this, limitations associated to the simplified SDoF approach means that the analysis is unable to account for the presence of windows, doors, or any other type of openings in the masonry.

For characterising blast loads on structures, Angelides et al. (2022) present the prediction software, EMBlast. This solver calculates the free-field and reflected blast parameters from a combination of polynomial equations that are fitted to blast trial results and semi-empirical methods. This application of reduced order modelling uses the Low Altitude Multiple Burst (LAMB) shock addition rules (Hikida & Needham 1981) to superpose the outputs from the real charge and numerous image charges to generate pressure-time histories for finite targets with a greater appreciation for the domain geometry than the KB method. The authors note that an additional module is planned to include consideration for internal explosions, but at present, complex, obstructed or confined environments cannot be analysed.

This is partially addressed by Gault et al. (2020), where a fast-running analysis code is developed following a series of experimental trials focussed on the influence of a variation in the charge position in a confined room. Using a combination of path tracing and the derived experimental relations, the code traces the shortest path of the wave to target, predicting the pressure, impulse and time of arrival in around 1 s. However, since experimental relations are used to form the predictions, it is not clear whether this approach could be applied to confined rooms with variable geometries. Thus, limiting the scope of its applications.

Nevertheless, the potential for the code to be expanded to account for varied geometries is evidenced by Frank et al. (2008) and Halswijk (2015), where path tracing (or ray tracing) is shown to provide a means of rapidly analysing the propagation of blast waves in urban environments. In the study by Frank et al. (2008), an image burst method is used to model wave reflections on surfaces around a charge with approximations from the CFD solver, SHAMRC, assisting with capturing the effects of shielding and diffraction. The tool also includes a mechanism to account for Mach stem development from ground reflections. It is noted that within a dense urban environment, predictions are generally good for targets around a corner of with a direct line of sight to the charge, with percentage errors between the peak pressure from experiments and the model being as low as 9.82%. However, pressure amplification and reduction in confined and shielded areas are not represented as consistently, with errors in peak pressure reaching upwards of 50% in many locations.

A similar method is used by Halswijk (2015) in the software, BeamBlast. Here, all of the physically valid paths between the charge and a target are identified and used to superpose a pressure-time history using KB predictions that are modified to account for the path's interactions with domain geometry. This tool can be applied to obstructed and confined geometries, with the authors providing validation for three targets that require an increasing number of reflections to be considered when forming the predictions. For a target on the front face of block that is next to the charge, peak pressure and impulse are predicted with less than 30% and 10% error respectively. Furthermore, arrival times and durations are less than 5% away from scaled experimental data. As the required complexity increases, and the target is positioned within an obstructed environment, predictions of peak values display <20% error. However, the time histories showed that accuracy decreased as the time increases, leading the authors to note that the tool can reproduce the propagation of *"relatively weak blast waves around rectangular obstacles up to a few blast time lengths"*.

Within this tool the user is able to control the level of recursion, specifically related to the number of wave reflections that can be processed. This draws attention to how neither this paper, or the previous one from Frank et al. (2008) provides a numerical run time of the developed methods. Additionally, both studies comment that the tools are fast running, however, neither makes a comparison to the state of the art of CFD solvers and computing power at the time of publication. This makes it challenging to determine whether the path tracing based tools can provide a significant computational benefit to enable the rapid analysis of obstructed environments in 2023, where CFD analyses can be conducted in a number of hours, rather than days.

Lastly, Pope (2011) introduces the Human Injury Predictor (HIP) as a tool that was designed to rapidly predict weapon and injury effects caused by a person-borne improvised explosive device (PBIED) in crowded obstructed environments. It expands upon the aforementioned path tracing tools by incorporating probabilistic positioning of people within a domain to form human injury predictions that account for blast loads and fragment throw on five distinct sections of the human body. To generate a pressure-time history at a target, the HIP code uses a dataset of pre-simulated, free-field blast quantities with scaling and superposition methods to account for interactions with the geometry of the domain. There is an increase in error between the predicted pressure-time histories and comparative CFD analyses as the distance from the explosive increases. This is likely to be associated to the omission of Mach stem consideration, and the application of LAMB rules that breakdown after a certain number of reflections are used to superpose a pressure profile. However, the comprehensive nature of the blast load and fragmentation analysis is well suited to operational needs, meaning that the tool can be used to assist security services in planning crowd management layouts and medical professionals in optimising their response to explosive events.

The benefits of fast running engineering models (FREMs) are therefore clear when considering the need to simulate a wide range of scenarios that help to develop robust understanding of the threat being posed. The range of tools that can be applied to obstructed environments is largely limited by the range of methods that can be used to predict reflected blast quantities in unique geometries where multiple reflections are present. Similarly, it is not clear how efficient many of the published tools are with respect to the current standard of CFD analyses. Throughout the following sections, the concept of Machine Learning (ML) is introduced as an approach that could contribute to the removal of these issues. A review of existing applications of ML in blast analyses is presented following an introduction to the basic concepts of ML that are needed to understand the remainder of this thesis.

## 2.4 An introduction to Machine Learning

### 2.4.1 Introduction

The term Machine Learning (ML) is often used interchangeably with Artificial Intelligence (AI), however, the former is commonly said to be a subset of the latter. Specifically, ML is a branch of AI that is concerned with the development of algorithms that learn from sets of data to make predictions when new data is made available. The output can be interpreted to make decisions, however, the role of ML tools is linked to pattern recognition and the

representation of the relationships between variables (Mohri et al. 2018). Conversely, AI has a larger scope related to the development of systems that can make decisions directly in ways that would normally require human intelligence (Russell & Norvig 2010). To achieve this, Machine Learning algorithms, that may involve Natural Language Processing (NLP), Computer Vision or regressive predictions for example, are often used collaboratively.

The contents of this thesis is focussed on the use of ML tools for blast parameter predictions in environments featuring various obstacles, and so details concerning AI, and its other subsidiaries, are not covered in the remainder of this chapter. Instead, an introduction to how ML tools are developed, with a focus on Artificial Neural Networks (ANNs), is provided.

### 2.4.2 Applications

The use of Machine Learning algorithms in Structural Engineering has seen an exponential rise over the last 10-15 years as researchers aim to implement data driven approaches when solving complex problems (Tapeh & Naser 2023). The specific applications of these algorithms vary greatly depending on the field of study, yet, reviews by Thai (2022) and Flah et al. (2021) note that the most common tasks are focussed on regression, classification, and image processing.

Firstly, Thai (2022) discusses how most publications using ML in structural design focus on regressive parameter predictions for unknown material properties, or classification of failure modes. Many of the typical algorithms, discussed in Section 2.4.6, can be adapted for both of these tasks. However, the key difference is that the former predicts continuous numerical outputs, whereas the latter assigns a discrete category, or class, to an input. Furthermore, in the area of Structural Health Monitoring (SHM), Flah et al. (2021) notes that Computer Vision based models are seeing increased popularity as they use image processing techniques that allow for the identification of patterns, features or objects in images or videos (Voulodimos et al. 2018). This has resulted in studies that generate accurate crack detection predictions or stress fields reconstructions that can assist with understanding the material behaviour with a greater fidelity than a more basic classification output (Cha et al. 2018, Chen et al. 2022).

Beyond this, at the time of writing, the most well-known application of ML is related to Natural Language Processing (NLP). This is driven by the popularity of OpenAI's generative tool, ChatGPT, that combines NLP with Computer Vision to analyse text or image inputs to generate content that resembles human created outputs (Ray 2023). This allows the tool to answer questions and respond to prompts given by a user for tasks including language translation, text simplification and coding (Koubaa et al. 2023). The core function of an NLP is to emulate natural text or speech (Devlin et al. 2019) and so fundamentally, the use cases for these tools differ greatly from the regression and classification models discussed previously. Therefore, whilst ChatGPT, and similar tools, *can* provide numerical outputs to problems, many scientific applications will benefit from having structured outputs that relate closely to the problem being modelled.

In summary, the core applications of ML algorithms are related to regression, classification, computer vision and language processing. These ideas underpin many other potential uses, with regression allowing for the prediction of future values based on historical data

in forecasting, and distinct classes being used in recommendation problems where the most likely outcome needs to be determined. Throughout this thesis, models are developed using ML algorithms to predict blast loads in environments that feature various obstructions, and so the remainder of this section will focus on regressive applications of ML.

### 2.4.3 Development approaches

A training process underpins the development and performance of ML algorithms, with statistical models and trial and error driving improvements in the ability of the tool to represent a given process. Depending on the type of ML algorithm being deployed and the problem being simulated, differing training regimes can be used. Three of the most common are introduced below, however, hybrid approaches are also possible Russell & Norvig (2010).

- Supervised learning: Use of known input and output combinations (a labelled dataset) to enable the model to directly learn the relationship between each variable. Allows for predictions of the outputs associated to new, unseen input data.
- Unsupervised learning: Use of known inputs without known outputs (an unlabelled dataset), requiring the model to identify data structures and patterns.
- Reinforcement learning: Use of inputs from interactions with an environment that leads to feedback in the form of rewards or punishments depending on the chosen action (output).

In each case, the goal of the training process is to iteratively reduce the amount of error in the output (or action). Therefore, there is often a need for large amounts of data to ensure that the interdependencies of the inputs are correctly represented when forming predictions of the outputs.

For applications in Blast Protection Engineering, it is important to validate the performance of a trained model with data that is accurate to the explosive scenario being explored. Accordingly, supervised training is used throughout this thesis to ensure that the model's output is compared to a known target.

### 2.4.4 Normalisation

Prior to training a chosen ML algorithm, normalisation can be employed to scale input, or output, data. This has the effect of standardising variable ranges so that different features are more comparable, thus, preventing a specific variable from having a disproportionate effect on the training progress (Flood & Kartam 1994).

This benefit is highlighted by Sola & Sevilla (1997) where linear transformations are used to normalise inputs that were expressed in different units, with magnitudes that differed by a factor of 100000. For this study, focussed on determining the output from a nuclear power plant, this ensured that larger variables did not insert bias into the algorithm's training process whilst also improving the training speed. Similar positive conclusions are drawn by Singh & Singh (2020), where the authors test 14 different normalisation

techniques for a classification problem. However, it is also noted that some normalisation techniques do not benefit the ultimate performance or training time of a model. Additionally, different normalisation approaches can change the relevance of each variable, which in turn influences the resulting accuracy.

Overall, this preprocessing technique could improve performance of a model, however there is not a single normalisation approach that will guarantee a positive change. In some cases, particularly those where inputs are of comparable magnitude and unit, it is also possible for the process to not be required at all.

### 2.4.5 Feature engineering

In ML, a feature is the name given to a specific input to a model. Feature engineering is therefore concerned with the extraction, transformation, selection, analysis and evaluation of the raw data that is subsequently used to interface with the chosen algorithm (Dong & Liu 2018). It is a process that aims to enhance learning and predictive accuracy by providing relevant and discriminative information.

Normalisation, discussed in the previous section, is therefore considered as feature engineering, as it manipulates the input data in an attempt to improve the ML algorithm's ability to learn. Other common approaches include imputation, where missing values are filled in with mean, median, or advanced techniques (Mera-Gaona et al. 2021); categorical data handling, where categorical variables are converted into numerical formats using methods such as one-hot encoding (Li, Wang, Shao, Li & Hao 2023); and the removal of variables that are deemed to be redundant for an approximation of the system to be produced (Hua et al. 2009). These processes are often considered as standard preprocessing techniques that can be applied to all applications of ML., with the aim of providing a cohesive dataset that has no inconsistencies or gaps that will be difficult for the algorithm to generalise between.

Despite this, domain-specific feature engineering can also be used to provide features that focus on the specific characteristics of a particular problem. This is evidenced by Mende et al. (2023), where the use of domain knowledge when selecting features improves the performance of a regression based Decision Tree model when compared to features that were selected using automatic tools. This is because the relationships between the parameters are better understood by a practitioner in the associated field of manufacturing engineering. However, automatic processes can still be successful, with Huang et al. (2023) showing that predictions of heart disease in patients is improved following a recursive feature elimination process. Here, the chosen model initially trains with all features. Then, the least significant is removed and the processes is repeated until only one feature remains to give a ranking of importance. Further experiments then identify the optimal number of features that provide the best predictive accuracy. Again, this highlights how the process of feature engineering is often iterative, and dependant on the specific problem being modelled.

An alternative approach to selecting features directly from a raw dataset involves feature construction. Here, the developer aims to be provide interaction/combination parameters that either segments a feature or links two or more features together to identify the best way to translate information about the problem to the algorithm. In their study aiming to

estimate ultra-short-term power outputs from turbines, Wei et al. (2023) explores this approach by introducing four new features based on bespoke equations that better represent the characteristics of wind power fluctuations. For example, the horizontal and vertical components of wind speed are calculated using trigonometry to standardise the axes that the measured value is acting on, and lost wind power is estimated by removing actual wind power from a theoretically calculated value for previous time steps. The authors report that this use of domain specific feature engineering allows for high correlations between the inputs and outputs, and the prediction of extreme fluctuation with a high level of accuracy compared to standard approaches.

In summary, many feature engineering techniques exist to format raw data in a way that benefits training of a ML algorithm. The core concepts are typically presented as standard preprocessing techniques, but domain-specific modifications are rarely used to maximise performance. The combination of domain knowledge and suitable ML architectures are vital for the development of useful models, and so this thesis will aim to use the concepts of blast wave mechanics to inform the creation of ML based tools.

#### 2.4.6 Algorithms

There are a number of different ML algorithms that can be developed to solve a given task, including Artificial Neural Networks (ANNs), Decision Trees (DTs) and Support Vector Machines (SVMs). Each option provides a range of positives and negatives associated to their accuracy, training requirements and complexity, and will vary in performance depending on the application, dataset size, data format, training time, memory requirements and interpretability of predictions.

##### Support Vector Machines (SVMs)

SVMs can be applied to classification problems by defining an optimal hyperplane (or decision surface) that separates the parameter space of a given problem into multiple categories. The position of the hyperplane is determined with the goal of maximising the margin between the groups of data, thus making it simple to determine the class that a new input pattern is associated to (Boser et al. 1992). Originally, this was applied to linearly separable patterns, however kernel functions can be used to transform the input features so that SVMs can define non-linear hyperplanes (Cortes & Vapnik 1995, Burges 1998). This enables multi-dimensional, non-linear problems to be evaluated with limited added computational complexity. Similarly, for regression analyses, where the hyperplane is used as a continuous prediction line, kernel functions allow non-linear approximations of the data to be formed. SVMs are therefore easy to interpret and highly effective for analysing smaller datasets.

##### Decision Trees (DT) and Ensemble Learning (EL)

DTs can also be applied to classification and regression tasks with a hierarchical structure that found early success with improving the effectiveness of analysing survey data (e.g. Morgan & Sonquist 1963). A DT is formed from a root node and internal (or decision) nodes, that represent a feature or attribute, branches, representing a decision based on the associated feature, and leaf nodes, assigned to a class or prediction. In supervised

learning regimes, DTs recursively split the input data based on identified features, with the goal of creating subsets that contain samples of the same class or with similar target values for regression problems.

The performance of DTs are often improved by utilising an Ensemble Learning (EL) technique that requires multiple models to be trained independently and then combined when forming final predictions. Various approaches for developing each framework exist, including the following given by Ganaie et al. (2022):

- **Bagging:** Multiple models are trained on different subsets of the training data, using a process called bootstrap sampling. The predictions of the individual models are then combined by averaging or taking the majority vote.
- **Boosting:** Multiple models are trained in a sequence, with each model being trained to correct the errors of the previous model. The final model is a weighted combination of all the individual models.
- **Stacking:** Multiple models are trained on the same data, and their predictions are combined using an additional model that is trained on the outputs of the individual models.

Whilst EL can be applied to any ML algorithm, its application is most common for DTs since an individual tree may not be able to generalise a complex problem space to a sufficient level by itself (Chencho et al. 2021). The original application of the bagging approach is therefore presented by Ho (1995), where the ‘Random Forest’ (RF) algorithm, based on DTs, is introduced. Use of the RF approach improves the generalisation capabilities, robustness and outlier resilience of the developed tool. However, as the complexity of the forest increases, the computation cost also increases and the ability to interpret the predictions is lost.

### **Artificial Neural Networks (ANNs)**

Finally, taking inspiration from how neurons operate in the human brain (Rosenblatt 1958), ANNs are able to identify features in an input dataset, to form predicted outputs, by processing signals of information that are passed between multiple layers of interconnected nodes (or perceptrons/neurons). Typically a network will include of input and output layers, that correspond to the problem being modelled, and a number of hidden layers, where the input information is processed and features are extracted. The particular operation and interconnectivity of the neurons in each layer is dictated by the type of ANN being developed and the desired application, however, when fully trained, they are capable of replicating and generalising multi-parameter, high-complexity problems.

For example, neurons in Multi-Layer Perceptrons (MLPs) receive input signals from all neurons in the previous layer, multiply them by weights, and then standardised the output signal that is passed to all neurons in the following layer. They are therefore commonly used for regression and classification tasks (e.g. Zaleski & Prozument 2018, Chen et al. 2022, Bakalis et al. 2023). Alternatively, Convolution Neural Networks (CNNs) require convolution and pooling layers to transform and subsample pixel values in image and video

processing (Jing et al. 2022). This adaptability of the neuron based approach has encouraged mass experimentation with controlling how information is used in ANNs, resulting in the development of numerous other architectures that provide bespoke benefits for specific tasks. These include, time series analysis and language processing with Recurrent Neural Networks (RNNs) and the generation of realistic images, audio and text using Generative Adversarial Networks (GANs) (Kumar et al. 2021, Goodfellow et al. 2014).

As the number of neurons increases, ANNs can become computationally expensive to train and they may not provide outputs that are interpretable. However, they are commonly applied to problems involving high dimensional data and non-linearity, and are well suited to processing large amounts of data of varied types (Flood & Kartam 1994, Alpaydin 2016). As a result, ANNs are the most commonly used ML algorithm in the field of structural engineering, with 29% of publications using ML from between 2011 and 2020 using this approach (Tapeh & Naser 2023). Although there are no survey papers that provide a similar statistic, it is discussed in Section 2.6 that this trend continues for blast related publications. The remainder of this thesis therefore employs regression MLPs as a means of rapidly predicting blast loads in obstructed environments. This enables meaningful comparisons with the current state of the art, whilst also ensuring that the models are developed with an algorithm that will be capable of accurately representing the highly non-linear wave coalescence effects.

The following sections expand on the information provided by this summary of the core ML algorithms to provide the theory for how MLPs are trained, and how they process information. This is followed by a review of ML applications in blast and protection engineering.

### 2.4.7 Multi-layer Perceptrons (MLPs)

#### Structure

Multi-layer Perceptrons, such as the example shown in Figure 2.7, are formed of individual neurons that are arranged to pass data between input, hidden and output layers via connections. They are defined as ‘fully connected’ ANNs, meaning that each neuron is connected to every other neuron in the layers on step before and after it, and information is passed from left to right in a ‘forward pass’.

The number of hidden layers and neurons in each layer can take any integer value. It is therefore common to experiment with various structures when developing an MLP as the performance of each option will vary, and it cannot be known if a specific structure will provide optimal performance before the analysis has taken place.

#### Neuron calculation

Each neuron is evaluated using the calculation process shown in Figure 2.8. In this example, the inputs to the network ( $x_1$ ,  $x_2$  and  $x_3$ ) are multiplied by numerical ‘weights’ that are assigned to each connection to control the magnitude of the values being passed from neuron to neuron. The summation of the input-weight products is added to a neuron specific ‘bias’ that provides a baseline value to each neuron, before the an ‘activation function’

is used to produce the output value that is translated along the outgoing connections to the next layer.

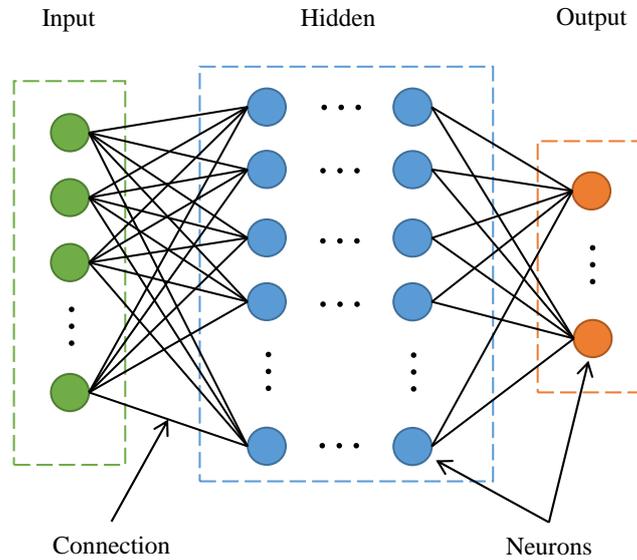


Figure 2.7: Example ANN structure.

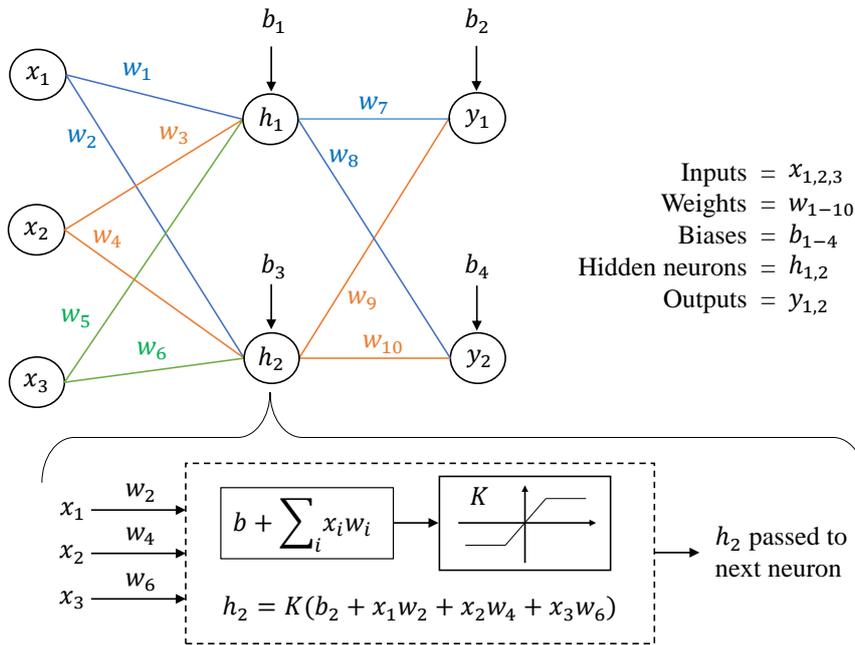


Figure 2.8: Example mathematical procedure for a single hidden neuron.

Activation functions are used to standardise the summation result at each neuron so that inputs of various magnitudes do not skew the values being passed through the network. It is not essential for every neuron to use the same function, however, it is common for regression based analyses to use an unrestricted linear function for the output layer, and either the sigmoid or rectified linear unit (ReLU) functions, given by Figure 2.9 and Equations 2.8 & 2.9, for input and hidden layers.

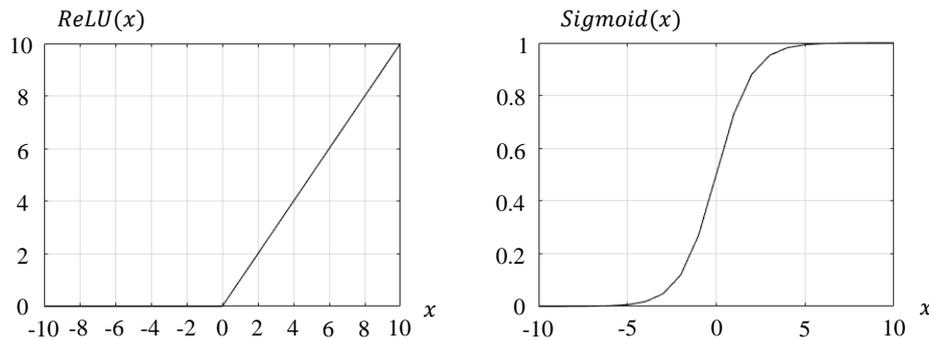


Figure 2.9: ReLU and Sigmoid activation functions.

$$\text{ReLU}(x) = \max(x, 0) \quad (2.8)$$

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (2.9)$$

### Training through Gradient Descent

Performance of an MLP is improved throughout a supervised training process by evaluating a loss function that quantifies the magnitude of error between a target and the output from the forward pass. This error is passed back through the network as part of a gradient descent optimisation algorithm to identify and update the weights and biases that caused the incorrect prediction. This process, known as ‘backpropagation’, occurs once per training step, with each step often including a batch of individual training input sets to benefit computational efficiency.

Gradient descent can be thought of as a means of descending down a topographical representation of the predictive error with each parameter (weights and biases) of the network representing the surface variables. The algorithm’s purpose is to find the adjustment in each parameter that steps closer to the global minima that is associated with optimal performance for the given network structure. Consequently, as with many other factors involved with the construction of an MLP, identification of the best algorithm requires experimentation with various options since the efficiency and accuracy of each method will vary depending on the complexity of the model.

### Regularisation

Following completion of the training process, validation using data that was not used in the training dataset can be evaluated to identify the performance of the ML model without updating any parameters. This process helps to identify if the network has been trained sufficiently, or if underfitting or overfitting has occurred. A model is overfit if it has learnt how to replicate the training dataset instead of generalising the process that is being modelled, learning the noise instead of the underlying patterns in the data. In this instance, the network will need to be retrained with a different setup or number of training steps to reduce its complexity. Conversely, a model that is underfit has not been able to learn the underlying processes, often due to an insufficient number of hidden neurons, training steps, or amount of training data.

Regularisation techniques exist to avoid overfitting in ANNs by applying constraints to the model during training to prevent over complexity. Lasso (L1) and Ridge (L2) regularisation encourages the weights of the network to tend towards 0 by applying a penalty to the loss function based on the sum of the absolute weights or sum of the squared weights respectively. Both methods have the effect of forcing each hidden neuron to have a lesser influence on the model, thus making it more difficult for the algorithm to exhibit high variance in its predictions.

Next, ‘dropout’ is a technique that is used to remove some hidden neurons from each training step, preventing their parameters from being updated in every iteration of the gradient descent process. This helps to restrict the formation of “*brittle co-adaptations that work for the training data but do not generalise to unseen data*” (Srivastava et al. 2014).

Lastly, ‘early stopping’ prevents the use of too many training steps by monitoring the performance of the model after each set of parameter updates. This is achieved by evaluating the validation dataset after each training step, without updating the model parameters. If the validation performance does not improve over a specified number of training steps, the process is terminated as any further improvements are associated to learning noise in the training data.

## 2.5 Machine Learning in wider engineering

Within the extensive domain of engineering, regressive Machine Learning (ML) has become integral in improving the computational efficiency of complex processes. Before discussing its application in blast loading analyses, this section addresses several overarching themes, offering insights into the effective implementation of suitable approaches to maximize the potential benefits.

As discussed in Section 2.3.1, ML can be used in data-driven Reduced Order Modelling (ROM) whereby a chosen algorithm can assist in evaluating the features that define a given process. This is achieved by Wang et al. (2018) where the authors develop a hybrid ROM based on Proper Orthogonal Decomposition (POD) and a Long Short-Term Memory network (LSTM). They note that the ML component is used to construct a set of hypersurfaces that represent the reduced system. The key benefit of this approach is therefore that the trained algorithm can be used to predict the coefficients needed for the POD basis functions for new scenarios without the need for snapshots of data from the corresponding full-order model.

A similar process is described by Xiao et al. (2019) where POD is combined with Gaussian Process Regression (GPR). In this case, the authors identify that the ROM is six orders of magnitude faster than the full-order counterpart and it produces similar results to experimental readings from a wind tunnel. GPR uses a linear combination of Gaussian-shaped functions to construct the surface representation of the system making it suitable for lower dimensional problems. However, the authors note that “*neural networks are often more effective for high-dimensional surface fitting*”. This further reinforces the conclusion of Section 2.4.6, where ANNs were identified to be the most suited to predicting the complexities of blast loading.

Generally, the use of ML in ROM emphasises the capability of various algorithms to generalise a process that is characterised by a series of inputs and outputs. As mentioned above, this is often achieved without replacing an entire full-order model, yet various ML algorithms are capable of learning high-dimensional problems based on many parameters. The alternative, based on Surrogate Modelling (SM), therefore requires the selected approach to learn the intricacies of full-order processes directly.

In predicting the buckling behaviour of thin-walled columns, Mojtabaei et al. (2023) uses a MLP to estimate elastic critical buckling loads, bending moments and the contributions of the key buckling modes. This replaces the Finite Strip Method (FSM) and the Equivalent Nodal Force Method (ENFM) that are computationally expensive to evaluate when the cross-section of a member becomes geometrically advanced. Thus, the surrogate approach allows for more effective design optimisation and experimentation without the use of a reduced-order model.

Surrogates that incorporate ML are also useful when the relationship between outputs and inputs is not explicitly known. For example, Zaparoli Cunha et al. (2023) comments that in active control of noise and vibration, ML is able to map the problem space for scenarios where the mechanistic models are currently unknown and/or incomplete. With this, Cunha et al. (2022) compared four ML algorithms that acted as a surrogate for sound transmission loss. It was found that the regression neural network outperformed the GPR, Random Forest (RF) and Gradient Boosted Trees (GBT) and that accuracy was satisfactory for *"highly non-smooth behaviour resulting from resonant and coincidence phenomena"*. Furthermore, through this performance assessment, the authors placed emphasis on the advantages of applying domain knowledge through feature engineering to produce more accurate and physically consistent predictions. This plays a key role in ensuring the validity of produced model, as it is noted that surrogates based on ML can have poor interpretability due to the lack of a connection to physical equations.

Finally, in situations where the problem surface is too complex for a single ML algorithm to learn the full-order system to a suitable standard, the problem can be broken down into smaller sections. This is best represented in a study by Zaleski & Prozument (2018), where rotational spectra, that can have many species, are evaluated in a framework called RAINet. Here, due to there being a measurable distinction between each type of spectra, a classifier is firstly trained to sort the problem space into known groups. Once classified, an input set is then fed into the relevant ANN that is used to regressively predict key parameters for that specific spectra type. This has the effect of reducing the complexity of the problem space to enhance the predictive accuracy of each individual ANN.

In summary, ML is widely used for SM and ROM across many fields in engineering and science. The ability to produce fast running models in a data-driven way enables robust analyses to be performed for computationally expensive or unknown processes. Risks associated to the generation of physically inaccurate predictions for highly complex processes present challenges to the validation of such tools, yet method such as feature engineering and the use of ML frameworks have been shown to assist with the development of accurate models.

## 2.6 Machine Learning in blast

As discussed in the previous section, ML tools are commonly used to model highly complex, non-linear problems in many fields of study, particularly those where the relationship between the inputs and outputs is not explicitly known (Alizadeh et al. 2017). They are therefore well suited to the analysis of blast waves where complex wave interactions can occur.

MLPs are among the most common type of ANNs and have been shown to be successful in predicting values associated to various task-specific blast scenarios. Most notably work by Remennikov & Mendis (2006), Remennikov & Rose (2007) and Dennis et al. (2021) shows how relatively basic network structures can result in prediction-target correlations ( $R_t^2$ ) over 0.99 for various blast parameters on city streets, behind blast barriers and in enclosed rooms respectively. A correlation of 1 indicates perfect agreement between model outputs and the known targets.

Despite this, in a study by Pannell et al. (2022), it is shown that the predictive performance of an MLP can be improved by incorporating a physical constraint to the loss function that is used in training. This kind of physics informed neural network (PINN) utilised the knowledge that blast waves decay with increasing stand-off to punish the network when predictions were made that did not fit with that expected decrease in blast parameters. This results in improved training times and resulting accuracy, whilst also giving the user confidence that the tool has been developed with a form of physical reasoning that is often omitted from ML development.

Furthermore, Pannell et al. (2023) presents the benefit of an alternative technique termed transfer learning. Here, the requirement for large datasets of new scenarios can be alleviated through knowledge sharing between multiple networks. In this example a trained MLP that predicts the impulse from a spherical charge is embedded into a new network that aims to predict for cylindrical charges. Only the weights and biases of the new connections and neurons are updated throughout the training process, and so the addition to the spherical network effectively applies a scale factor to its output to account for the new charge shape. Ultimately it is shown that the transfer neural network (TNN) consistently outperforms an MLP that was trained without prior knowledge, particularly when less data is available for training.

Recent studies have demonstrated the advantages of using different network types and ML tools over the more commonly used MLP. For instance, 3-dimensional convolution neural networks, typically used in image processing, have been applied to predict peak pressure between buildings with relative errors of less than 7% when compared to equivalent numerical model outputs (Kang & Park 2023). Similarly, transformer neural networks have shown relative errors of less than 3.5% and 14% for predictions of Boiling Liquid Expanding Vapour Explosions (BLEVEs) in free air and around rigid obstacles (Li, Wang, Shao, Li & Hao 2023, Li, Wang, Li, Hao, Wang & Li 2023). These adapted ANNs outperform MLPs when modelling the complexities associated with explosions and wave propagation, and a similar conclusion is reached by Zahedi & Golchin (2022) when using a gradient boosted decision tree to evaluate protruded structures. However, it is important to note that these conclusions require further exploration to assess more varied applications, data processing approaches, amounts of training data, and hyperparameter restrictions.

Regardless of the specific ML tools that is used, the key benefit of using ML approaches is related to the reduction in the computation time required to analyse specific problems. However, training and validation processes require large amounts of high quality data, which is often generated entirely (or at least supplemented by) numerical modelling results, (e.g. Bortolan Neto et al. 2020). Thus, generation of the database itself brings a significant capital investment in computation time. For many studies in the sector, this remains a key issue that is not commonly discussed.

An example of how this becomes prohibitive is made evident in a study by Dennis et al. (2021) where an MLP was trained to predict peak specific impulse on a 2D plane for a charge size of 3–10 kg located within a specific rectangle of a  $10 \times 7$  m domain with around 10% error. Achieving this performance required 72 individual numerical analyses to populate the training/validation data set, each requiring around two hours of computation. The benefit of developing a fast running ANN is clearly offset by this initial investment, highlighting how the reliance on computationally expensive numerical models remains a key issue.

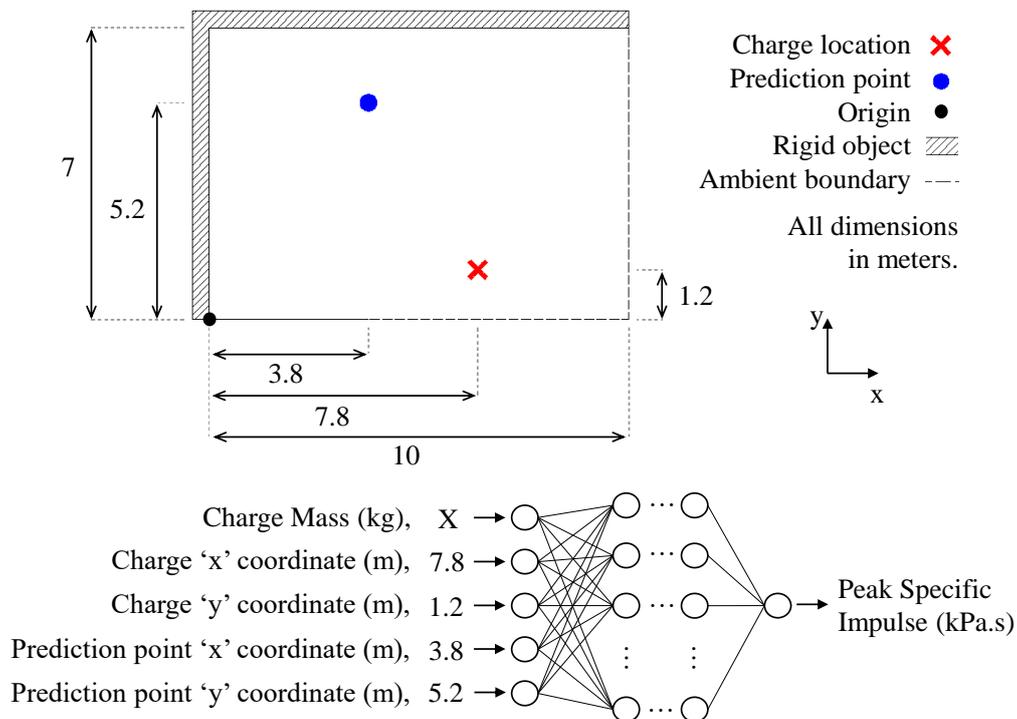


Figure 2.10: Previous approach to blast wave analysis using ANNs (Dennis et al. 2021). Reference to the prediction point and charge are made to a user-defined origin.

Additionally, the application of ML tools is often limited to very specific scenarios due to the nature of task-specific input features and training data. For example, as mentioned above, the study by Dennis et al. (2021) focussed on predictions in an internal environment with a fixed domain size. But, requiring a prediction of a different domain would provide inputs to the network that are outside of its training ranges. The prediction would therefore rely on the model's ability to extrapolate based on the relationships it derived during the training process, yet without the application of transfer learning, neural networks cannot reliably achieve this (Pannell et al. 2023).

This issue is highlighted by Figure 2.10 which shows how inputs are assigned to the network for each prediction point relative to the user-defined origin of the domain. By providing the charge and point of interest (POI) location to the network, a prediction is made with details of the domain size and boundary conditions being embedded into the architecture of the ANN during the training process. A change in these conditions therefore renders the developed network unusable, with the user being required to develop a new tool or conduct additional CFD analysis if, for example, they wanted to evaluate the effect of a blast barrier on the output. Therefore, considering the relationship between computation time, solution accuracy and model versatility, Figure 2.11 shows where a study from the current state of the art is positioned relative to the optimal combination at the target zone.

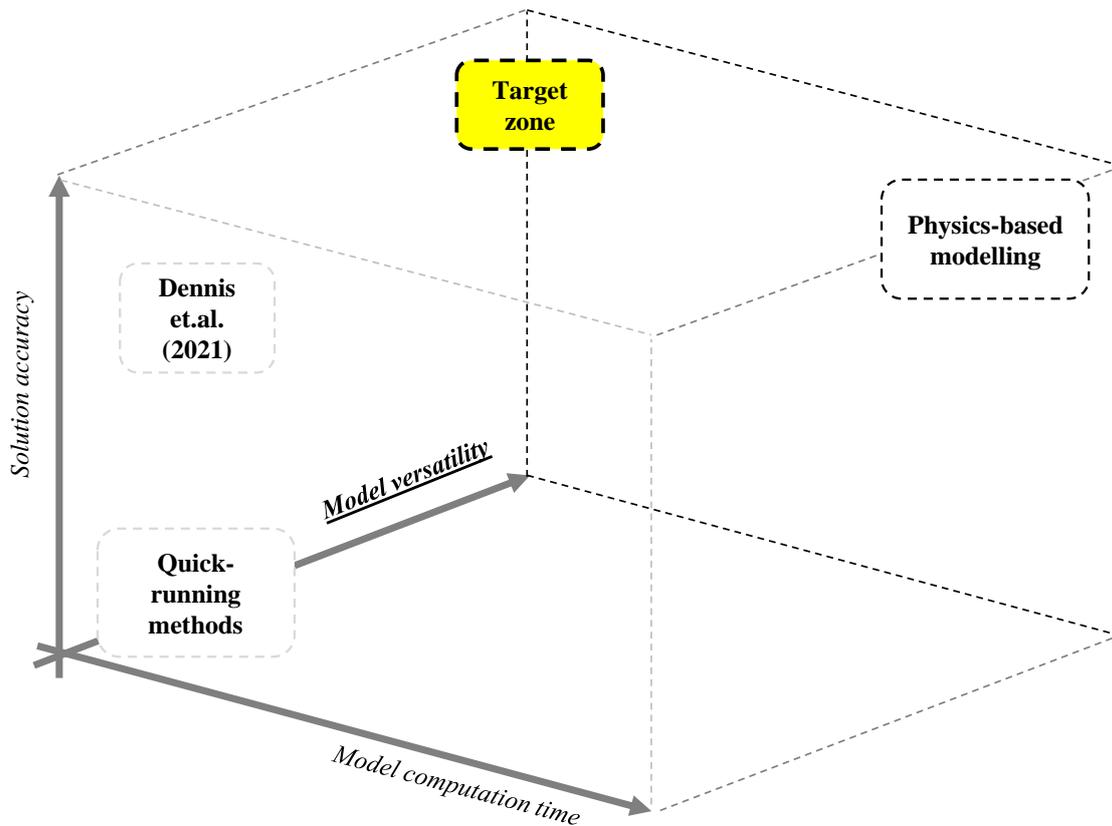


Figure 2.11: Comparison of modelling approaches in relation to the output from Dennis et al. (2021) considering computation time, solution accuracy and tool versatility.

As evidenced by the previous sections of this literature review, the need for probabilistic analyses of complex, obstructed environments is hindered by the lack of analysis tools that are versatile, accurate and fast running. ML approaches provide a means of achieving this goal, however, at present, no such tool exists.

## 2.7 Summary

This section has presented the background theory of various concepts in Blast Engineering that are deemed to be relevant for the contents of this thesis. Through an introduction to detonations and blast wave propagation, it is shown that regions of intense loading caused by wave reflections are critical for the evaluation of confined or obstructed environments.

Insights from the analysis of these environments using probabilistic approaches are shown to account for the inherent uncertainty related to various charge conditions and structural properties. However, use of numerical solvers or physical experiments can limit the number of unique domains being evaluated, thus restricting the depth of the conclusions being formed. Existing simplified analysis tools that generate predictions for these scenarios typically use path finding algorithms and modified free air predictions. This results in poor performance when multiple reflections and dense environments need to be considered, and it is unclear how these tools compare to the current state of the art of CFD analyses in terms of computation time. Surrogate Modelling and Reduced Order Modelling through Machine Learning (ML) was therefore introduced as a means of rapidly analysing non-linear, multi-dimensional regression based tasks. This included a summary of various ML applications, algorithms and development approaches, followed by the key details associated to the development of one of the most common architectures, the multi-layer perceptron.

Findings from this chapter highlight the need for the development of tools that improve the computational efficiency of conducting probabilistic analyses in obstructed environments, resulting in *Objective 1* of this study being met. Throughout Chapters 4, 5, 6 and 7, various methods and tools are developed to address the key issues associated to ML and predictions in complex environments. Specifically, the need for a comprehensive training dataset to be formed using CFD/FE or experimental data, and the lack of a generalised approach to producing predictions in obstructed environments with varied domain geometries.

# Chapter 3

## Numerical modelling

### 3.1 Introduction

Simulating the detonation of a high explosive and the resulting blast wave can be achieved using physics-based numerical solvers that apply the governing equations associated to the conservation of mass, momentum and energy in a discretiation of space and time. The blast-specific Computational Fluid Dynamics (CFD) solver, `Viper::Blast` (Stirling 2023), is used as the primary numerical model in this thesis. It is experimentally validated in this Chapter following an introduction to its key features, a mesh sensitivity study and an assessment of a suitable scale factor to transfer simulation data to a domain using 3D elements after the initial detonation is modelled in 1D.

Following this, the Finite Element (FE) solver LS-DYNA is also introduced as a means of modelling blast wave propagation, with experimental validation being provided for a specific scenario that replicates the simulations conducted in this thesis. A comprehensive scale factor analysis and mesh sensitivity study are not provided for this tool seeing as previous articles have also conducted the same simulations to perform experimental validation (Rigby et al. 2018, 2020b). It is also not the focus of Chapter 4 (where this solver is used) to evaluate the accuracy of the models directly.

The contents of this chapter are presented to meet *Objective 2* of this study.

### 3.2 `Viper::Blast`

#### 3.2.1 Introduction

The numerical solver `Viper::Blast` (herein referred to as `Viper`) is a CFD solver that is founded upon the theoretical framework established by Wada & Liou (1997) and Rose (2001), using the AUSMDV numerical scheme to solve the inviscid Euler equations. It has a wide range of applications in a number of recent studies including an air blast variability analysis, the evaluation of multiple simultaneously detonated charges, and the evaluation of explosions at the opening of a mine (Marks et al. 2021, Zaghoul et al. 2021, Remennikov et al. 2022). It is therefore shown to provide a diverse set of features that enable it to be applied to a range of scenarios, with the most important being detailed in the following sections.

Seeing as this tool also utilises the computing power of Graphics Processing Units (GPUs), it should be noted that all simulations in this thesis were performed using `Viper` version 1.20.6a on a computer that utilises a NVIDIA T1000 dedicated graphics card with a CUDA compute capability of 7.5, in addition to 16GB of system RAM and an Intel Core i7-10700 processor.

### 3.2.2 Simulations methods

Simulations can be performed in Viper using the Jones-Wilkins-Lee afterburn (JWLAB) or Ideal Gas (IG) methods. These approaches represent the charge in differing ways, with the latter applying a single material isothermal burst that represents a fully detonated charge, without detonation products. This is accompanied by an Ideal Gas (IG) model that assumes the domain is filled with gas that obeys ideal gas laws, including the assumptions that the particles are point masses that do not interact except through elastic collisions.

The IG laws are true for low pressure, high temperature interactions, and since the detonation of an explosive will create a high pressure blast wave, these laws are not always applicable and can result in some physical inaccuracies in the simulations. The JWLAB method therefore uses a multi-material, initially undetonated explosive with detonation products that are defined by the Jones-Wilkins-Lee (JWL) (Lee et al. 1968) equations of state. This uses additional parameters for the charge that relates the pressure, volume, temperature and internal energy to the material being simulated. Furthermore, an afterburn model is applied with an energy factor and duration so that any additional energy from detonation product reactions is released into the domain over a specified period of time.

It is noted in the manual of the solver that the JWLAB approach is more suited to scenarios where the expansion of the detonation products needs to be more accurately replicated, such as in the near-field, or if blast-obstacle interaction close to the source is expected. Whereas, IG simulations require reduced computational effort making them more suited to solving far-field problems.

### 3.2.3 Domain creation and mapping

Viper allows simulations to be conducted 1D, 2D or 3D, with data mapping capabilities between each option. In the 1D phase, the solution is provided using a spherical calculation process that terminates as the blast wave reaches the domain boundary. Similarly, 2D models are cylindrically evaluated, however, boundaries can be set to ‘terminate’, ‘transmit’, or ‘reflect’ to allow the simulation to progress beyond the first interaction of the wave with the environment. In both phases, the user is able to control the size of the domain, but additional geometries cannot be added.

Conversely, the 3D phase allows for cuboid domains of any size with obstacle creation via stereolithography (STL) ASCII format file read-in or nodal input. Each domain boundary is set as either ‘reflect’ or ‘transmit’ and obstacles can be defined as breakable or rigid reflecting. For breakable obstacles, an associated Pressure-Impulse diagram should be provided so that failure is caused at the correct time, after which the simulation will progress with the entire obstacle being removed.

Any number of monitoring locations (gauges) can be specified in any phase, either through manual input or text file import. Each gauge can be provided with additional triggers to either terminate the model, or produce a remap file as soon as a non-ambient value is recorded.

### 3.2.4 Explosive types

The version of Viper that is used throughout this document (1.20.6a) provides built-in charge properties for Trinitrotoluene (TNT), Composition C-4 (C4), Ammonium Nitrate/Fuel Oil (ANFO), Nitromethane, Pentaerythritol Tetranitrate (PETN) and Plastic Explosive No.4 (PE4). It also allows for vapour detonations and custom compositions with user defined properties.

It is important to note that the properties provided for PE4 are only suitable for the JWL solving approach. However, as discussed by Bogosian et al. (2016), C4 and PE4 are nominally identical and so the C4 model can be used for IG simulations aimed to replicate the detonation of PE4 if required.

To identify a suitable solving method and cell sizes for both 1D and 3D phases, the following sections provide a mesh sensitivity study and mapping scale factor analysis. The former determines a suitable approach through comparisons to the Kingery and Bulmash (KB) method (Kingery & Bulmash 1984) using TNT and PE4, whereas the latter compares results from a 3D domain to experimental trails of PE4 hemispheres, using various charge sizes and stand-off distances.

### 3.2.5 Mesh sensitivity

A mesh sensitivity analysis is presented to identify the cell size and solving method required to preserve the initial release of energy when simulating the detonation of an explosive in a 1D phase, with the intention of mapping these results to 3D domains throughout the remainder of this thesis. This is achieved by observing convergence of the outputs from a series of gauges in a 1D Viper model with comparisons made to the KB method (Kingery & Bulmash 1984).

Table 3.1: Viper::Blast model parameters for PE4 detonations.

	Parameter	Value	Unit
Generic	Pressure	101325	Pa
	Temperature	288	K
	CFL: 1D	0.5	
	CFL: 3D	0.4	
PE4 (C4): IG	$\rho$	1580	kg/m <sup>3</sup>
	$E_0$	6.06E6	J/kg
PE4: JWL	$\rho$	1576	kg/m <sup>3</sup>
	$E_0$	8.698E6	J/kg
	$A_1$	9.5929E11	Pa
	$R_1$	5.616	
	$A_2$	4.914E9	Pa
	$R_2$	1.804	
	$\omega$	0.136	
	$D_{CJ}$	7929	m/s
$P_{CJ}$	2.4E10	Pa	

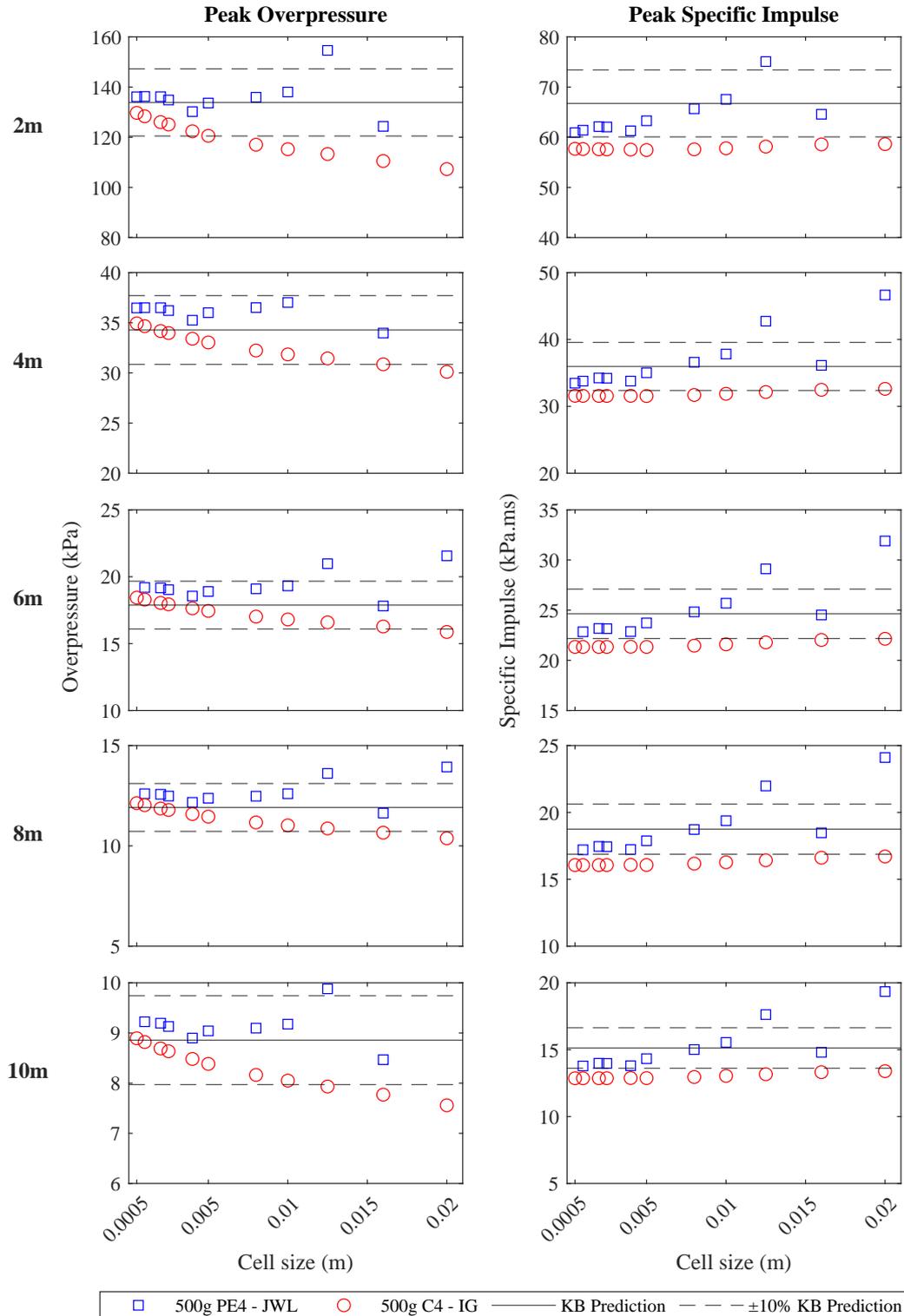


Figure 3.1: Overpressure and specific impulse from mesh sensitivity analysis and evaluation of IG and JWL detonation models for PE4.

PE4 is chosen as the explosive since experimental comparisons provided in the next section will use this compound. Similarly, hemispherical charges are modelled to enable continuity throughout the validation process. Both the JWL and IG solving methods are evaluated to output the incident peak overpressure and specific impulse at five stand-off distances between 2 and 10 m for a charge size of 250 g.

As mentioned in the previous section, the built-in properties provided in Viper for PE4 are only suitable for the JWL solving approach, and so the C4 model can be used for IG simulations. Additionally, each charge is modelled as twice the hemispherical mass to account for how the 1D solver (spherically-symmetric) does not include a reflecting ground surface. Viper input parameters for this process are shown in Table 3.1.

Figure 3.1 shows that as the cell size reduces, the peak pressure and specific impulse of both solving methods begin to converge on the KB predictions. Since the KB method uses semi-empirical equations derived from tests using TNT instead of PE4, an equivalency factor of 1.2 is used to convert the mass of PE4 to an equivalent mass of TNT when predicting these values (Rigby & Sielicki 2014).

For all stand-off distances, the peak overpressure output from Viper is within 10% of the KB output as the cell size approaches 0.005 m. However, specific impulse is routinely under predicted, in particular when using the IG simulation. Convergence of both methods occurs with a cell size of 0.002 m, providing around 20 cells across the radius of the charge.

### 3.2.6 Cell size in 3D

When mapping from 1D to 3D, an increased cell size is required to prevent prohibitively large computation times. However, as the data is interpolated into a 3D domain, cell sizes that are too large will result in rounding of the pressure traces leading to inaccurate estimates of the blast parameters.

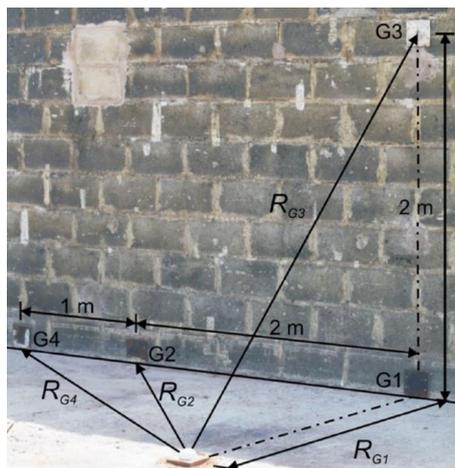


Figure 3.2: Experimental arrangement and gauge positions (Rigby et al. 2015)

In this section, Viper models featuring 250 g PE4 hemispherical charges are compared to experimental results obtained by Rigby et al. (2015) for the arrangement shown in Figure 3.2. A 1D phase is simulated to a stand-off of 2 m with a cell size of 0.002 m to provide  $\sim 21$  cells across the charge radius. This initial detonation is mapped into a series of 3D domains featuring differing cell sizes to identify a suitable increase.

Experimental peak values are taken from curves that were fit to the pressure traces of gauge 1 (G1) to negate the effects of sensor noise and variation. The blockwork wall that the gauge is mounted to is represented in Viper as a non-reflecting boundary.

As shown in Figure 3.3, for stand-off distances of 4 m and 8 m the computation time greatly increases as the cell size decreases below 0.016 m, particularly for JWL simulations. There is also minimal improvement in the peak pressure and specific impulse comparisons below this point. The peak overpressure may be a function of the number of cells in the domain, however, there still appears to be a maximum cell size that should be provided to prevent a loss of resolution that reduces the peak readings. Here, for a 250 g PE4 hemisphere, this is identified to ten times the 1D cell size, 0.02 m. Conversely, the predictions of the peak specific impulse are consistent with all cell sizes, suggesting that it is the initial burst energy, handled in the 1D phase, that is more important to preserve for this parameter.

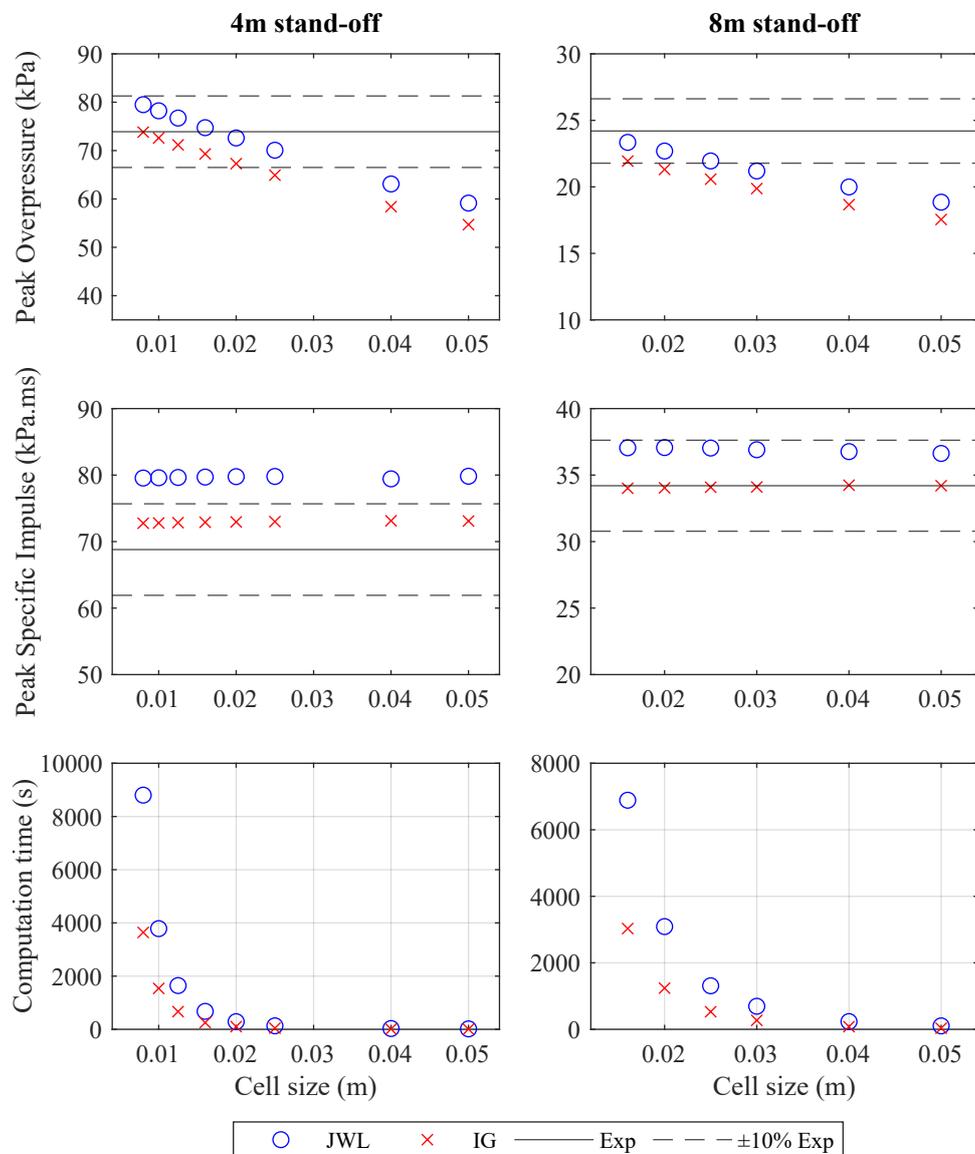


Figure 3.3: PE4 validation results for 250g hemispherical charges and gauges placed at 4 m and 8 m stand-offs.

### 3.2.7 Far-field experimental validation

The previous sections have shown that in order to preserve the peak overpressure and specific impulse of a simulation, the 1D cell size of the Viper models should provide at least 20 cell across the charge radius. Furthermore, in 3D, the cell size should not exceed ten times the maximum 1D cell size. For a 250 g PE4 hemisphere, this restricts the cells to be 20 mm or below. To verify these findings, and assess the ability of Viper to produce reliable pressure-time histories, nine additional comparisons are made against existing experimental data.

Figures 3.4, 3.5 and 3.6 show the pressure-time and specific impulse-time histories for the range of stand-off distances and hemispherical PE4 charge sizes shown in Table 3.2. In every case, IG Viper models have been simulated with 2 mm cells in the 1D phase. Furthermore, 16 mm cells are used in the 3D phase for all domains apart from when the stand-off is 10 m in Figure 3.6, here 20 mm cells were specified to ensure that the entire model could be evaluated using the same Viper solving method, using one GPU.

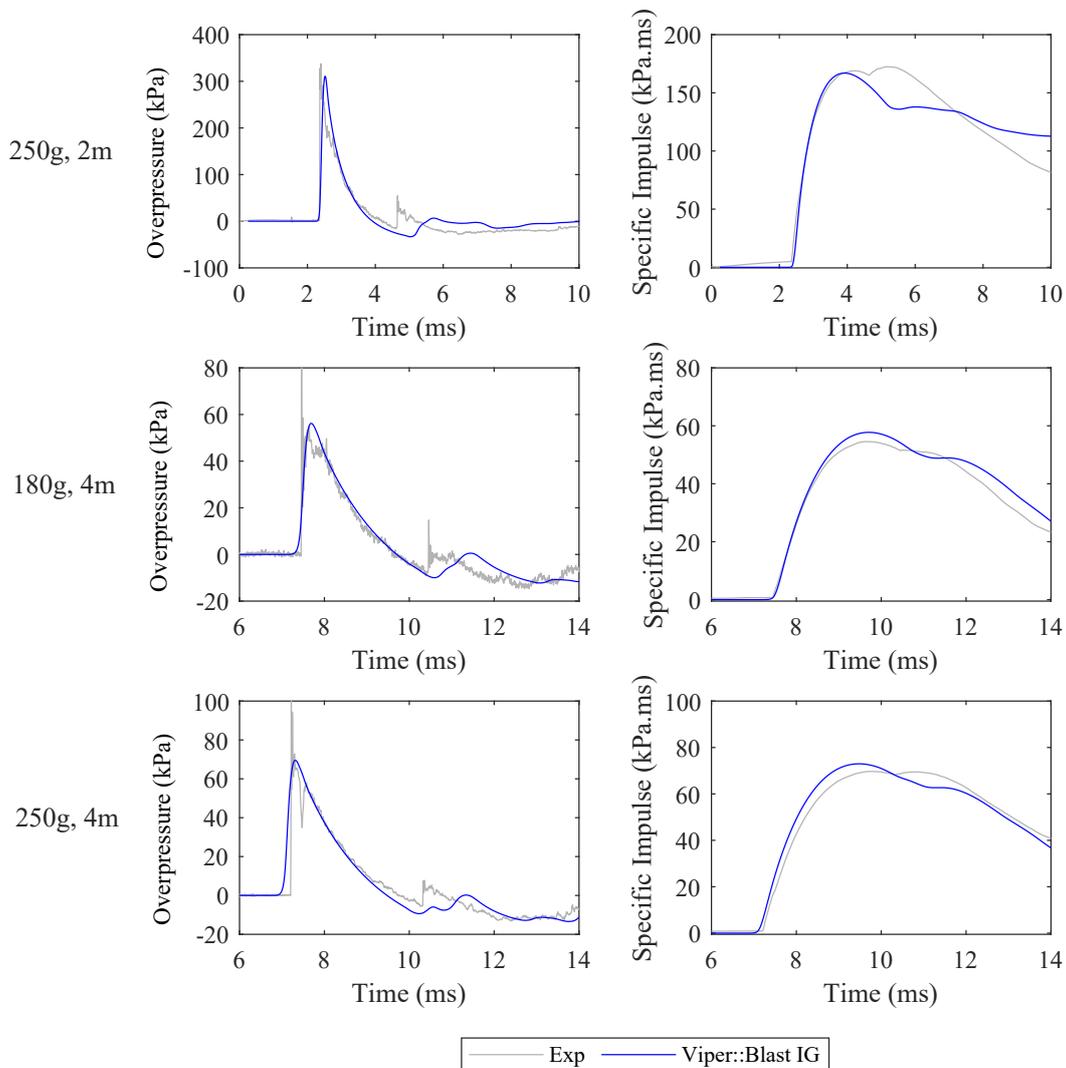


Figure 3.4: Overpressure–time histories for 250 g/2 m, 180 g/2 m and 250 g/4 m, PE4 hemispherical charge size/stand-off pairings. Viper arrival times matched to experiments.

Table 3.2: Far field validation with PE4 hemispheres, model details.

Charge size (g)	Charge radius (m)	Cell size (mm)		Stand-off (m)	Scaled distance (m/kg <sup>1/3</sup> )
		1D	3D		
250	0.042	0.002	0.016	2	2.99
180	0.038	0.002	0.016	4	6.67
250	0.042	0.002	0.016	4	5.98
350	0.047	0.002	0.016	4	5.34
250	0.042	0.002	0.016	6	8.96
290	0.044	0.002	0.016	6	8.53
350	0.047	0.002	0.016	6	8.01
250	0.042	0.002	0.016	8	11.95
250	0.042	0.002	0.020	10	14.94

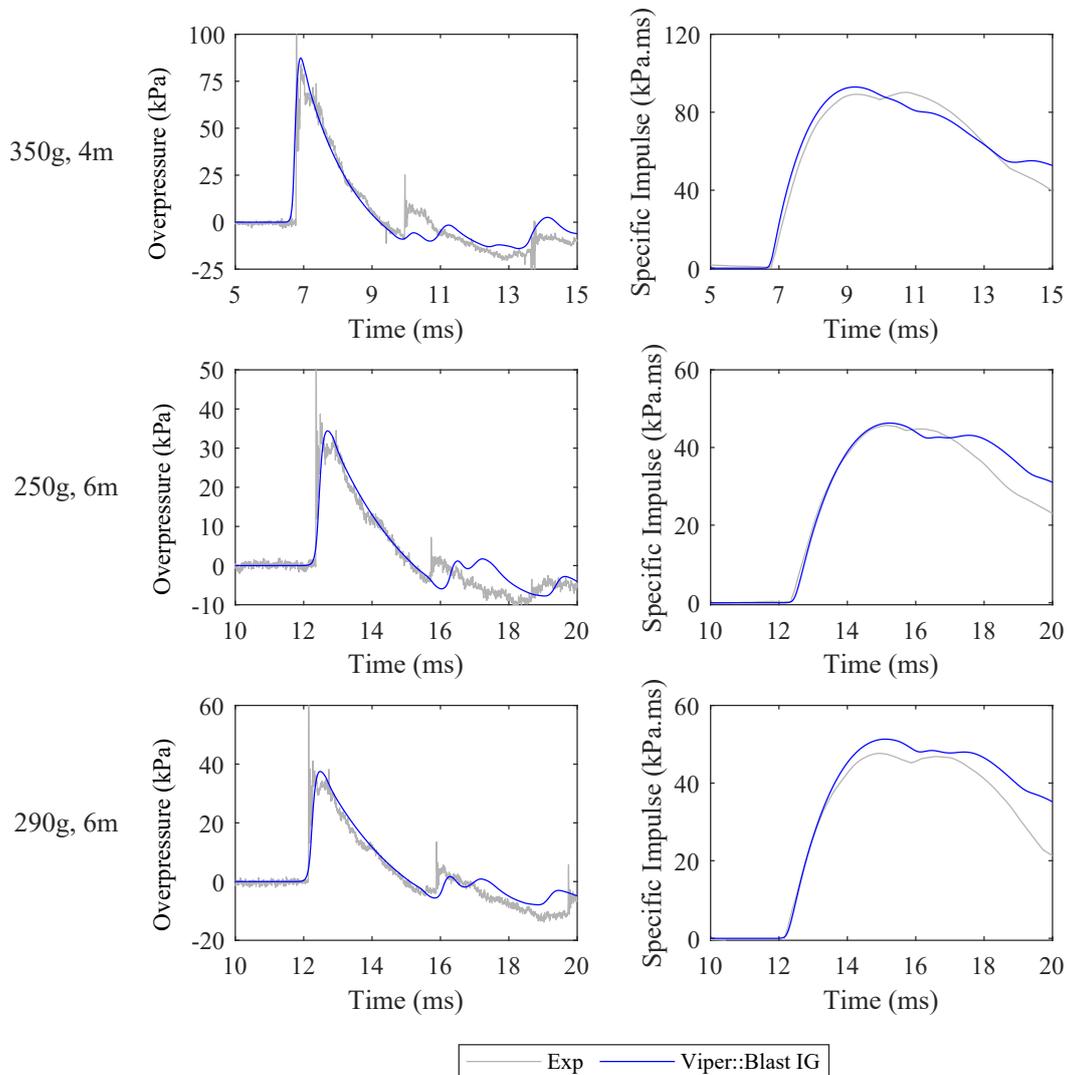


Figure 3.5: Overpressure–time histories for 350 g/4 m, 250 g/6 m and 290 g/6 m, PE4 hemispherical charge size/stand-off pairings. Viper arrival times matched to experiments.

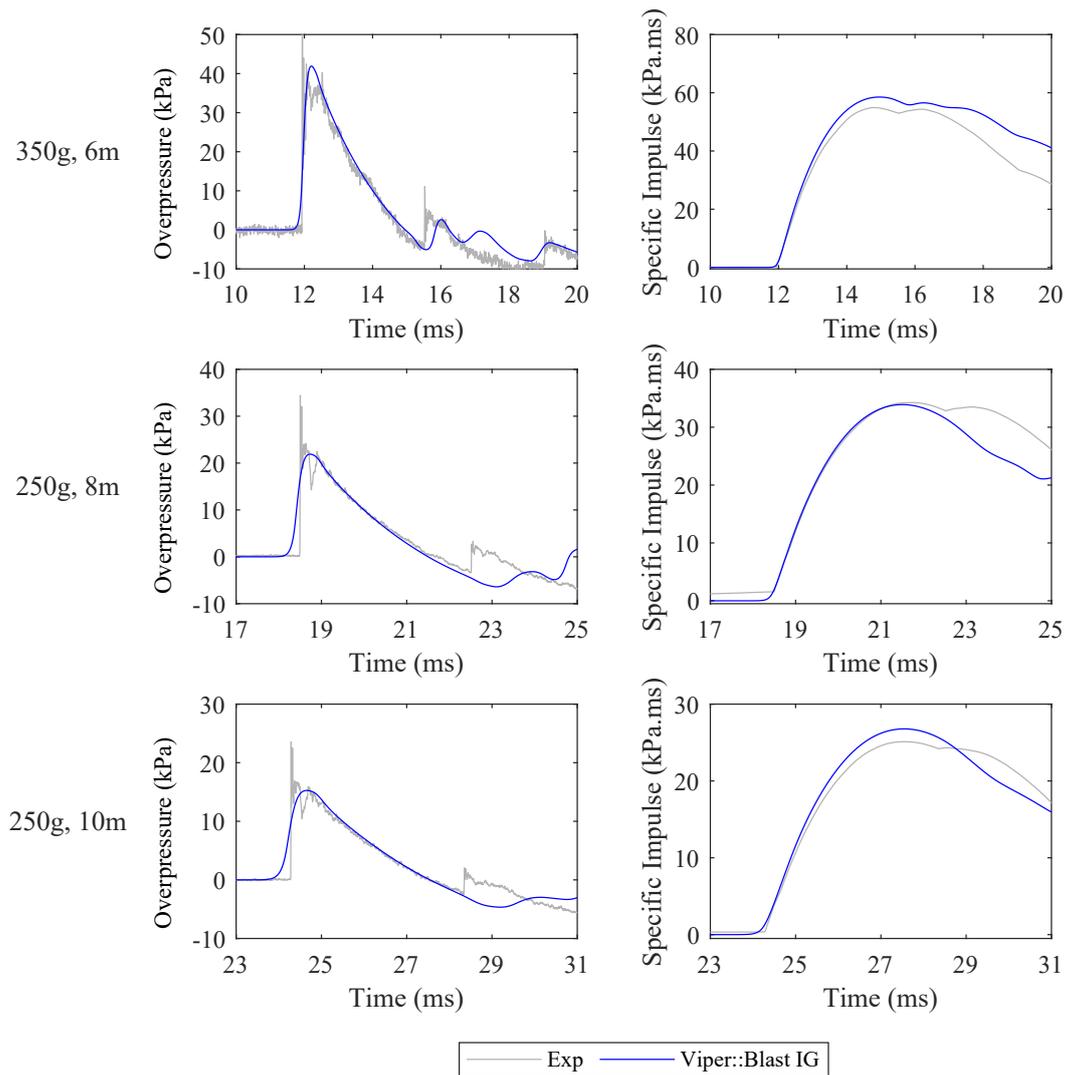


Figure 3.6: Overpressure–time histories for 350 g/6 m, 250 g/8 m and 250 g/10 m, PE4 hemispherical charge size/stand-off pairings. Viper arrival times matched to experiments.

Agreement is fairly good for all stand-off distances with suitably shaped traces and peak values, particularly for specific impulse and for when the noise present in the initial rise of the experimental traces is suitably ignored. However, as expected, when the simulation progresses to gauges positioned at larger scaled distances ( $> 8 \text{ m/kg}^{1/3}$ ), the pressure traces begin to display increased amounts of rounding in the overpressure peak. Further reductions in the cell size would help to prevent this at the expense of computation time. Additionally, the time of arrival of the Viper traces were matched to the experimental records and the secondary shock is not predicted accurately, however, this is a known drawback of CFD analyses (Rigby & Gitterman 2016).

### 3.2.8 Summary

In summary, the numerical solver Viper::Blast has been experimentally validated for use in this thesis following mesh sensitivity and mapping scale factor analyses. By ensuring that each model uses a 1D cell size that provides at least 20 cell across the charge radius, and a 3D cell size less than ten times the maximum permissible 1D cell size, the peak overpressure and specific impulse of each simulation will be representative of the physical problem.

## 3.3 LS-DYNA for blast analyses

### 3.3.1 Introduction

LS-DYNA is a Finite Element (FE) solver that has been applied to a wide range of problems related to material failure and fluid-structure interaction. Its operation is based on the breakdown of a given domain into sections that represent different materials, each with differently assigned mechanical properties and equations of state that dictate the response to a specific stimulus. The materials are discretised by a mesh that creates a connected array of elements upon which governing differential equations are calculated in explicit or implicit time stepped procedures.

Unlike the blast focussed solver, Viper::Blast, wave propagation can be modelled using an explicit Arbitrary Lagrangian-Eulerian (ALE) method. This formulation combines the Lagrangian and Eulerian approaches to defining elements, where the former provides a mesh that is fixed to the material to allow for distortion, but not the passing of any material between elements. Whereas, the latter fixes the mesh in space, allowing flow of material through the mesh without distortion.

With both meshes applied to a given domain, mapping functions are used to transfer information between each mesh at each time step. This allows the deformation caused by the blast wave to be tracked, without excessively large deformations being applied to the Lagrangian elements. Additionally, the ALE method is able to model material interfaces to evaluate multi-material problems, making it suitable for simulating the detonation of an explosive compound in air (Souli et al. 2000).

### 3.3.2 ALE mapping

A key feature included in LS-DYNA is ALE mapping. It allows the user to begin the current ALE simulation using data that is read from the last cycle of a previous ALE run (Livermore Software Technology Company 2015). This applies to Multi-material ALE (MM-ALE) simulations and is commonly used in cases where symmetry enables part of a given simulation can be performed with a different number of dimensions to the remainder of the model.

For example, Aquelet & Souli (2008) explains that in their simulations that evaluate blast wave propagation through water, a 2D model is used to compute the solution until the blast wave's symmetry is no longer respected, at which point the data is mapped to a 3D domain for the simulation to continue. Through making this adjustment, the computation

time decreased from 670s to only 371s when compared to running a model fully in 3D (Aquelet & Souli 2008). Similarly, Lee et al. (2020) utilises a 1D mesh for a symmetric free air blast, but when the ground is contacted the parameter field is mapped to a 2D axisymmetric mesh. Then, as the closest wall is impacted, the data is mapped again to a 3D domain that runs until termination. The obtained results show good agreement with equivalent experimental data to prove that ALE mapping in LS-DYNA is a viable approach for applying new element meshes to domains.

Remapping data can also be used to apply a new mesh to a domain if there is movement of the boundaries in response to the process being simulated. In these cases, the mesh that was initially defined needs to be adapted to fit the new deformed geometry (Menon et al. 2015). This process results in element connectivity and node position alterations, however these changes are not intended to reduce simulation times, instead they focus on making the simulations more accurate to real world experiments where materials can deform in response to external forces.

Throughout this section, ALE mapping is used in LS-DYNA to ensure the detonation of an explosive compound is evaluated with a sufficient number of elements in a reduced domain, before the data is mapped into the full domain with a resolution that benefits computation time. Experimental validation will indicate the element sizes of each mesh to prevent a loss in solution accuracy.

### 3.3.3 Experimental validation

As discussed in Section 3.1, LS-DYNA is not the primary solver used in this thesis. Experimental validation is therefore only provided for a two-dimensional scenario that replicates the models that are evaluated in Chapter 4. This is because the 2D domains that feature in that chapter required additional obstructions to be created and Viper does not support this.

Table 3.3: LS-DYNA keyword values.

<b>*MAT_NULL</b>			<b>*MAT_HIGH_EXPLOSIVE_BURN</b>		
Parameter	Value	Unit	Parameter	Value	Unit
$\rho$	1.225	kg/m <sup>3</sup>	$\rho$	1601	kg/m <sup>3</sup>
			$D$	8193	m/s
			$P_{CJ}$	28.00E9	Pa
<b>*EOS_LINEAR_POLYNOMIAL</b>			<b>*EOS_JWL</b>		
Parameter	Value	Unit	Parameter	Value	Unit
$C_0$	0.0	Pa	$A$	609.77E9	Pa
$C_1$	0.0	Pa	$B$	12.95E9	Pa
$C_2$	0.0	Pa	$R_1$	4.50	-
$C_3$	0.0	Pa	$R_2$	1.40	-
$C_4$	0.4	-	$\omega$	0.25	-
$C_5$	0.4	-	$E_0$	9.00E9	Pa
$C_6$	0.0	-			
$E_0$	253.40E3	Pa			

The experimental results first published in Rigby et al. (2018) are used to validate the detonation of a spherical 100 g PE4 charge placed 80 mm from the rigid plate using LS-DYNA's MM-ALE solver. This allows the solver to simulate explosive scenarios, with each cell in the domain containing air and high explosive without either material needing to coincide with the cell boundaries (Peery & Carroll 2000). Table 3.3 provides the values that are assigned to each of the relevant material model and equation of state parameters.

It is essential for the progression of the blast wave that the initial energy is preserved throughout the detonation process (Lapoujade et al. 2010) and generally over 10 elements across the radius of the charge should be used to ensure sufficient accuracy of the detonation in LS-DYNA (Schwer & Rigby 2018). The use of 1.25 mm square elements provides  $\sim 20$  elements across the radius of the charge and is deemed acceptable in this instance. Figure 3.7 shows a comparison on the numerical and experimental reflected pressure and specific impulse (cumulative temporal integral of pressure) histories at the centre of the target panel (0 mm), 50 mm away, and 100 mm away from the centre. Furthermore, Table 3.4 provides the peak values from these plots to verify that the magnitudes and general form of the profiles are in reasonable agreement, giving confidence in the ability of the numerical approach to adequately simulate the salient physical processes.

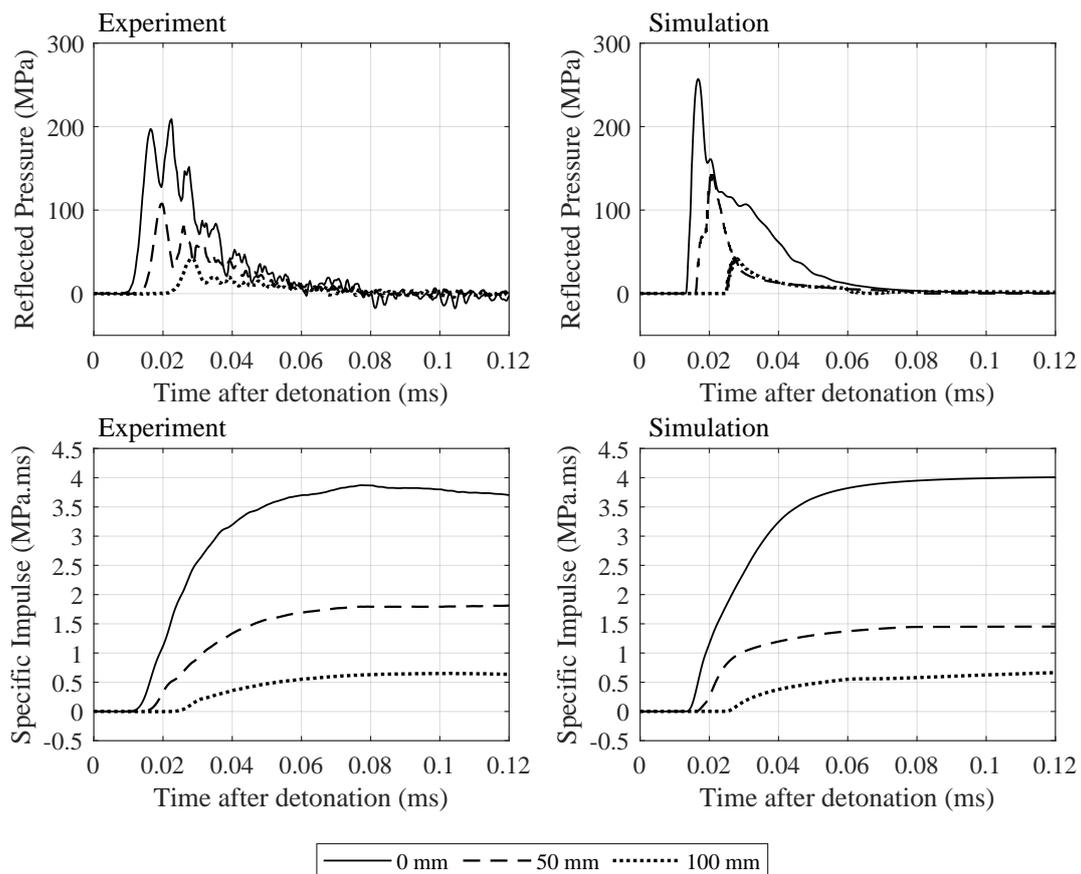


Figure 3.7: Pressure-time and specific impulse-time histories for the detonation of a 100 g PE4 charge placed 80 mm from a rigid plate for three distances from the centre of the plate.

Table 3.4: Peak reflected pressure and specific impulse values from the experiment and simulation profiles shown in Figure 3.7.

Distance from plate centre (mm)	Peak reflected pressure (MPa)		Peak specific impulse (MPa.ms)	
	Experiment	Simulation	Experiment	Simulation
0	209.08	256.92	3.87	4.01
50	107.78	146.73	1.96	1.45
100	42.03	43.22	0.65	0.66

The discrepancies between experimental measurements and numerical output have been seen previously Rigby et al. (2018), and are likely attributed to the strong directionality, high pressure/velocity gradients and magnitudes of the near-field blast. Whilst they appear significant when viewing discrete pressure and impulse histories, they are less significant for target response considerations due to the relatively low areas over which they are acting (compared to the pressure histories at 100 mm from the centre which demonstrate much better agreement).

### 3.3.4 Summary

In summary, the solution of a problem is highly dependant on the resolution of the mesh, the quality of the material models being applied and the contact mechanisms that are allowed between each element. This section therefore validated LS-DYNA as a means of evaluating the propagation of a blast wave through 2D domains where rigid obstacles are present. Use of 1.25 mm square elements and an MM-ALE solving process is shown to provide results that match to previously published work, ensuring that its use in this thesis is justified.

# Chapter 4

## Informed data mapping to reduce computation times

### 4.1 Introduction

As discussed in Chapter 2, it is clear that there is a need for rapid analysis tools that can be used to model environments featuring various obstacles when subjected to the detonation of a high explosive. Machine learning methods present a means of achieving this, however, the computational expense associated to data collection and processing limits their accessibility and ease of development.

This chapter therefore presents ‘The Branching Algorithm’; an algorithm for informed data mapping between numerical models that feature similar initial conditions. The approach is introduced in this chapter alongside a proof of concept, before further developments and applications are shown in the following chapters.

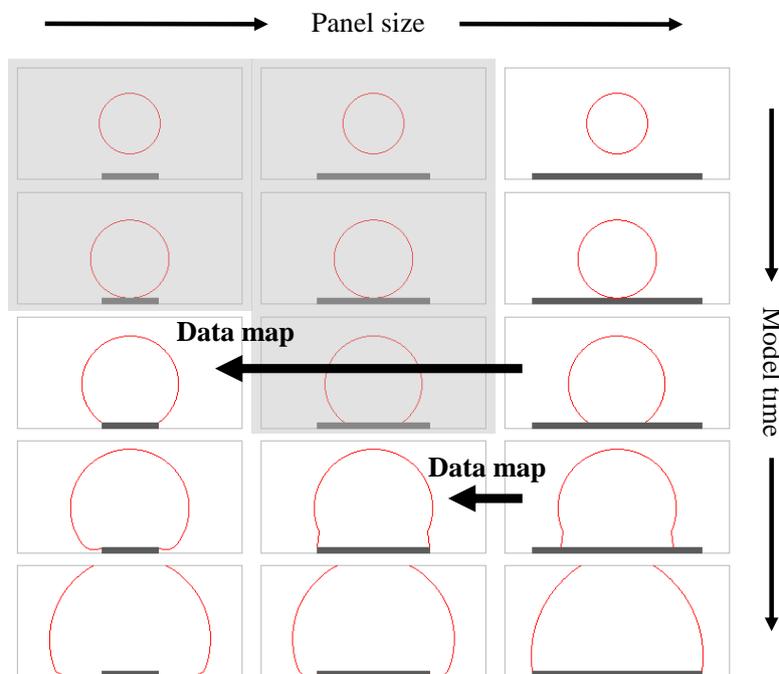


Figure 4.1: Initial idea for a method that saves computation time with removed simulation steps shaded in grey. Blast waves shown in red, rigid panel in dark grey.

## 4.2 The Branching Algorithm

### 4.2.1 Concept

The Branching Algorithm (BA) originated from an initial sketch, similar to Figure 4.1, showing that if the same sized explosive material is detonated using an identical method in varied domains, multiple models will feature identical parameter fields (relative to the position of the charge) up to the time when the respective blast waves impact a unique point in any given domain. In this two-dimensional example, the panel width varies between ‘small’ and ‘large’, meaning before the blast wave reaches the point where it would be at the edge of the small panel, all three parameter spaces are identical. The majority of each field remains ambient, and the blast wave brings a change in pressure that leads to various other parameter alterations.

This duplication of results, and therefore duplication of calculation steps, means that modelling in this manner is inefficient in terms of computation time. The BA is therefore designed to identify when deviations in the parameter field would occur prior to any numerical simulation. Thus enabling informed data mapping that can be used to remove the repeated calculation steps.

### 4.2.2 Inspiration

#### Code reuse and programming

The idea of removing unnecessarily repeated steps is common in computer programming and software engineering projects and is often concerned with code that was developed for one system (source) being reapplied to another (target) (Feitosa et al. 2020). This aims to reduce the number of person-hours required to successfully implement a system, and since the source is known to function correctly, the chance of encountering an error is reduced. In essence, this idea relates to avoiding repeated steps within the development process as there may be existing solutions for a problem that can be reapplied for the new challenges that are faced.

A similar ideology is applied to writing code with the term DRY (do not repeat yourself) programming relating to where a programmer aims to avoid writing code in more places of the project than is necessary. This term was formally defined in a book by Hunt & Thomas (1999), but is now commonly seen as a ‘best practice principle’ for researchers aiming to develop programming skills (Wilson et al. 2014). Various programming software such as Python and MATLAB allow users to repeatedly call back to a set of instructions that have been defined in one place to abide by this rule with the use of ‘functions’ (van Rossum 1995, MathWorks Inc 2021). This removes the need for the same block of code to be copied throughout the entire project with the key benefit being that a change to the process included in the function only needs to be coded once, not at every instance in the overall script. Furthermore, this provides added flexibility to the program due to how the output of a given function can be edited easily with the changes then being implemented at every location. This is where the connection to time dependant modelling becomes apparent as this gives a foundation to the idea that data can be shared by various processes. Despite these ideas not featuring in many research papers as a result of them

being considered as standard practice, the idea of removing repeated steps is not being utilised outside of computer science and programming despite the potential benefits to research and design efficiency.

### Computer processing and parallel simulation

Running multiple processes in parallel to save computation time is something that has been explored in computer science for a number of decades. Historically when a computer was used to execute an operation, a single processor would deal with the request and its clock speed would indicate how many processing cycles would be performed per second. A higher clock speed therefore indicates better performance as a greater number of basic instructions could be executed. With only a single processing unit, tasks could not be performed simultaneously, instead cycles of the processor would be distributed to make it appear like multiple tasks progressed at the same time (Alessandrini 2015).

More recent developments in this field saw the introduction of multi-core processors. These enable tasks to be split up on multiple processing units, or cores, within one processor slot on the computer motherboard, thus allowing for entire tasks to be performed simultaneously. Despite advancements in transistor technology also allowing commercially available processor clock speeds to increase from around 900 MHz (900 million cycles per second) to upwards of 5 GHz (5000 million cycles per second) in the last 20 years, there are still scenarios where the power of a single processor, with multiple cores, is not enough to complete certain tasks. To overcome this challenge, multiple processors can also be used in a network that enables the joint processing power to be accessed provided that the specific problem is broken down into smaller executable tasks. One of the world's most powerful supercomputers, Summit, utilises this approach with additional graphics processing units (GPUs) acting as accelerators to achieve a theoretical peak processing capability of 200 quadrillion calculations per second (Hanson 2020).

When considering modelling applications of parallel computing, many are not concerned with running a framework of tests at the same time. Instead, they aim to simulate highly complex scenarios with multiple cores acting as one solver (e.g. Wu & Tseng 2005, Meng & Berzins 2013, Bruneau & Khadra 2016). A common benefit of this being that greater model fidelity can be achieved as a more refined element mesh can be simulated without impractical computational expense. Furthermore, tasks that were previously too complex to solve could be processed provided that a sufficient number of cores were connected with an effective means of communication. The approach is therefore very useful for specialist applications such as multi-scale material modelling, where the detailed plastic response of a material is simulated on a fine-scale model and used to inform a coarse-scale finite element material model that runs at the same time (Barton et al. 2011). Alternatively, in gas particle tracking, individual particles and collisions can be modelled throughout a domain that is split into sections for each processor to compute separately until there is a crossing of a boundary (Furlani & Lordi 1990).

The ideas presented here concerning the development of parallel computing directly inspired the algorithm and approach proposed in this thesis. In particular, the method followed by Bruneau & Khadra (2016) features sub-domains that are calculated in a progressive manner, with a given row of cells waiting to be initiated by data that is passed from the neighbouring cell being calculated by a separate core. In order to remove

unnecessary duplicated steps within a framework of models, each model can be thought of as a section of the overall domain in this example. Now, for the framework of tests, a single model could begin being simulated on one processor/core until a condition is met where another model would no longer share the same output. At this point data can be communicated to another processor for that simulation to begin at the point where duplicate steps do not need to be simulated again. Instead of allowing for a continuous dialogue between processors as the examples noted here require, now, each one simply waits for the model to be passed on to it, and it then runs independently of the others until the termination time is reached.

### 4.2.3 Principles and terminology

Time-varying numerical forward models can be used to estimate a parameter field,  $\theta$ , given initial conditions,  $\Omega$ , in some domain,  $\Delta$ . The parameter field could, for example, include pressure, temperature and stress at each specified point and it could be used to assess the impact of an explosive, development of a crack, or variation in any relevant quantity. With the aim of the proposed algorithm being to avoid duplicate calculation steps in a batch simulation process, consideration of how the parameter field,  $\theta$ , changes with respect to space and time in each given domain is required. The calculated parameter field at a location given by  $x$ ,  $y$  and  $z$ , at time,  $t$ , is therefore given by Equation 4.1, with  $f(\cdot)$  representing the given numerical model.

$$\theta_t(x, y, z) = f(x, y, z, \Omega, \Delta, t, \theta_{t-1}). \quad (4.1)$$

Provided that multiple models are compatible for data mapping according to a user defined subset of the input conditions,  $\chi$ , it is possible for the Branching Algorithm (BA) to define a ‘trunk model’, given by  $\Delta^T$ ,  $\Omega^T$  and  $\theta^T$ , that corresponds to the test arrangement that simulates a developing parameter field that is common to all other models relative to a specific domain location up to a certain time step.

The specific time step when the trunk model and any other given model (or ‘branch’ of models) from the batch of tests stops producing identical parameter fields can be defined as the ‘deviation point’ for that model. This is given by Equation 4.2, with  $\theta_t^T(x, y, z)_r$  representing the trunk model’s parameter field at time step,  $t$ , in a position given by the relative  $x$ ,  $y$  and  $z$  coordinates. Similarly,  $\theta_t^n(x, y, z)_r$  denotes the parameter field at the same relative location and time step, in model  $n$  of the batch.

$$\theta_t^n(x, y, z)_r \neq \theta_t^T(x, y, z)_r \quad (4.2)$$

The relative position in the domain is given by Equation 4.3:

$$(x, y, z)_r = (X, Y, Z)_r - (x, y, z) \quad (4.3)$$

with,

$(x, y, z)$  = Position of the point in model  $n$ ’s domain

$(X, Y, Z)_r$  = Relative point that is common between both models,  $n$  and  $T$

The comparison between the trunk model's state,  $\theta_t^T$ , and any other model at the same time step,  $\theta_t^n$ , is only possible if the relative domain location is shared as this removes the influence of varying domain shapes and sizes. Comparing multiple domains that are modelling an explosion requires the shared location to be at the detonation point of the explosive as this is the initiation point that causes changes to the parameter field in the surrounding area. However, this approach could be applied to other fields of study where differing relative indicators would be required.

Further, a deviation point corresponds to the first time step where the model's calculated parameter field would be different from the trunk model. Mapping the data associated to  $\theta_{t-1}^T$  to the new domain for use with its own input and domain conditions,  $\Delta^n$  and  $\Omega^n$ , with  $(X, Y, Z)_r$  acting as the origin, therefore enables simulation steps in model  $n$  from  $t = 0$  to the current mapping time to be saved.

The BA determines the deviation point with respect to the spatial location,  $(x, y, z)$ , for each model prior to any numerical simulation by assessing the 'influences' present in each domain. These influences are defined as the factors that change the input conditions into the observed outputs, meaning they are bespoke to each potential application of the algorithm. However they will be defined by a subset of the inputs to the numerical calculation:

$$\lambda_i^n \subset [(x, y, z)_r, \Omega^n] \quad (4.4)$$

where  $i$  is the influence number. Example influences include specific boundary constraints, time-dependant calculation variables, or possibly entire calculation methods that are introduced after a threshold value is reached.

In order for the algorithm to discern when each influence is reached in each respective model, a comparison metric,  $\gamma$ , is defined for all entries. The time step associated with when each potential deviation is reached in the numerical model may not be known prior to the simulation of each domain, however, an expected sequence of when each point is met can be defined with consideration of additional parameters. Again, for the example of modelling an explosion, the distance/time that the blast wave will travel before it will reach each influence can be used to define the order of influence activation. Therefore allowing the algorithm to identify if there are any deviating simulations.

The selected influence that correlates to the deviation point of a given model is provided as the output to the algorithm in an ordered vector denoted by  $\epsilon$ . Here,  $\epsilon$  provides the deviation condition and the associated model number in a list that is sorted according to when each entry is reached in the trunk model's simulation. For example:

$$\epsilon = \begin{bmatrix} 1 & \lambda_3^1 \\ 4 & \lambda_2^4 \\ \dots & \dots \end{bmatrix} \quad (4.5)$$

This shows that model 1 is the first to deviate from the trunk, due to the influence defined by  $\lambda_3^1$ . Model 4 follows this due to a deviation at its second influence,  $\lambda_2^4$ .

As the parameter field of the trunk model,  $\theta^T$ , undergoes a change that is not expected to be present in the model given by  $\epsilon_1(n)$ , the latter should be initiated by data that is

mapped from the last time step of the trunk model. The deviation condition given by  $\epsilon_1(\lambda)$  is used to identify when this will occur. Once mapping to this domain has taken place, the trunk model simulation continues until the condition defined by  $\epsilon_2(\lambda)$  is met, and another phase of data mapping can take place. This process continues until all compatible models have been initialised as shown in Algorithm 1 provided in Section 4.2.4.

#### 4.2.4 The generalised Branching Algorithm

##### Summary of variables

$i$	influence entry
$n$	model number
$T$	trunk model number
$t$	time step
$\beta$	greatest unique initial influence
$\gamma$	comparison measure for when an influence becomes active
$\Delta$	model domain
$\epsilon$	deviation conditions
$\theta$	parameter field
$\Lambda$	influence table
$\lambda$	influence
$\chi$	subset of inputs
$\Omega$	inputs / initial conditions

---

##### Algorithm 1: The Branching Algorithm.

---

```

1  $\chi$  = required compatibility check inputs;
2 n = number of models;
3 for 1 : n do
4    $\Omega^n$  = model n inputs ;
   // Check if the model meets the compatibility requirements.
5    $\chi^n \subset \Omega^n$ ;
6   if  $\chi^n \neq \chi$  then
7     | Model is not compatible, do nothing;
8   else
9     // Store compatible model numbers.
    compatible.append(n);
10    // Create influence list for model n.
     $\lambda^n \subset [(x, y, z)_r, \Omega^n]$  ;
    // Create influence table for model n where  $\gamma^n$  is the comparison metric for
    each influence entry,  $i$ .
11     $\Lambda_n = [n, \lambda_i^n, \gamma_i^n]$  ;
    // Sort influence table according to the comparison values, with the earliest
    occurring entry being listed first.
12    sort( $\Lambda_n, \Lambda_n(\gamma)$ ) ;
13  end
14 end

```

---

---

```

// Define an overall influence table including all compatible model influence tables.
15  $\Lambda = [\Lambda^{\text{compatible}(1)} ; \dots ; \Lambda^{\text{compatible}(\text{end})}] ;$ 
// Remove influence entry if it is repeated in every compatible model.
16  $\Lambda(\Lambda[\lambda, \gamma].\text{count}(\Lambda_i[\lambda, \gamma]) = \text{count}(\text{compatible})) = [] ;$ 
// Sort the influences according to the comparison value, with the earliest occurring
// entry being listed first.
17  $\text{sort}(\Lambda, \Lambda(\gamma)) ;$ 
// Initialise the greatest unique initial influence used to identify the trunk model.
18  $\beta = 0$ 
// Compare the unique initial influence of each compatible model to the existing
// greatest unique initial influence,  $\beta$ , in order to find the trunk model,  $T$ .
19 for  $n = \text{compatible}(1) : \text{compatible}(\text{end})$  do
20    $\Lambda^{\text{n,temp}} = \Lambda^n ;$ 
21    $\Lambda^{\text{T,temp}} = \Lambda^T ;$ 
// Temporarily remove influences present in both the trunk model and the comparison
// model so that only unique influences are considered.
22    $\Lambda^{\text{n,temp}}(\Lambda^{\text{T,temp}}[\lambda, \gamma].\text{count}(\Lambda^{\text{n,temp}}_i[\lambda, \gamma]) > 0) = [] ;$ 
23    $\Lambda^{\text{T,temp}}(\Lambda^{\text{n,temp}}[\lambda, \gamma].\text{count}(\Lambda^{\text{T,temp}}_i[\lambda, \gamma]) > 0) = [] ;$ 
24    $i = 1 ;$ 
25   if  $\Lambda^{\text{n,temp}}_1(\gamma) > \beta$  then
|   // If the unique initial influence of the comparison model occurs at a later
|   // stage than existing greatest unique initial influence, refine the latter and
|   // set the new trunk model.
26   |    $\beta = \Lambda^{\text{n,temp}}_1(\gamma) ;$ 
27   |    $T = n ;$ 
28   else if  $\Lambda^{\text{n,temp}}_1(\gamma) = \beta$  then
29   |   while  $\Lambda^{\text{n,temp}}_i(\gamma) = \Lambda^{\text{T,temp}}_i(\gamma)$  do
|   |   // If unique initial influences occur at the same step, compare subsequent
|   |   // entries until there is a difference or until there are no remaining
|   |   // entries to compare.
30   |   |    $i = i + 1 ;$ 
31   |   |   if  $\Lambda^{\text{n,temp}}_i(\gamma) = \text{undefined}$  then
32   |   |   |    $\Lambda^{\text{n,temp}}_i(\gamma) = 0 ;$ 
33   |   |   if  $\Lambda^{\text{T,temp}}_i(\gamma) = \text{undefined}$  then
34   |   |   |    $\Lambda^{\text{T,temp}}_i(\gamma) = 0 ;$ 
35   |   |   if  $\Lambda^{\text{T,temp}}_i(\gamma) = 0$  and  $\Lambda^{\text{n,temp}}_i(\gamma) = 0$  then
36   |   |   |    $\Lambda^{\text{T,temp}}_i(\gamma) = 1 ;$  // Ensures the trunk model is not updated.
37   |   |   if  $\Lambda^{\text{n,temp}}_i(\gamma) > \Lambda^{\text{T,temp}}_i(\gamma)$  then
|   |   |   // Update the trunk model and greatest unique initial influence.
38   |   |   |    $\beta = \Lambda^{\text{n,temp}}_i(\gamma) ;$ 
39   |   |   |    $T = n ;$ 
40   |   end
41 end

```

---

---



---

```

// Remove all trunk model entries and all entries that are common between the trunk
// model and remaining models in the overall influence table.
42  $\Lambda(\Lambda = \Lambda_i^T) = []$ ;
43  $\Lambda(\Lambda[\lambda, \gamma] = \Lambda_i^T[\lambda, \gamma]) = []$ ;
44  $j = 0$ ;
45 while  $\Lambda$  is not empty do
46    $j = j + 1$ ;
47   // Store the deviation point for model  $\Lambda_1(n)$ 
    $\epsilon_{j,1} = [\Lambda_1(n), \Lambda_1(\lambda)]$ ;
   // Check if multiple models deviate at the same time since the influence causing
   // deviation is equal in back-to-back influence table entries.
48    $k = 2$ ;
49   while  $\Lambda_k(\lambda) = \Lambda_1(\lambda)$  do
50     // Multiple models deviate at the same time. Store the deviation point for
     // model  $\Lambda_k(n)$ .
      $\epsilon_{j,k} = [\Lambda_k(n), \Lambda_k(\lambda_n)]$ ;
     // Remove remaining influences associated with this additional deviating model.
51      $\Lambda(\Lambda(n) = \Lambda_k(n)) = []$ ;
52      $k = k + 1$ ;
53     Submodel = 1;
54   end
   // Remove remaining influences associated with the deviating model.
55    $\Lambda(\Lambda(n) = \Lambda_1(n)) = []$ ;
56 end
// Deviation conditions for all models are given by  $\epsilon$ , with any entry including
// multiple columns requiring a sub-trunk model to be defined for additional time
// saving.
57 if Submodel = 1 then
58   // Multiple models branched from the trunk model at the same point, further time
   // saving may be possible.
   Repeat the Branching Algorithm for each set of models that branched off from
   the trunk model at the same time ;
59 else
60   Multiple models do not branch from the trunk model at the same point, process
   ends;
61 end
Output: Deviation points for each model,  $\epsilon$ . Trunk model number, T. If required:
sub-trunk model numbers, T'.

```

---

### 4.2.5 Context and potential applications

This section aims to present a non-exhaustive series of examples where a method aiming to remove duplicated simulation steps could be utilised outside of blast and protection engineering. The first is related to vehicle design and safety assessments. Here, the simulation of multiple crashes is required to assess a range of parameters and test conditions including collision speed, vehicle materials and impacted object (e.g. Lu et al. 2020, Wu et al. 2020). More recently, numerical models are being used to develop automated driving systems that can detect when crashes are likely to occur, thus giving the driver a warning

to avoid a collision (Varnavsky & Rogulina 2020). The results from the analysis models are used to inform the system's parameters and there is the requirement for many scenarios to be modelled in order to form a data set that provides a varied view of potential conditions on the roads. In a study by Long et al. (2018), the authors simulate “*59 crash scenarios, involving 5 typical roadside obstacles, 2 types of guardrails, 15 embankment slopes, and 3 types of vehicles*”. Whilst the specific simulation times are not noted in the article, it is clear that the duplicated vehicle response, for crashes that each begin in a similar manner, could be removed to make the modelling process more efficient. In particular, the analysis of various embankment heights at a range of angles provides a good chance for the removal of duplicated simulation steps as the outcome will be identical for a larger slope height and a smaller one up until the point at which the vehicle reaches the top of the smaller slope and the physical systems diverge.

Similarly, search and rescue tools that aim to identify the region in the ocean where a missing person or object is located often require the simulation of a large number of potential flow paths (e.g. Coppini et al. 2016). By using information such as the last known location, time, and wind speed, the modelling process can incorporate a Monte Carlo (MC) analysis whereby the influencing parameters are perturbed in a trajectory model that generates an ensemble of outcomes (Breivik & Allen 2008). Collating the results provides rescue teams with a focused region where it is likely for the person/object to be found. This approach lends itself well to the removal of duplicate steps as some flow paths may remain consistent with each other for a large number of calculations. A more efficient analysis could therefore be undertaken if these trajectories are grouped until the randomly selected conditions cause the flows to diverge. This, in turn, benefits consideration of uncertainty of the flow parameters being incorporated in the solutions of all potential object starting locations.

Next, the failure of pressurised water mains leading to rapid cratering that may result in local flooding and injury from debris has recently been studied by Barr et al. (2020). Here, crater size was evaluated as a function of pipe diameter, internal pressure, air content, and burial depth using finite element analysis in order to develop a predictive tool. Clearly, for a batch of models where only burial depth varies, the models will be identical until the instant where the soil surface is broken by expansion of the water. Hence, this type of study would be well suited to a modelling approach aimed at removing duplicate calculation steps.

Further applications in engineering could include fluid-structure interaction or wave overtopping of flood defences (Mehreganian et al. 2018, Chen et al. 2021). Nevertheless, considering each potential topic discussed in this section, it is clear that the conclusions formed from the analyses would be improved by obtaining more data or by completing the process in a more efficient way. This in turn enables a greater understanding of the uncertainty associated with the test scenarios to be analysed.

#### 4.2.6 The Branching Algorithm for blast analysis

Adapting the BA methodology, introduced in Section 4.2.4, for use with blast analyses requires the key terminology to be assigned to blast relevant parameters. Notably, a deviation point is provided in terms of its spatial location relative to the charge, since the air that fills the domain will remain at ambient conditions until the blast wave emanating

from the explosive material reaches that particular point in the associated domain. The solutions to each model will therefore diverge at the moment that the pressure wave reaches a location or material in a given domain that is not present in the other models. The influences used to identify these locations can then be associated to obstacle or boundary surfaces, or changing material properties.

For the BA to allow for data to be mapped between various models it is essential for some of the input conditions to be identical. For blast analyses this relates to the ambient conditions of each domain and the charge shape and composition. If there is a difference between two models because of these variables, the simulation would be different from birth and so no mapping would be possible. The algorithm splits models with differing conditions into various trees that are sorted and analysed separately. However it should be noted that the size of the charges used in each model will not necessarily be a reason for defining a new deviation tree as domain scaling can be used according to Hopkinson-Cranz scaling laws (Hopkinson 1915, Cranz 1926). Use of this approach improves the compatibility of models because each domain in the batch of tests can be defined in terms of scaled distance, with a single charge size relating all arrangements. For the examples given by this study, scaling was not required.

A simplified version of the blast BA is provided below to show how the comparison of each domain allows for informed data mapping to take place.

1. Identify unique geometries and create discrete influences associated to the obstacles and boundaries for each model.
2. Identify the groups of models (deviation trees) that can share numerical data with no loss of modelling accuracy by considering the probabilistic inputs being used (e.g. separate groups are required for differing charge shapes and/or ambient conditions as data mapping cannot occur if difference are present from birth of the simulations).
3. For each tree:
  - (a) Calculate the wave travel distance to each influence of each model so that comparisons between influences can be made.
  - (b) Compile all influences and identify which model has the unique influence that is the furthest distance from its charge. This model acts as the ‘trunk model’, with all other models deviating from its solution.
  - (c) Remove influences that are common between the trunk model and any others. Sort the remaining influences from all models so that the first entry is the one that is closest to its respective charge.
  - (d) Take the first entry as the first deviation from the trunk model and remove any other influences associated to this model from the remaining influence store. Continue this process until all models have deviated.
  - (e) If multiple models share the same deviation at any point, a ‘sub-trunk model’ can be defined by repeating the process with this sub-set of models to allow for additional savings and multiple informed data mappings.

Ultimately, with the model geometries, boundary and ambient conditions and charge characteristics being defined as the input values, this process automatically produces a modelling order for a batch of tests. Through generating a number of deviation locations based on the blast wave's progression through the domain of a parent model (trunk or sub-trunk), the chosen numerical solver can be monitored so that data maps are created just before the trunk model's parameter field diverges from that of a branching model. The associated branching models can then be initialised with the exported data maps, removing the need for repeated calculations to be computed, specifically around the charge before the blast wave impacts any boundaries or surfaces.

An example of a deviation tree being output from the algorithm is provided by Figure 4.2 with model 'n' being assigned as the trunk model and various branches showing where data mapping can be utilised. It is also shown how model 'n-1' is defined as a sub-trunk model, meaning models 'n-1', 'n-3' and 'n-4' were all identified to deviate from model 'n' at the same influence. Additional repeated simulation steps were then identified in a second pass through the BA when considering only these three models. The recursion capability of the algorithm is therefore essential in allowing for the greatest computational time saving.

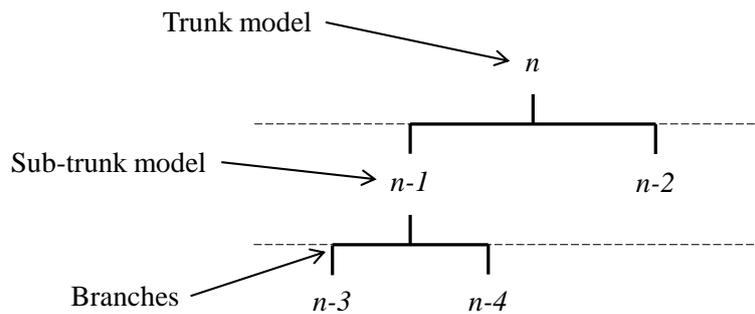


Figure 4.2: Example deviation tree obtained using the Branching Algorithm.

After application of the BA, a tool capable of performing analysis of the branched network of models is required. It is important to ensure that there is no/limited detrimental impact to the progressive accuracy of the solutions as data is transferred from one model to another. It is not expected that this would be a problem for many modelling techniques as studies adopting methods such as domain decomposition, multigrid solving and mesh adaptation are able to efficiently solve highly complex problems with data being transferred between parallel computing processors many times throughout the simulation (Wu & Tseng 2005, Bruneau & Khadra 2016, Park & Kwon 2005).

This introduction to the BA aims to provide a basic understanding of the logical process being employed by the proposed method, however the following section expands on this for 2D blast simulations with a walk-through of how the influences are identified and compared.

## 4.3 Proof of concept for blast analyses in 2D

### 4.3.1 Problem scenario

To show that the proposed algorithm and simulation strategy can be highly beneficial in saving computation time, a batch of nine 2D models based on the experimental work presented by Rigby et al. (2018) and Rigby et al. (2020b) have been simulated using the proposed method. Simulation times are compared between when the BA is used, and when a standard birth to termination approach is adopted for the same nine models. All other factors remain identical (mesh size, computational resource, etc.).

In the aforementioned experimental work, tests were performed using a 100 g spherical PE4 charge positioned 80 mm and 380 mm from a rigid target. The results were then used to validate the numerical solver LS-DYNA for use in modelling near-field explosions. Expanding upon this framework, Table 4.1 gives additional scenarios that may be explored, with varied charge locations providing three distinct stand-off distances; 80, 230 and 380 mm for three different target blast panels diameters; 200, 300 and 400 mm. As shown in Figure 4.3, the panels are assumed to be rigid with a constant thickness of 50 mm in a domain that features non-reflecting, ambient boundaries (aside from the axis of symmetry that enables these problems to be considered in 2D half-space). The charge size, shape and material also remain consistent in all cases.

Considering how LS-DYNA features ALE mapping, there is a clear capability for data to be transferred from one model to another in the way that the proposed BA requires. This is the step that ultimately enables time saving to be significant for CFD or similar progressive mathematical processes where the scenario can be broken down into a range of steps, separated by influences that are present within the tests.

Table 4.1: Model setup parameters.

Model No.	Panel width (mm)	Charge stand-off (mm)	Charge size (g)	Charge material and shape
1	200	80	100	PE4 sphere
2	300			
3	400			
4	200	230		
5	300			
6	400			
7	200	380		
8	300			
9	400			

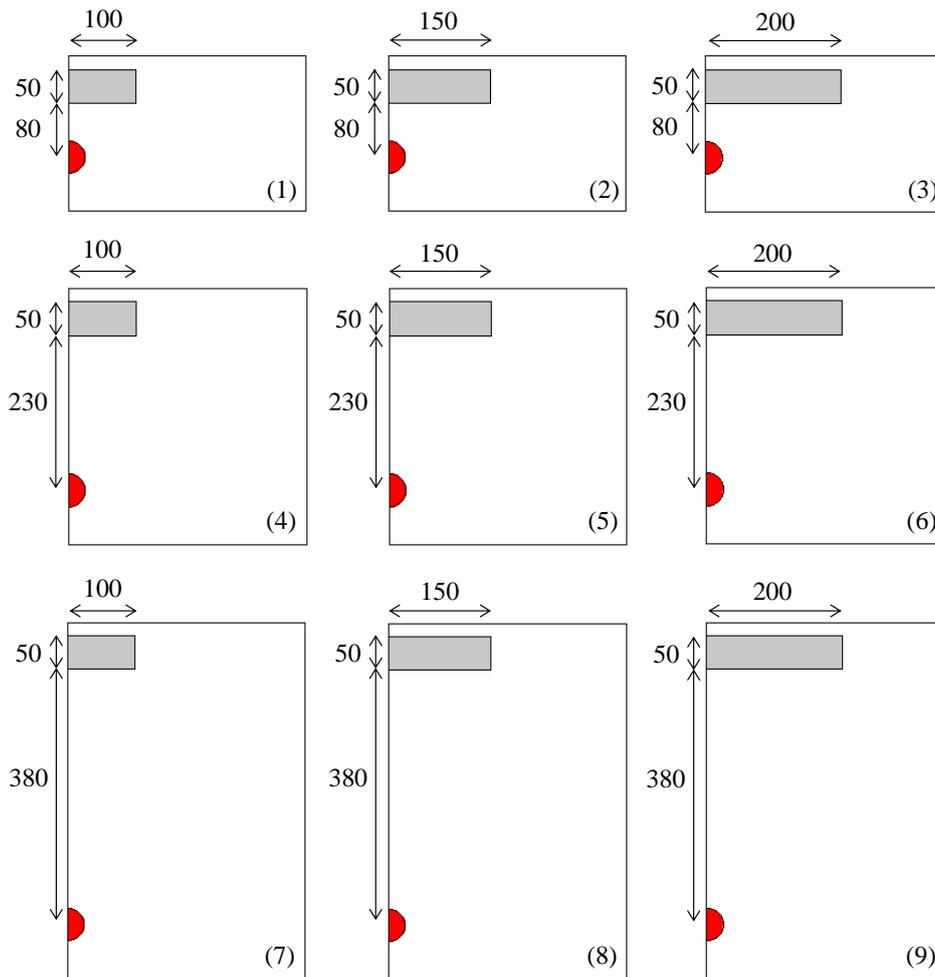


Figure 4.3: Explosive test arrangements to be used with the BA shown in 2D. Charge is given as a red circle and the impacted panel is shown in grey. Boundaries are ambient non-reflecting. Model numbers are provided in the lower right corner of each image. All dimensions are in millimetres.

### 4.3.2 Model specification

Following the successful validation of LS-DYNA with 1.25 mm elements (Section 3.3), each scenario in the example batch of tests given by Table 4.1 and Figure 4.3 is modelled using this mesh density and the same material model and equation of state parameters provided by Table 3.3. They are also modelled in 2D axi-symmetry with the domain boundaries being defined at distances that are sufficiently large so that although reflections will occur as the shock waves impact the boundary nodes, the returning shock waves will not reach the panel before the termination time is met.

As shown in Figure 4.4, models 1, 2 and 3 were simulated in a  $350 \times 230$  mm domain, with the panel positioned with  $y$ -coordinates of 160 mm and 210 mm. Models 4, 5 and 6 were simulated in a  $350 \times 380$  mm domain, with the panel positioned with  $y$ -coordinates of 310 mm and 360 mm. Finally, models 7, 8 and 9 were simulated in a  $350 \times 530$  mm domain, with the panel positioned with  $y$ -coordinates of 460 mm and 510 mm.

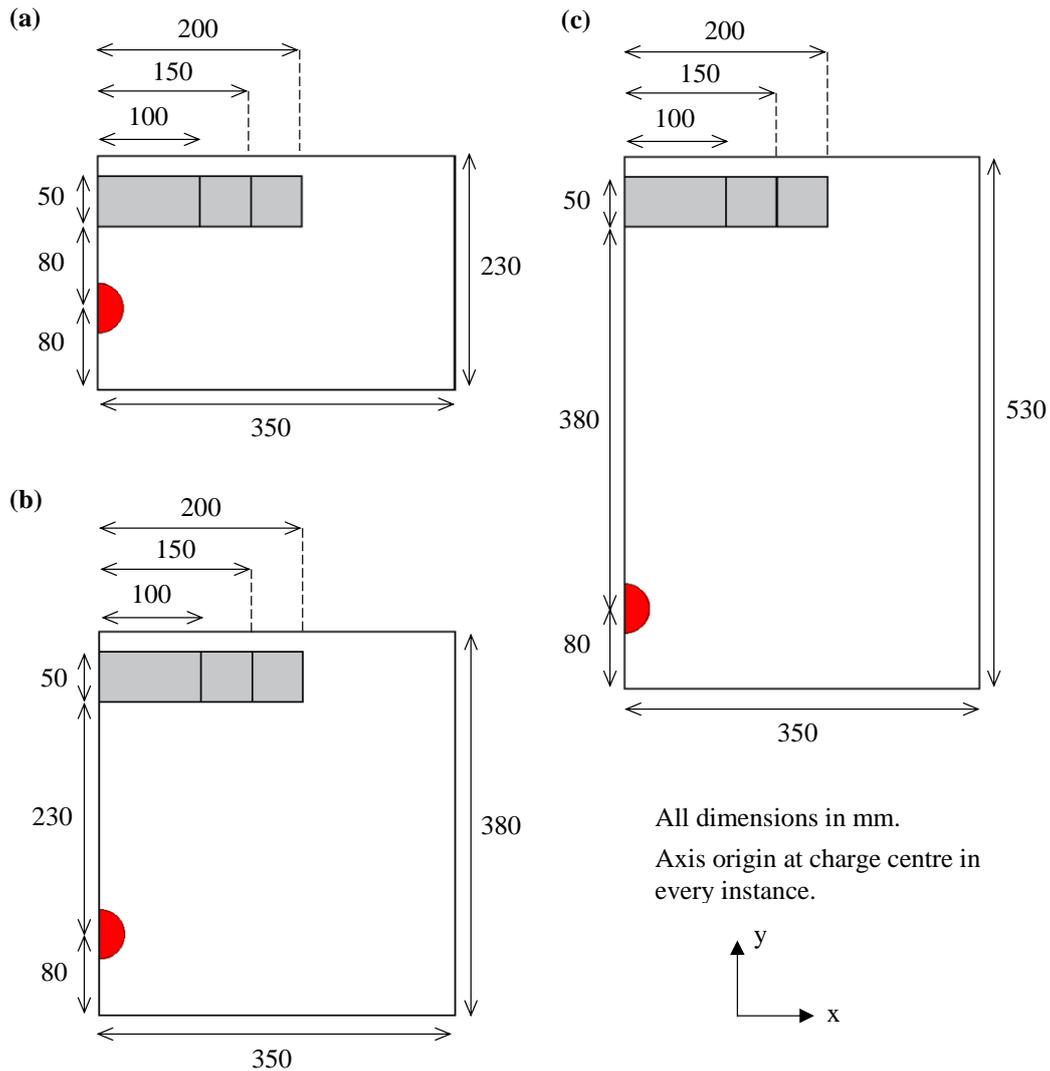


Figure 4.4: Numerical models showing the various stand-off distances, blast panel diameters (grey rectangle) and the 49.2 mm diameter charge (red semicircle). (a) Models 1, 2 and 3 (b) Models 4, 5 and 6 (c) Models 7, 8 and 9.

The termination times for models 1, 2 and 3 were then set at 0.12 ms, models 4, 5 and 6 at 0.2 ms and 7, 8 and 9 at 0.28 ms. This allows for the initial blast wave to fully clear the panel without interference from any unwanted reflections.

In every instance the charge is centred with a  $y$ -coordinate of 80 mm so that the data mapping, achieved using the `*INITIAL_ALE_MAPPING` keyword (Aquelet & Souli 2008), was implemented relative to a consistent point in all domains. Furthermore, since the panel dimensions and its location coincides well with the element mesh of each domain, it is modelled by fully restraining the elements that fall within the respective geometry. This has the effect of creating a rigid reflecting surface and it removes the need to create a new part and material type in the LS-DYNA model.

### 4.3.3 Algorithm walk-through and output

With the model geometries clearly defined, the BA can be used to identify the deviation points where the numerical outputs from each simulation will cease being identical. This section steps through the key stages of the algorithm to show how it systematically selects when each model should be initiated with data mapped from the trunk model.

It is to be noted that this walk-through ignores the complexity that may be required for other applications or in use with more detailed test scenarios. For example, the panels are assumed to be rigid, i.e. any motion of the panel following application of the blast loading occurs on timescales such that the development of loading is itself unaffected. This has been shown to be a reasonable assumption when considering blast loading on structural panels (Rigby et al. 2019). Each model is also being solved with the same method, ambient conditions, charge material/detonation location and panel properties, and therefore it is only the differing domain geometries that will provide diverging solutions. This also allows influences to be stored with their spatial locations and comparison values only, as the material properties and surface slopes will not lead to deviations. In the next example application, the 3D problems require a more comprehensive set of variables to be stored to enable influence comparisons to be made correctly.

Table 4.2: Individual influences for each model being considered are formalised in the influence table below. Rear nodes of the panels are omitted for brevity. Influence defining the trunk model is shown in italics.

Model No.	Influence Location	Relative x (mm)	Relative y (mm)	Distance from charge centre, $\gamma$ (mm, 0 d.p)
1	Panel face	0	80	80
1	Panel vertex	100	80	128
2	Panel face	0	80	80
2	Panel vertex	150	80	170
3	Panel face	0	80	80
3	Panel vertex	200	80	215
4	Panel face	0	230	230
4	Panel vertex	100	230	251
5	Panel face	0	230	230
5	Panel vertex	150	230	275
6	Panel face	0	230	230
6	Panel vertex	200	230	305
7	Panel face	0	380	380
7	Panel vertex	100	380	393
8	Panel face	0	380	380
8	Panel vertex	150	380	409
9	Panel face	0	380	380
<i>9</i>	<i>Panel vertex</i>	<i>200</i>	<i>380</i>	<i>429</i>

By entering the geometry and charge location in Figure 4.3 into the BA, the influences associated to rigid obstacle and boundary vertices and faces can be defined as shown in Table 4.2. Here the coordinates of the key points on the panel relative to the charge are used to define the influences themselves (columns 3 and 4). These are then assigned a comparison value ( $\gamma$ ) that enables the algorithm to sort when each influence will be reached in a numerical simulation. For this example  $\gamma$  is equal to the direct distance between the charge and the identified point (column 5). This value would ideally be assigned to the arrival time of the blast wave at each location, however for simplicity the distance measure is used. It is at this stage where any influences that are present in all models would be removed if required.

Next, the trunk model can be defined by finding the model that encounters its first unique influence at the latest calculation step in its respective domain. This ‘greatest unique initial influence’ corresponds to the test arrangement that will see all other models deviate from its solution before the step where it would have deviated from the other models itself. Thus enabling the largest number of duplicated steps to be removed from each simulation process.

By assessing Table 4.2, it can be seen that model 9 will be the trunk model since its first unique influence corresponds to the blast wave impacting the panel vertices at the greatest distance from the charge; 429 mm. The influences present in this newly-defined trunk model can then be removed from all other model’s tables, provided that they would occur at identical locations within each simulation. For this example, this relates to the panel face impacts for models 7 and 8, since columns 3, 4 and 5 of Table 4.2 contain identical values.

Table 4.3: Combined influence table is formed and sorted according to the time when each entry is encountered by the blast wave. Entries included in the trunk model are removed.

<b>Model No.</b>	<b>Influence Location</b>	<b>Relative x (mm)</b>	<b>Relative y (mm)</b>	<b>Distance from charge centre, <math>\gamma</math> (mm, 0 d.p.)</b>
1	Panel face	0	80	80
2	Panel face	0	80	80
3	Panel face	0	80	80
1	Panel vertex	100	80	128
2	Panel vertex	150	80	170
3	Panel vertex	200	80	215
4	Panel face	0	230	230
5	Panel face	0	230	230
6	Panel face	0	230	230
4	Panel vertex	100	230	251
5	Panel vertex	150	230	275
6	Panel vertex	200	230	305
7	Panel vertex	100	380	393
8	Panel vertex	150	380	409

Following this step, a combined table of influences is formed (Table 4.3) and ordered according to the direct distance from the charge. The first entry therefore corresponds to the first model that deviates from the trunk (i.e. the model that reaches an influence causing the output of the simulation to diverge first). As the algorithm works through the table of remaining influences, any model that deviates from the trunk has its entries removed. The order of when each model deviates can therefore progress to each model in the batch of tests until no entries remain.

For this example, the first deviating model (1) diverges due to an influence that is also present in two other models (2 and 3). It is therefore essential for the algorithm to compare the influence causing model 1 to deviate with the next entries to check if a sub-trunk model can be defined. This allows the algorithm to identify further mapping opportunities that can save additional computation time by removing duplicate steps that occur after the initial mapping from the trunk model.

Removing the influences associated with these three diverging models shows that the second divergence from the trunk model also features three models. Models 4, 5 and 6 are shown to have identical influences for when the blast wave hits the panel at a 230 mm stand-off distance. After these have branched off from the trunk model, models 7 and 8 remain, with their relative deviation conditions then being defined by the blast wave reaching the vertices of the panels that are positioned at the same stand-off distance as in the trunk model.

Since two sub-trunks will be required, the BA utilises recursion and is repeated for the smaller batches of models that deviated from the trunk model at the same point. Table 4.4 shows the relevant remaining influences for models 1, 2 and 3, with model 3 being defined as the sub-trunk model. It is clear to see how model 1 will deviate first, as the blast wave reaches its panel vertex. This occurs before model 2 deviates for the same reason, at its panel vertex that is slightly further from the charge. The process for models 4, 5 and 6 progresses in a similar way to this example, with model 6 acting as the sub-trunk due to it having the widest panel in its configuration. Model 5 then follows model 4 as the next to deviate. It is worth noting that if multiple models within this sub-trunk had an identical plate diameter but, say, different plate thicknesses, then recursion would be used again to sort those models according to the additional influence relating to plate thickness.

Table 4.4: Combined influence table for the first sub-trunk. Model 3 is deemed to be the sub-trunk model.

Model No.	Influence Location	Relative x (mm)	Relative y (mm)	Distance from charge centre, $\gamma$ (mm, 0 d.p.)
1	Panel vertex	100	80	128
2	Panel vertex	150	80	170
3	<i>Panel vertex</i>	<i>200</i>	<i>80</i>	<i>215</i>

According to the sorting process employed by the BA, the output vector for this tree denoted by  $\epsilon$  is given by:

$$\epsilon = [ \text{Model number} \quad \text{Influence causing deviation} ] = [ n \quad \lambda_i^n ] \quad (4.6)$$

Where the influence number,  $i$ , causing deviation for model,  $n$ , is defined as:

$$\lambda_i^n = [ \text{Influence location} \quad \text{Relative x coordinate} \quad \text{Relative y coordinate} ] \quad (4.7)$$

This results in the output for this example being equivalent to:

$$\epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ 7 \quad \text{Panel vertex, 100, 380} \\ 8 \quad \text{Panel vertex, 150, 380} \end{bmatrix} \quad (4.8)$$

with,

$$\epsilon_1 = \begin{bmatrix} 3 \quad \text{Panel face, 0, 80} \\ 1 \quad \text{Panel vertex, 100, 80} \\ 2 \quad \text{Panel vertex, 150, 80} \end{bmatrix} \quad (4.9)$$

$$\epsilon_1 = \begin{bmatrix} 6 \quad \text{Panel face, 0, 230} \\ 4 \quad \text{Panel vertex, 100, 230} \\ 5 \quad \text{Panel vertex, 150, 230} \end{bmatrix} \quad (4.10)$$

Figure 4.5 presents these outputs in the form of a flow chart (a), and deviation tree (b) to show how the trunk model data can be mapped to various domains at the simulation step prior to when there is a change in the ambient conditions at the given locations.

As discussed, the recursion capabilities of the method are highlighted by how  $\epsilon_1$  and  $\epsilon_2$  present the output from additional passes through the BA for the sets of models that deviated from the trunk model at an identical influence. This process could occur any number of times to meet the requirements of the models being simulated, with the possibility of sub-sub-trunk models and beyond being identified as discussed above with regards to additional parameters such as plate thickness. Thus, the level of recursion is intimately linked to the number of unique input variables of a batch of numerical models.

#### 4.3.4 Computation times

With the progression of the models through the algorithm now complete, the order of branching and the corresponding influences causing deviation can be implemented using the relevant solving software to enable an efficient simulation process.

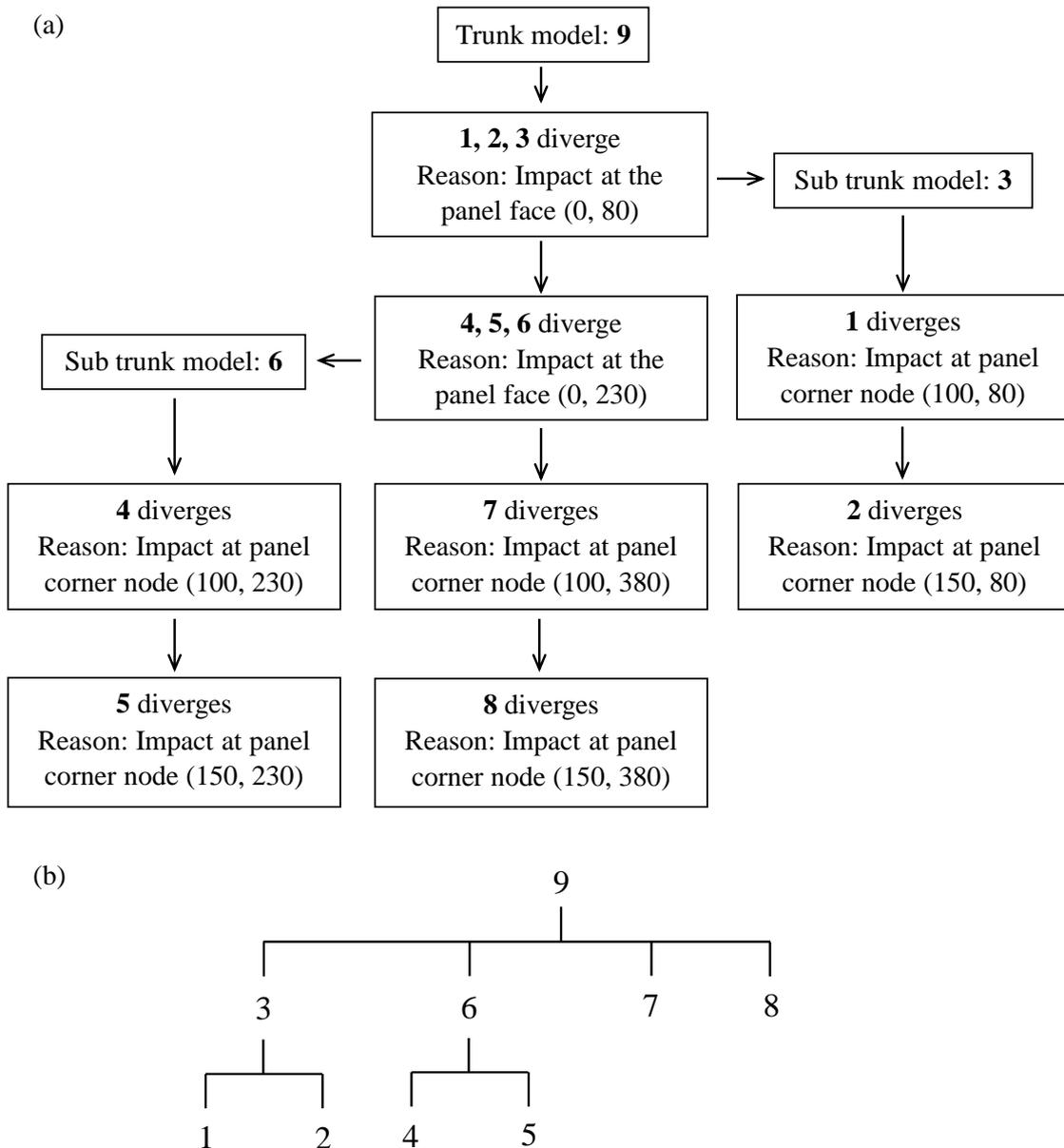


Figure 4.5: Algorithm output (a) flow chart, (b) deviation tree. Coordinates given in (a) are given in millimetres to indicate where the model will deviate relative to the charge centre in each model.

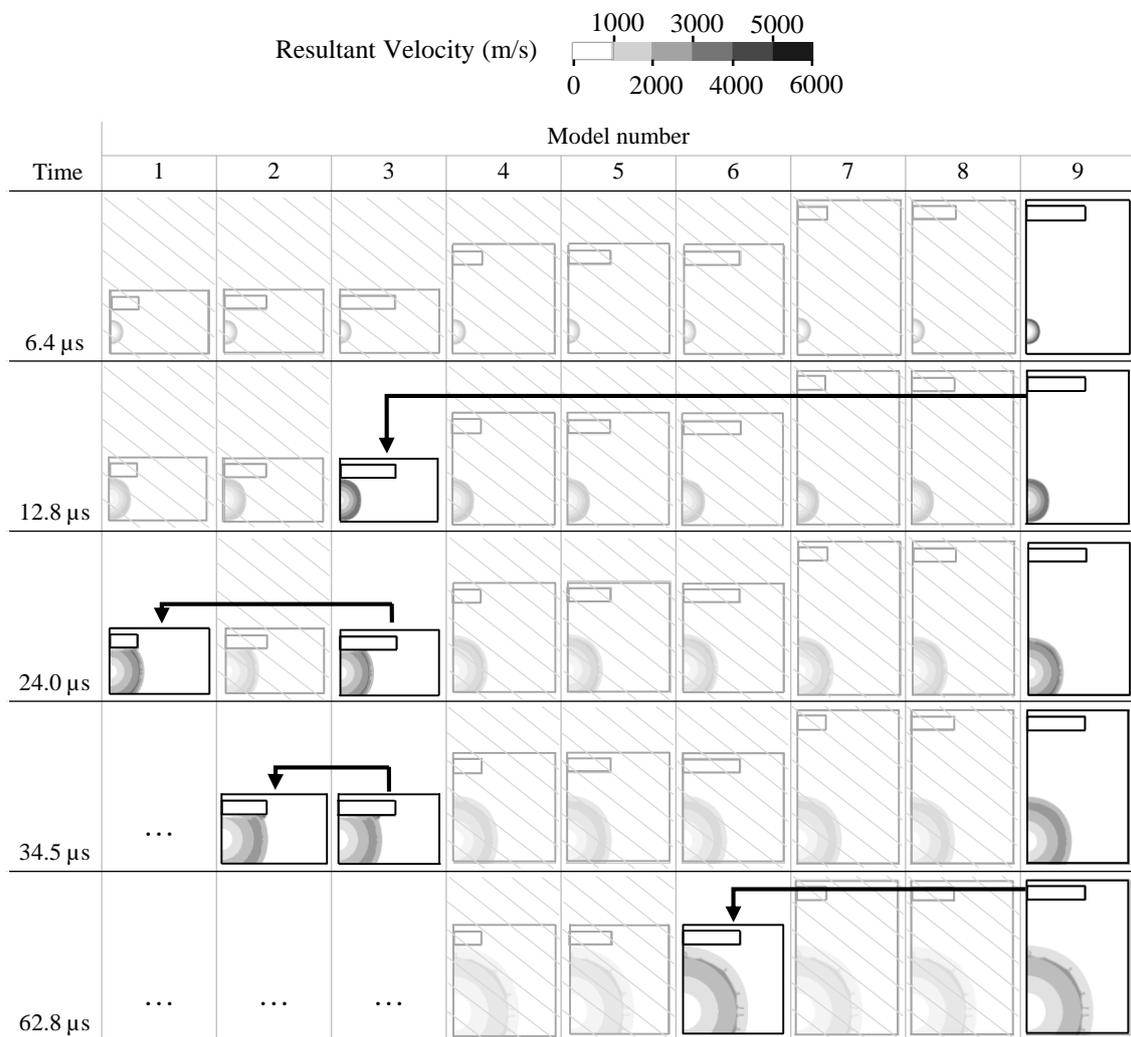
Table 4.5 shows that when simulating all 9 models in LS-DYNA from birth to termination, the required computation time is 1643 s (27 mins 23 s). Use of the BA and the simulation order shown in Figure 4.5 reduces this to only 847 s (14 mins 7 s), a saving of approximately 50% (13 mins 16 s; 48.4%). The reported simulation times were achieved with each model being solved one after another on an Intel Core i7-10700 2.9 GHz processor with 16 GB of RAM. It should be noted that the magnitude of the achieved benefit can be expected to vary greatly depending on the similarities and complexity of the models specified in the batch of tests. However, this basic example still proves that simulation efficiency can be enhanced with the use of the proposed method.

Table 4.5: Required computation time with and without use of the BA method.

Model No.	1	2	3	4	5	6	7	8	9
No branching (s)	90	90	88	173	171	175	284	283	289
With branching (s)	55	49	73	74	70	100	72	65	289
Saving (s)	35	41	15	99	101	75	212	218	0
Saving (%)	38.9	45.6	17.0	57.2	59.1	42.9	74.6	77.0	–

Algorithm computation time:	1 s
Standard method computation time:	1643 s
Branched method computation time (inc. algorithm):	848 s
Total saving (inc. algorithm):	48.4%



Continued in the next Figure.

Figure 4.6: Progression of the shock front from the LS-DYNA models showing where data mapping occurs. Faded cells show the simulation steps that are not required if the BA is used. Data mapping is shown with black arrows.

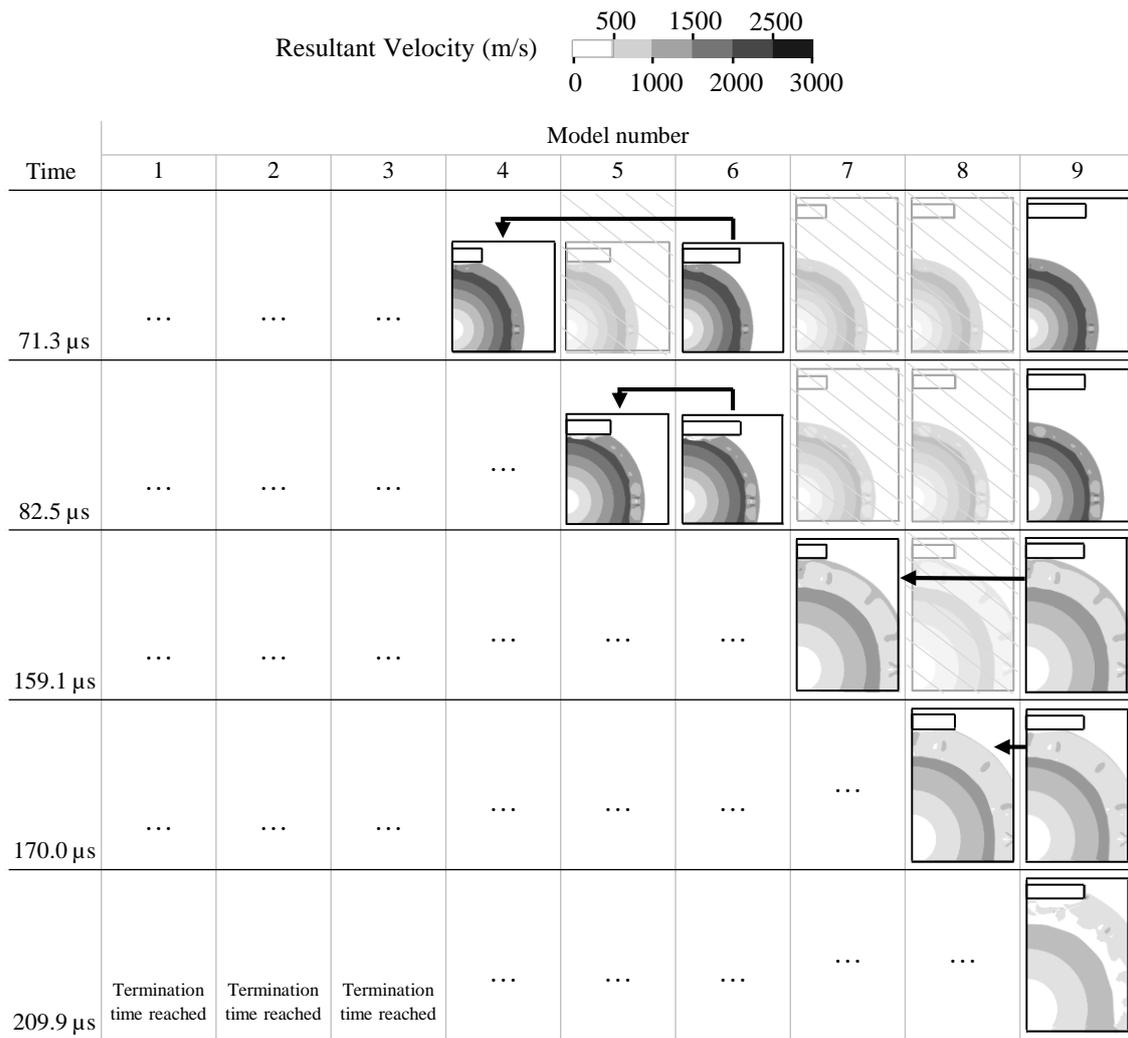


Figure 4.7: Continuation of Figure 4.6. Progression of the shock front from the LS-DYNA models showing where data mapping occurs. Faded cells show the simulation steps that are not required if the BA is used. Data mapping is shown with black arrows.

As expected, Table 4.5 also shows that when using the BA, models 7 and 8 experience the greatest reduction in required computation time (75–77%) due to how the deviation points occur at later stages of the time-stepped solutions. A larger amount of the trunk model (9) can be simulated without a difference in output for these cases when compared to models 1, 2 and 3 that must diverge as soon as the shock wave has progressed 80 mm from the charge. This is why the BA works to identify the trunk model by assessing which model from the framework has the unique initial influence that occurs at the latest time step of the simulation.

For a visual representation at how the data is mapped between each domain, Figures 4.6 and 4.7 show how the progression of the blast wave occurs in each domain, with the faded images relating to steps that are omitted from the analysis if the BA is used. These figures also highlight the importance of using the charge centre as a coordinate that is shared between each model since the blast wave can be mapped relative to this point regardless of the domain size and shape.

### 4.3.5 Mapping results

The mapping functionality in many numerical solvers is intended to allow for adaptive mesh refinement where dense grids of elements and nodes need to be specified for the start of a simulation where more detail in the parameter field may be required to preserve the energy of the detonation. When this stage of the process has concluded, the data can be mapped to the same domain geometry, but with a coarser mesh that helps to reduce computation time. Consequently, the mapping process requires interpolation of values to fit to the new mesh.

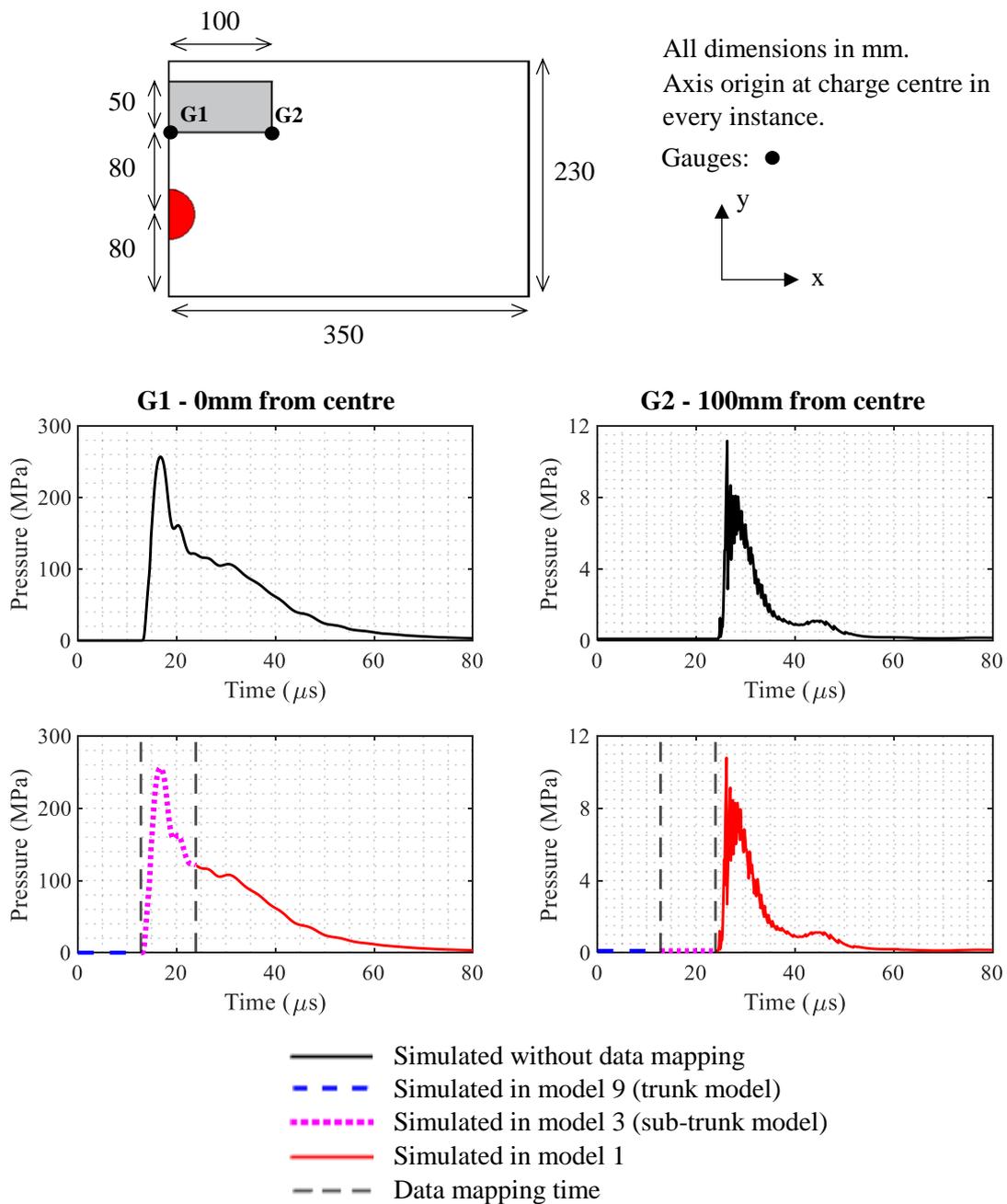


Figure 4.8: Pressure time histories for model 1 generated from simulations with and without use of the BA.

Interpolation is not required for the models in this example since the mesh conditions remain consistent throughout the batch. This results in rapid initialisation of the models receiving the parameter field from a trunk or sub-trunk model with no loss of information that could lead to differing outputs between a mapped and non-mapped solution.

Figure 4.8 shows this to be the case with results from the branched analysis being compared to those from the full simulations for model 1. The combined outputs from each stage of the mapping procedure are shown using different line types, with the results from the full simulation shown with a solid black line and mapping times shown as black dashed vertical markers. The results are effectively indistinguishable, demonstrating that the BA has successfully identified a robust and repeatable methodology for removing unnecessary steps in the simulations.

## 4.4 Summary

This chapter has proposed and applied a new branching algorithm that can be used when modelling a range of similar numerical models with differing input parameters. The algorithm removes the need to simulate the steps of a simulation more than once if they occur within multiple models by identifying the similarities between the inputs and the newly defined ‘influences’.

It is shown that by using the developed approach for a simple blast analysis featuring 9 models, approximately 50% of the required computation time can be saved when compared to simulating each model independently. Furthermore, this process occurs with no loss of accuracy.

Currently, this method is applied in 2D, with influences being defined by obstacle surfaces and vertices. However, seeing as blast loads are typically modelled using CFD in 3D, the following chapter expands on this methodology to ensure that any batch of domains can be processed by the BA to reduce the required computation time.

# Chapter 5

## The Branching Algorithm for blast analyses in 3D

### 5.1 Introduction

In the previous chapter, the Branching Algorithm (BA) was introduced as a tool that can reduce the computation time of batches of numerical models. The concept has been shown to work for a blast scenario in 2D, however the greatest savings will be achieved when modelling three-dimensional problems.

This chapter discusses the developments required for adapting the BA specifically for use with blast models in 3D. This includes the development of a range of features that allow the code to provide suitable data mapping times for any batch of geometries and setup conditions, with a focus on accurate environment representation and wave tracking.

### 5.2 Domain discretisation and influence comparison

A key difference between the blast example shown in Section 4.3, where each model 2D, and the need for the algorithm to function in 3D relates to how influences are represented and defined. As shown in the previous example and in Figure 5.1, 2D blast models could be represented by influences that only cover the obstacle vertices and a single surface definition. This is because if the wave hits the surface, it is known that the next point it will hit is the closest vertex. Conversely, in a 3D model, the wave can progress in multiple directions meaning a greater understanding of the wave's travel path is required.

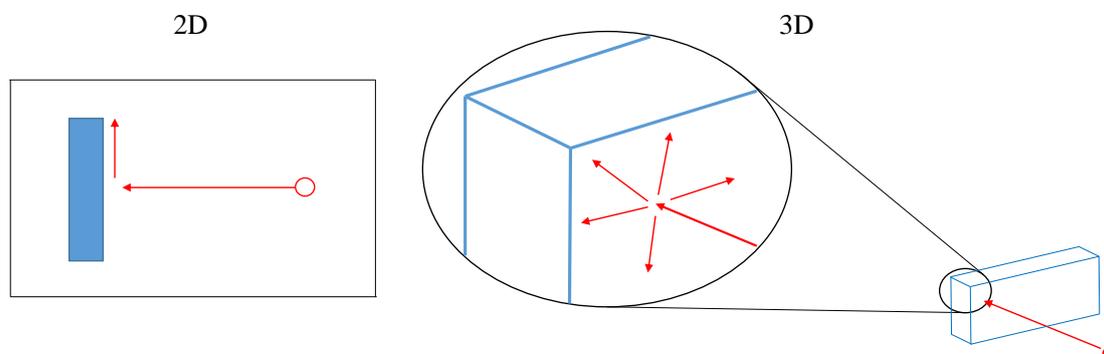


Figure 5.1: Comparison of how a blast wave may interact with a rigid obstacle in 2D and 3D. Red arrows indicate the path of the blast wave emanating from the charge, shown as a red circle.

The way in which the algorithm identifies and compares influences must account for variations in the type of impact the wave is predicted to experience. Specifically, for the BA to identify which parts of a given domain are reached first, potentially causing a deviation, the geometries of each model must be represented discretely. This requires a series of nodes to be defined that act as individual influences on all surfaces of the obstacles and boundaries, and along all edges in each domain.

A mesh of influences representing each obstacle and boundary in every model of a given batch can be generated using a user defined mesh spacing that coincides with the edges of the obstacles and locations of any charges. As shown in Figure 5.2 on the left hand side, if the mesh spacing is specified to be 0.1 and a local grid of points is used to form the surface and edges, there is a high possibility of misaligning influences. In this case, the BA would identify the blast wave's impact at any node as a numerical deviation when comparing these models. It is therefore necessary to use a global grid, that is centred on the respective charge centres of each domain, with the same mesh spacing in every model.

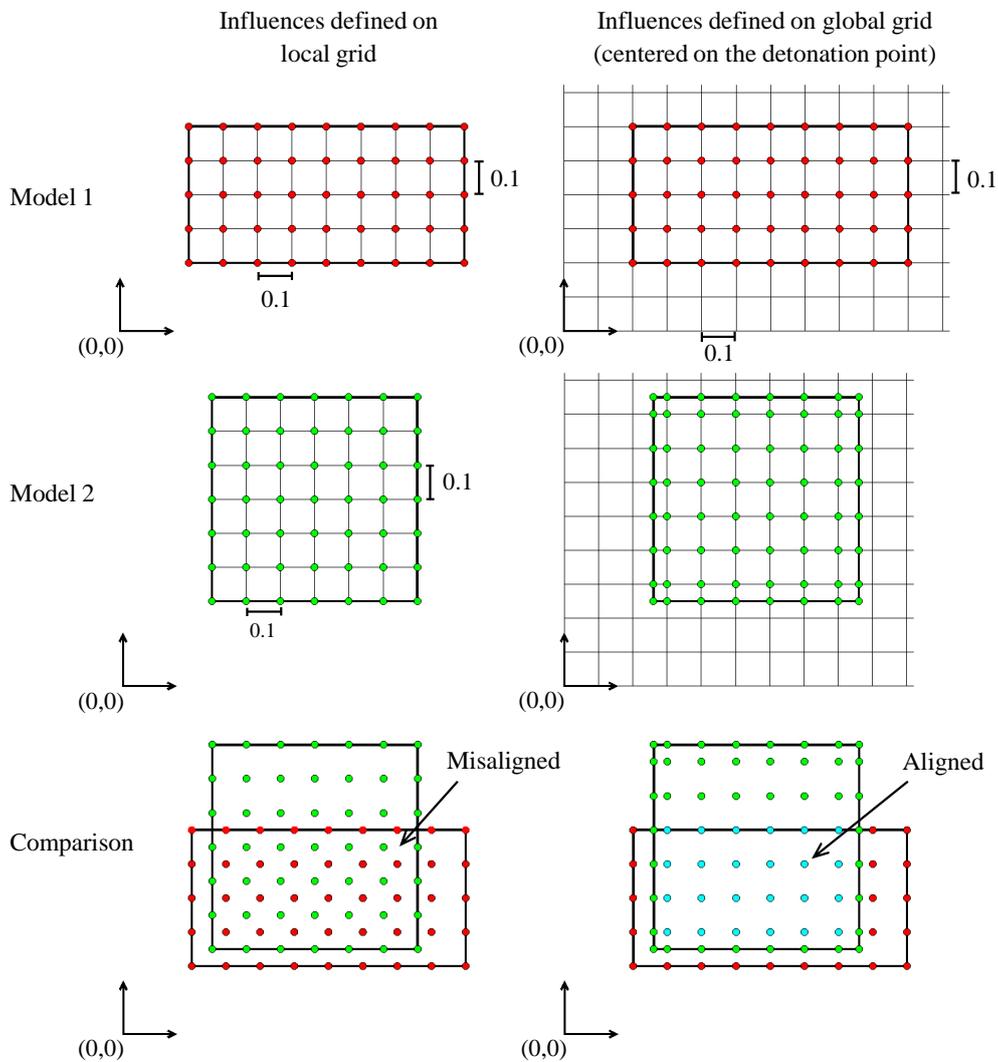


Figure 5.2: Example of how the adopted meshing strategy, using a mesh spacing of 0.1, enables influences to be defined relative to the charge centre,  $(0,0)$ , in comparable terms regardless of the orientation of the obstacles in each model.

The right hand side of Figure 5.2 shows of how this adjustment allows for aligned nodes that the algorithm would ignore when considering if a deviation has occurred. Through implementing this meshing strategy, if a surface is present in multiple models at the same relative location, a range of influences will be defined in an identical way regardless of the domain orientation or size.

Comparison of these globally defined influences is achieved through consideration of the relevant variables assigned to each new influence type shown in Figure 5.3. These vectors are stored with the influence locations to allow for the deviation points to be identified in the BA. An example is given by Table 5.1.

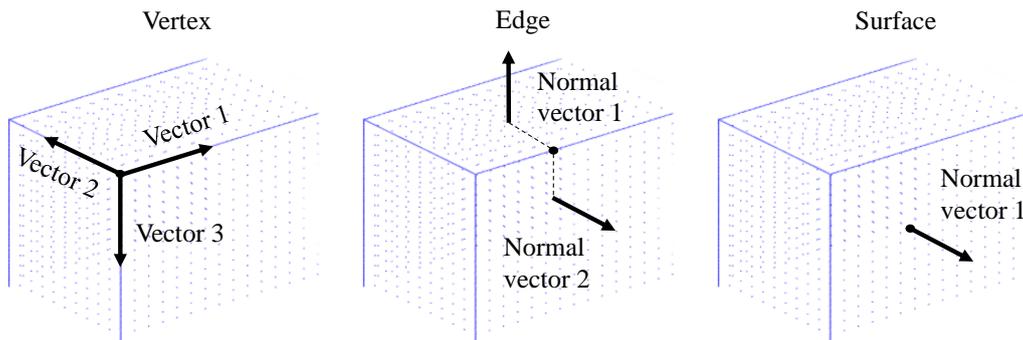


Figure 5.3: Variation of values stored by the algorithm enabling comparisons to be made between each identified influence. Blue dots correspond to each node in the discretisation of an arbitrary obstacle.

Table 5.1: Example influence table structure for 3D analyses.

Model No.	Relative coordinate			Vector(s)	Influence type	Comparison metric
	x	y	z			
n	~	~	~	~	~	~
...	...	...	...	...	...	...
n+1	~	~	~	~	~	~
...	...	...	...	...	...	...

It should also be noted that the comparison metric will no longer be assigned to the direct distance as seen previously. For the 3D case, a more robust approach will be discussed in Section 5.5.

### 5.3 Obstacle influences

Simulating a batch of models using existing numerical solvers requires the user to input each model's geometry alongside any input conditions related to the explosive charge and ambient properties. Due to the complexity of some structures, it is common for the geometries to be added using the stereolithography (STL) ASCII format, where each arrangement is represented by a series of triangular surfaces. STL files can also possess various useful attributes such as the outward normal vector of each surface and node connectivity, allowing solvers to rapidly initialise a test domain.

These benefits can be applied to the BA, with the three different influence types, vertex, edge and surface, being generated using various discretisation methods. The following sections will explain each process before the adopted approach for wave tracking is presented to conclude the setup phase of the proposed method.

### 5.3.1 Surface influences

In defining the influences present on an obstacle's surfaces, the STL of a given geometry must be refined so that there are no triangles with equal surface slopes that also possess a shared edge. This commonly occurs as polygons are divided into multiple triangles with the dividing lines resting on a surface that maintains a consistent plane. To ensure that the BA can correctly compare influences, the shared edges possessing this characteristic must be defined by 'surface' rather than 'edge' influences.

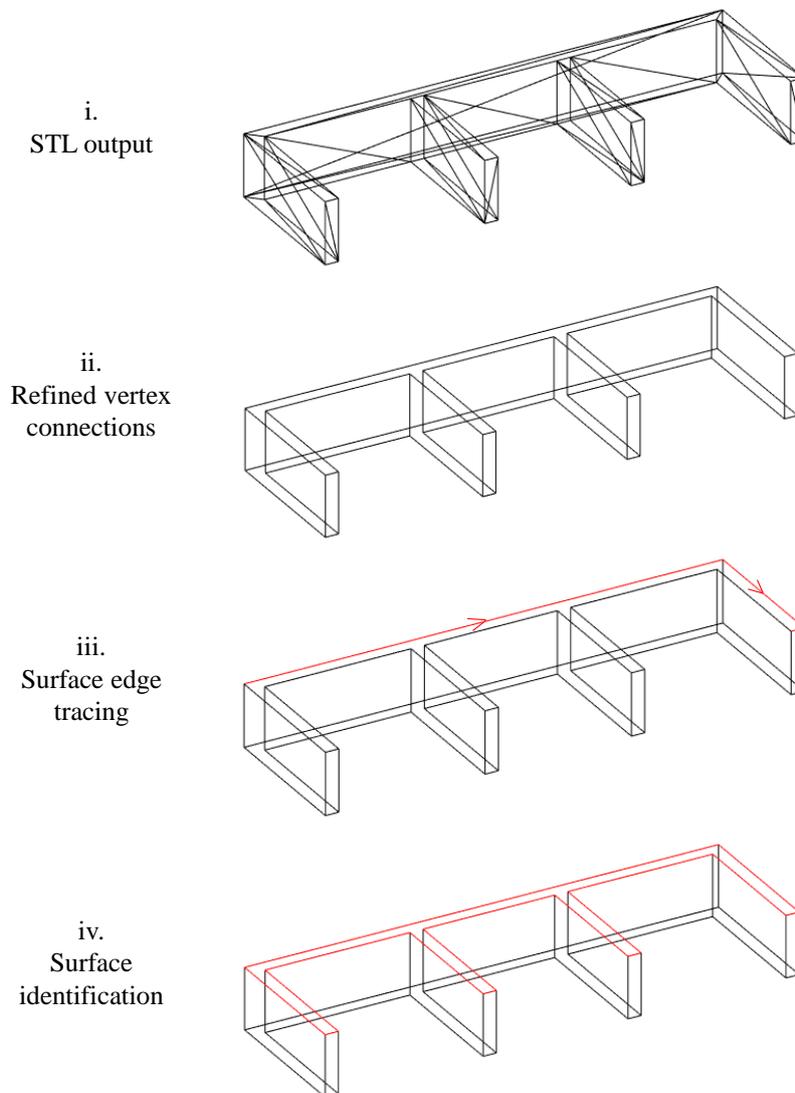


Figure 5.4: Surface identification from an STL file required for defining global influence locations. Between steps i and ii the surface count decreased from 84 to 48 as various triangles were combined to form polygons with varying vertex counts.

Figure 5.4 presents the approach for optimising the surface representation of a given geometry with the first step, given in plot i and ii, being to refine vertex connections. This requires a comparison of the two connected surface's outward normal vectors for every edge given by the STL. Where only one unique vector is present, the edge is removed.

Next, plot iii shows how new surfaces are compiled. Firstly a vertex on the geometry is selected, then one of its connections is followed to another vertex and the common outward surface vector for the connected surfaces is stored. From this second vertex, the path is traced to its connection that shares the stored surface vector. This continues until the path is back to the starting node with each visited vertex forming part of the new surface, shown in plot iv.

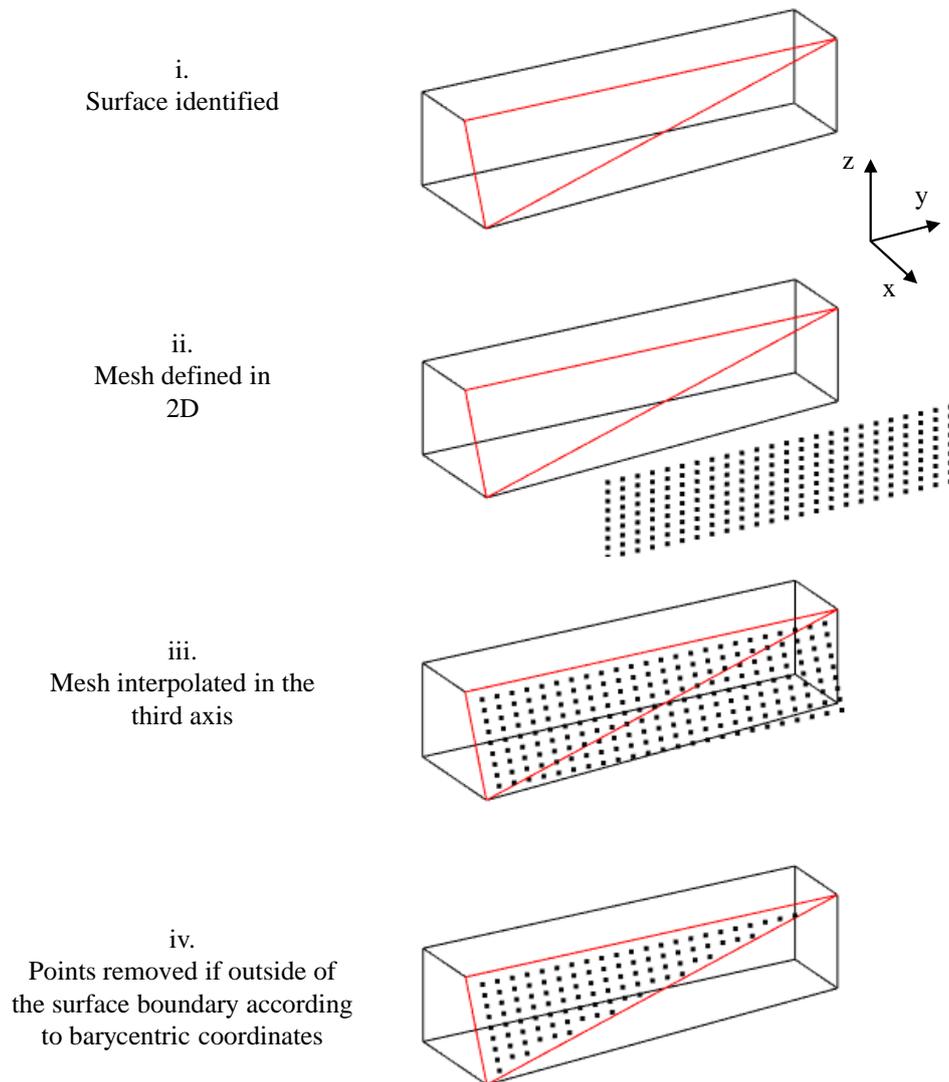


Figure 5.5: Process of defining influence locations on the surface of an obstacle.

Using the refined surfaces, the discrete influences can now be defined. Figure 5.5 details this process that starts with the global mesh of potential points being defined in 2D within the relevant vertex constraints. Consideration of the surface plane is required for this step so that the correct axes are used for forming the grid of nodes. For the example geometry shown in Figure 5.5, this plane is defined in the y and z axes because the smallest variation between the vertex coordinates is in the x axis.

Plot iii shows the output of interpolation in the 3rd axis, achieved using the 3D surface plane equation given as:

$$ax + by + cz + d = 0 \quad (5.1)$$

Where,  $\mu$ ,  $\nu$  and  $\tau$  are three vertices associated to a given surface with,

$$a = (\nu_y - \mu_y)(\tau_z - \mu_z) - (\tau_y - \mu_y)(\nu_z - \mu_z) \quad (5.2)$$

$$b = (\nu_z - \mu_z)(\tau_x - \mu_x) - (\tau_z - \mu_z)(\nu_x - \mu_x) \quad (5.3)$$

$$c = (\nu_x - \mu_x)(\tau_y - \mu_y) - (\tau_x - \mu_x)(\nu_y - \mu_y) \quad (5.4)$$

$$d = -(a\mu_x + b\mu_y + c\mu_z) \quad (5.5)$$

Following interpolation, the nodes lie on the global grid in the y and z axes with x coordinates that place them on the surface of the obstacle. In this example it is clear that there are a number nodes that lie outside of the surface boundary, consideration of barycentric coordinates are therefore required as they can be used to determine if the point, P, rests within the surface constraints of a given triangle.

The concept of barycentric coordinates is given in Figure 5.6, with the point, P, being positioned according to some combination of the  $\alpha$ ,  $\beta$  and  $\gamma$  variables that are defined using the ratio of areas relating all four points of interest. This is represented by the following equations:

$$\alpha = A_\mu/A \quad (5.6)$$

$$\beta = A_\nu/A \quad (5.7)$$

$$\gamma = A_\tau/A \quad (5.8)$$

$$P = \alpha\mu + \beta\nu + \gamma\tau \quad (5.9)$$

Where, vector mathematics is required to calculate the area of each triangle on the surface:

$$v = (\nu - \mu) \times (\tau - \mu) \quad (5.10)$$

$$v_\mu = (\tau - \nu) \times (P - \nu) \quad (5.11)$$

$$v_\nu = (\mu - \tau) \times (P - \tau) \quad (5.12)$$

$$v_\tau = (\nu - \mu) \times (P - \mu) \quad (5.13)$$

Leading to the equations for  $\alpha$ ,  $\beta$  and  $\gamma$  to be:

$$\alpha = \frac{v \cdot v_\mu}{\|v\|^2} \quad (5.14)$$

$$\beta = \frac{v \cdot v_\nu}{\|v\|^2} \quad (5.15)$$

$$\gamma = \frac{v \cdot v_\tau}{\|v\|^2} \quad (5.16)$$

Resulting in a point being defined as inside the triangular surface if all of the following constraints are satisfied:

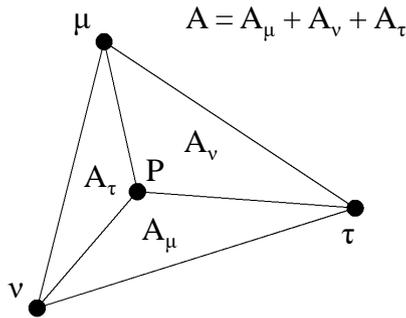
$$\alpha + \beta + \gamma = 1 \quad (5.17)$$

$$0 < \alpha < 1 \quad (5.18)$$

$$0 < \beta < 1 \quad (5.19)$$

$$0 < \gamma < 1 \quad (5.20)$$

i.



ii.

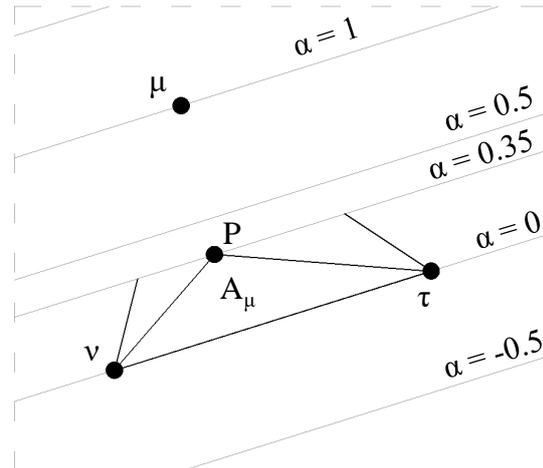


Figure 5.6: Visual representation of using area ratios with barycentric coordinates to identify if a point lies within a triangle. Plot ii shows how the value of  $A_\mu$  only depends on  $\alpha$ . This also applies to  $A_\nu$  and  $A_\tau$ , with  $\beta$  and  $\gamma$  respectively.

If a point is on an edge of the triangle, one of either  $\alpha$ ,  $\beta$  or  $\gamma$  will equal 0, and the other two will be between 0 and 1. Similarly, a point is at a vertex if two of these values equal 0 and the third is 1. Logically, the centre of the triangle is present at the point where  $\alpha$ ,  $\beta$  or  $\gamma$  all equal one third.

For the BA, the STL representation of the geometry is used so that the obstacles are defined by triangular surfaces in the relevant 3D space. As each surface influence is identified, the outward normal vector of the surface that it is associated to is stored in the vector column of the example table shown in Table 5.1.

### 5.3.2 Edge influences

The creation of edge influences requires a similar approach to previous section, but with consideration for the vertex connections rather than the triangular surfaces. For this process, Figure 5.7 shows how for each edge in the domain, start and end nodes are identified with a vector being calculated to represent the connection's direction according to the following equation:

$$\vec{\mu\nu} = \nu - \mu \quad (5.21)$$

The start node is assigned to the vertex with the lowest value in the axis that has the greatest variation in magnitude between the nodes. Here, this dominant axis is in the  $z$  direction.

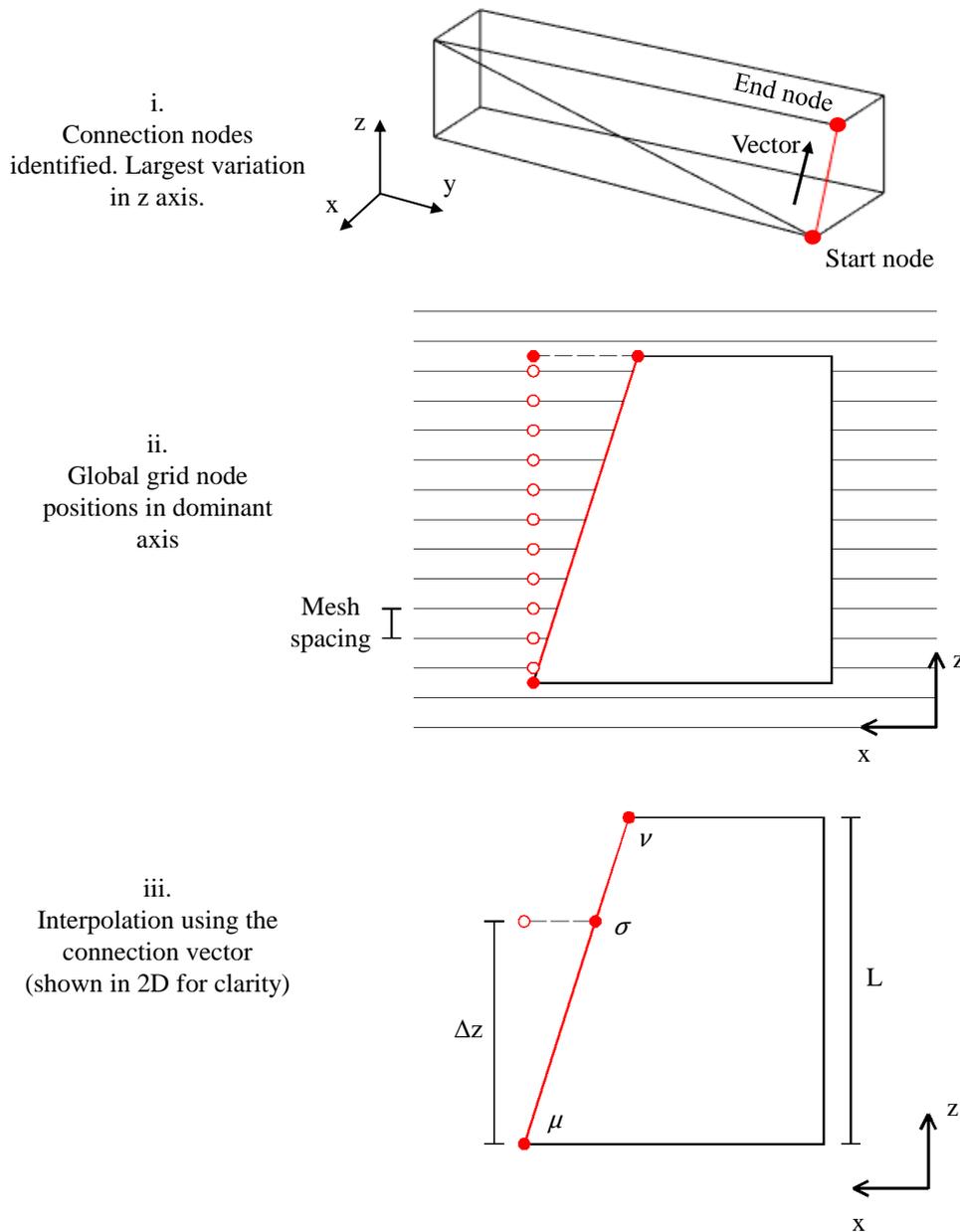


Figure 5.7: Defining influence locations on the edges of an obstacle.

Next, the closest global grid position in the  $z$  direction that is greater than the starting node is used to define the first edge coordinate. The chosen mesh spacing provides the subsequent influence positions in this axis up to the end node, shown in plot ii.

Following this, each point,  $P$ , is interpolated onto the edge using its corresponding length ratio,  $r$ , as a means of scaling the connection vector by an appropriate amount to achieve the required spacing. This ensures that the edge direction is also preserved. The interpolation process using the vector line equation is summarised below, with the numerator in Equation 5.22 corresponding to the numerical difference in the dominant axis between the starting node and the point being interpolated.

$$r = \frac{\Delta z}{L} \quad (5.22)$$

$$P = \mu + r \times \mu\vec{\nu} \quad (5.23)$$

For clarity the Figure 5.7 shows this process in 2D. However, vector  $\mu\vec{\nu}$  will also include the difference in the  $y$  values of the vertices, allowing for correct adjustments in every axis.

The comparison values stored with each edge influence equates to the outward normal vectors of the two connected surfaces running alongside each given edge. These vectors are provided by STL input files and as discussed in Section 5.3.1, refined surface connections are generated making it simple to identify the relevant surface properties.

### 5.3.3 Vertex influences

The final influence type associated to obstacles in a given domain requires no additional computational effort to generate and store the related comparison values. This is because the vertex influences are given by obstacle nodes directly in a geometry's STL variables with the stored connection vectors being calculated when generating edge influences.

## 5.4 Boundary influences

Another set of influences that must be included are related to the boundary conditions of each model. Commonly defined as ambient/transmit or rigid/reflect, both boundary types are discretised because multiple models may not feature the same domain shape and size. This means that data mapping might not be possible as the blast wave reaches a boundary, regardless of if it is able to transmit with a negligible impact to the parameter field.

Each boundary is discretised using the process shown in Figure 5.8. A mesh of nodes is defined according the global node spacing in both axes related to the given boundary. Any obstacle surface influences that share the position of a boundary node are removed along with the boundary node that they matched to. This is because the duplicate surface influences are effectively internal and the blast wave would not be able to reach those points. In plot ii, the removed surface influences are shown as black dots. Similarly, any boundary nodes that share the position of an obstacle vertex or edge influence are removed, but the obstacle influences remain as they can still be reached by the blast.

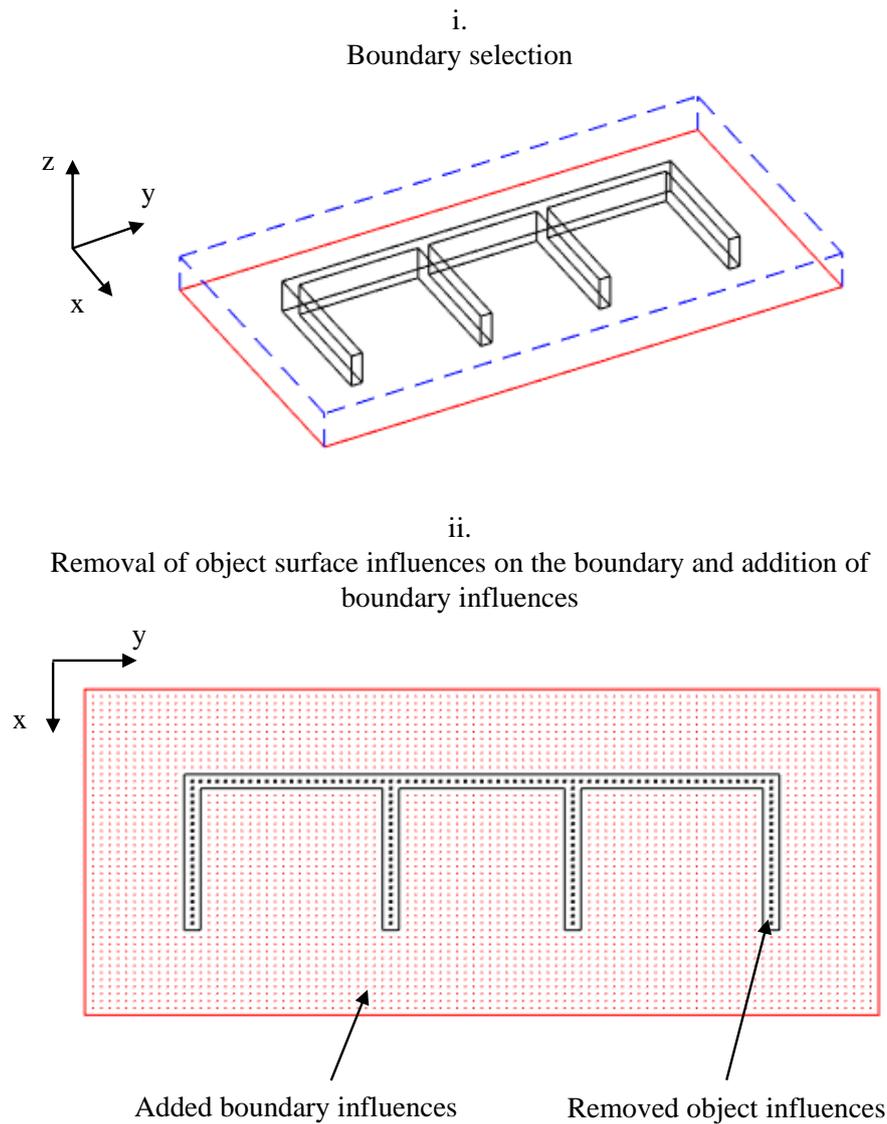


Figure 5.8: Generating boundary influences with the removal of obstacle surface influences on the boundary.

The vectors stored in the influence table relate to the surface vector facing into the domain. This allows models to share identical parameter spaces even if the wave impact a rigid boundary in one domain, and an obstacle in another, provided that both are perfectly rigid and in the same location relative to the charge.

## 5.5 Comparison metric

The comparison metric used to sort the influences according to the step where they are encountered in the numerical simulation was previously assigned to the direct distance between the charge and the relevant point to utilise the simplicity of the 2D geometries presented in the previous chapter. However, this is not suitable for complex models where the blast wave is expected to reflect off rigid surfaces and clear around obstacle corners.

Ideally, the time of arrival of the blast wave would be calculated at each influence to definitively sort when each influence becomes active in each numerical model. However, obtaining this value would require some simulation of blast wave propagation before the algorithm is used, hence ignoring the benefit of the proposed method. Alternative approaches could include the use of suitable Fast Running Engineering Models for complex 3D geometries that predicts the wave arrival time, but as discussed in Chapter 2, at present no such method exists.

Consequently, the use of shortest path analysis has been explored and adapted for use with blast models to provide a reasonable estimation for the order of when each influence in a given batch of tests is reached in each simulation.

### 5.5.1 Shortest path analysis

Taking inspiration from route finding algorithms used in modern satellite navigation products such as ArcMap (Environmental Systems Research Institute 2020), the discrete series of influences can act as nodes on a connected graph linking the charge to all points in the domain. Well-established shortest path analysis (SPA) methods, such as Dijkstra's algorithm, can then be used to identify the distance of each connection that corresponds to the path the blast wave would have to travel to each point. In the aforementioned commercial software, SPA is implemented to allow for varied path goals. These include, finding the route with the shortest distance, lowest travel time, or the one that is the most scenic.

In the example shown by Figure 5.9, Dijkstra is applied to find the shortest distance between each node in the same way that it would be applied in the BA. Here the first node, B, is given the permanent label, 0, and order label, 1. From here, plot i shows that each connected node is assigned a temporary label with the closest being labelled as a permanent move. In this case it corresponds to node A, which is then given the order label, 2. Each temporary label for the connections from this node is then updated only if the existing value is larger than the new one, as per ii. When moving from node A to node F, the temporary label is not updated since the distance would equal 7 whereas it was only 6 when moving from node B. The next permanent label can now be defined as the smallest temporary label given to any node, corresponding to node F in this case.

Following this process for all connections and nodes allows for the shortest paths through the network to be discovered, plot iii. By tracing backwards through the network, the route to a given target can be derived with the path lengths corresponding to the permanent labels. In navigation applications, the nodes could correspond to cities in the UK, or houses along a delivery route, with the connections acting as the roads that form overall graph.

An alternative to Dijkstra that is implemented in a similar way is the  $A^*$  method. The key difference with this adapted approach is that it includes a heuristic to focus the progression of the search, reducing the number of paths being travelled. The most common heuristic for travel and distance applications is the direct, 'as the crow flies', distance which is used to assess if movement along a certain connections is getting closer to the target point. Dijkstra's algorithm is therefore often reported to be less computationally efficient for single searches when compared to the  $A^*$  approach (Ortega-Arranz et al. 2014).

However, for this application all nodes (influences) in the graph need to be visited by a route starting at the charge. Using Dijkstra therefore means that every node is reached in one pass of the algorithm, and the distance to each node can be stored as the shortest path comparison metric in the influence table.

Shortest path from B to D.  
Path lengths shown on the connections.

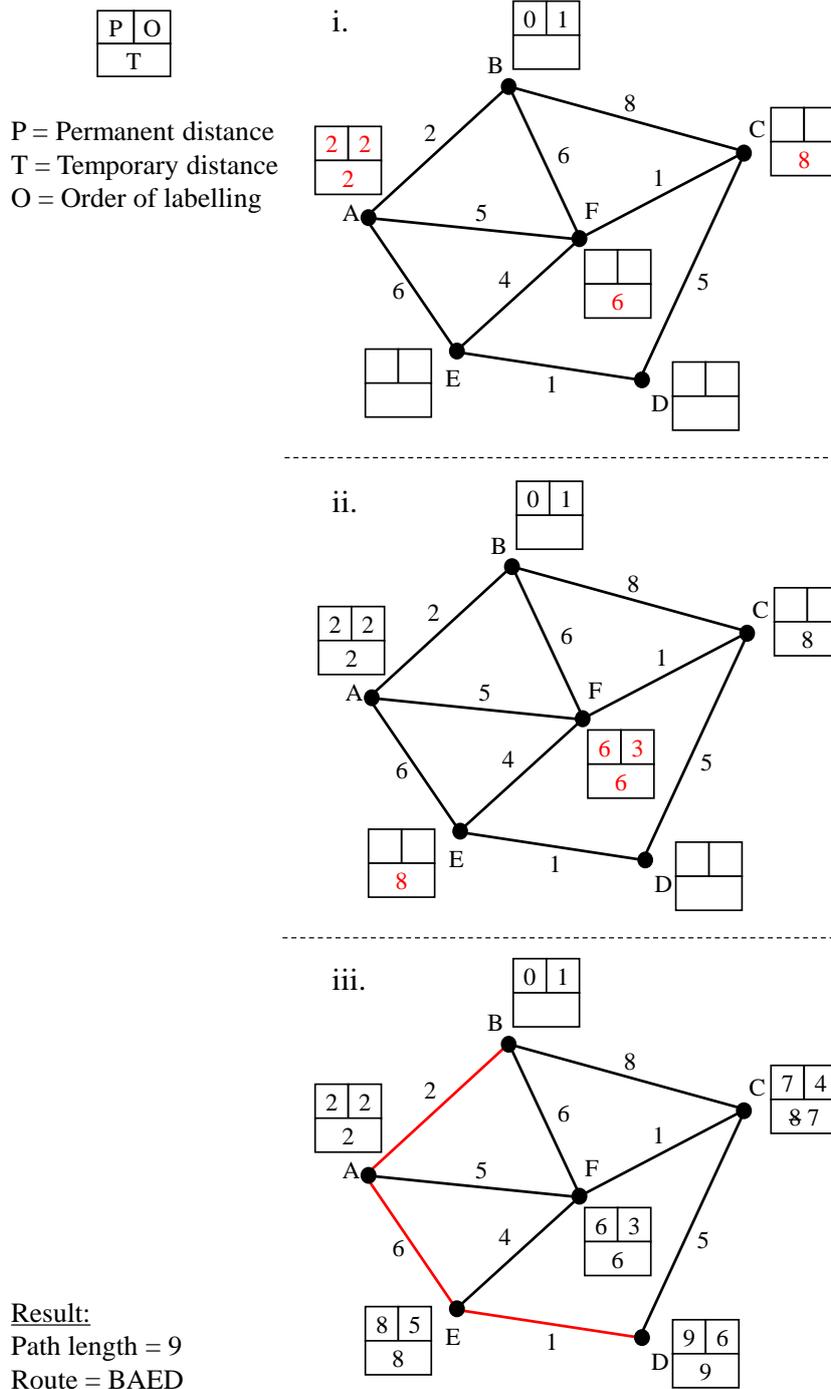


Figure 5.9: Example use of Dijkstra’s algorithm for shortest path analysis in 2D.

This method of obtaining a comparison metric is therefore suitable for the BA, especially considering that only the connection distances between each pair of nodes need to be calculated and stored. The resulting travel distances are used for this study directly as the comparison metric, however, further consideration of the wave velocity could be included to derive arrival times. Adapting this approach for the BA in 3D therefore involves condensing the geometry and charge location into a 2D graph similar to the one shown in Figure 5.9. This requires consideration of physically valid node connections, discussed in the following sections.

## 5.6 Node connections

### 5.6.1 Nearest neighbour analysis

With the shortest path analysis method being used to determine the wave travel distance to each influence in a given domain, the network of influences must be connected in a physically accurate way. This must be achieved whilst also maintaining a low algorithm computation time, hence necessitating a balance between the number of connections and the freedom of the path's movement.

Since the obstacle and boundary influences are discretised according to a user specified mesh spacing, connections made between neighbouring influences can be easily determined through consideration of the connection distance. In particular, if the connection length between two influence nodes,  $l$ , is less than the distance to the corner of the cube of nodes formed around a given influence, then the connection is stored. This results in a maximum number of connections per influence of 26 as shown in Figure 5.10 plot i and formalised in Equation 5.24.

$$l = \sqrt{\sqrt{m^2 + m^2} + m^2} \quad (5.24)$$

Where  $m$  is the mesh spacing.

To identify the connections that meet the condition given by Equation 5.24, a k-dimensional tree (kd-tree) data structure is adopted (Bentley 1975). This process organises points in such that for problems involving  $N$  samples and  $D$  dimensions, the computational cost scales approximately according to  $O[DN \log(N)]$ . Therefore, they can be more efficiently processed when compared to storing and processing data in simple lists or arrays with scaling according to  $O[DN^2]$ .

Increasing the maximum number of connections by allowing for paths spanning to the second tier of surrounding nodes would benefit the possible path directions by allowing for a more accurate representation of a blast wave's omnidirectional expansion. However, increasing the number of edges in a connected graph also increases the time required to analyse the shortest path. The decision has therefore been made to restrict the maximum connection count to 26 for each node, with an additional optimisation step being introduced in Section 5.6.5 to assist with the accurate representation of wave propagation.

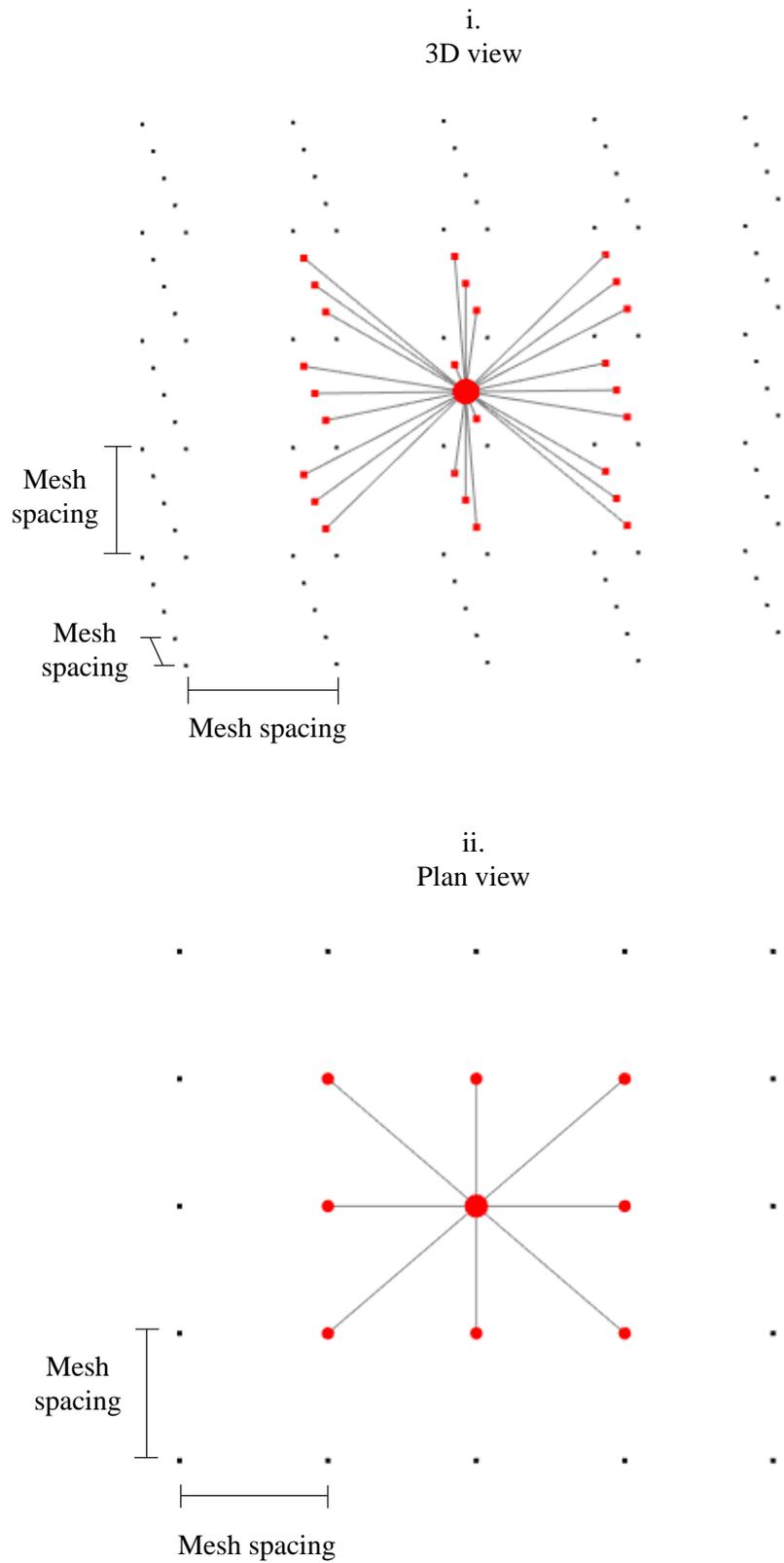


Figure 5.10: Visual of the 26 potential connections that an influence or free node can make in the connected graph of the domain. Red nodes are connected to the central node, black dots are not connected.

### 5.6.2 Charge connections

A key element of establishing node connections for use with SPA is related to how the charge can link to the network of obstacle and boundary influences. Clearly the charge may not be close enough to each obstacle to satisfy the distance requirement set out by Figure 5.10 and so an alternative method must be used.

The first option includes establishing direct connections from the charge to all visible influences. These being the influences that can form a connection to the charge without the connecting line intersecting any solid bodies in the domain. However, as shown in Figure 5.11, obstacles being shielded by others will not be connected to the charge using this approach in conjunction with the nearest neighbour method. Thus limiting the ability of the BA to identify if the presence of any of these entries would cause a numerical deviation.

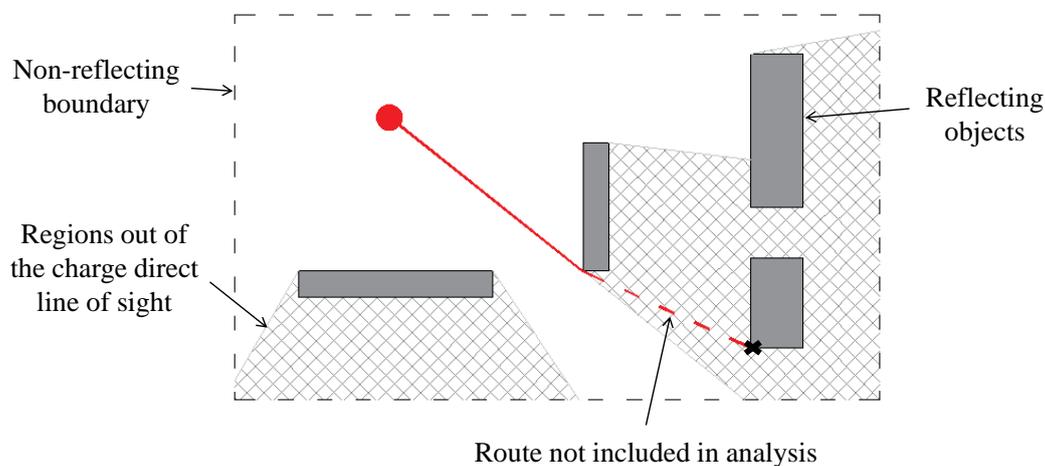


Figure 5.11: Example of how direct connections from the charge may lead to obstacles with no traceable wave path.

The alternative, adopted approach therefore takes inspiration from numerical models where the obstacles *and* the free space is represented by individual elements. Using the user defined mesh spacing, a grid of ‘free nodes’ can be formed, allowing the charge to link into the node network according to the nearest neighbour criteria discussed in Section 5.6.1. This also enables the existing obstacle and boundary influences to form connections across the free space, resulting in a fully traversable graph to be used with Dijkstra’s algorithm.

### 5.6.3 Generating free nodes

Figure 5.12 shows the process required to create the free node network that will link the charge to the obstacle/boundary influences. In simple terms, the approach works by projecting lines through the domain according to the chosen mesh spacing. The intersection points with any obstacles are then identified and the gaps between each intersection are categorised as ‘internal’ or ‘external’. Only the external spaces are discretised with increments of the mesh spacing to fill the domains free space with nodes.

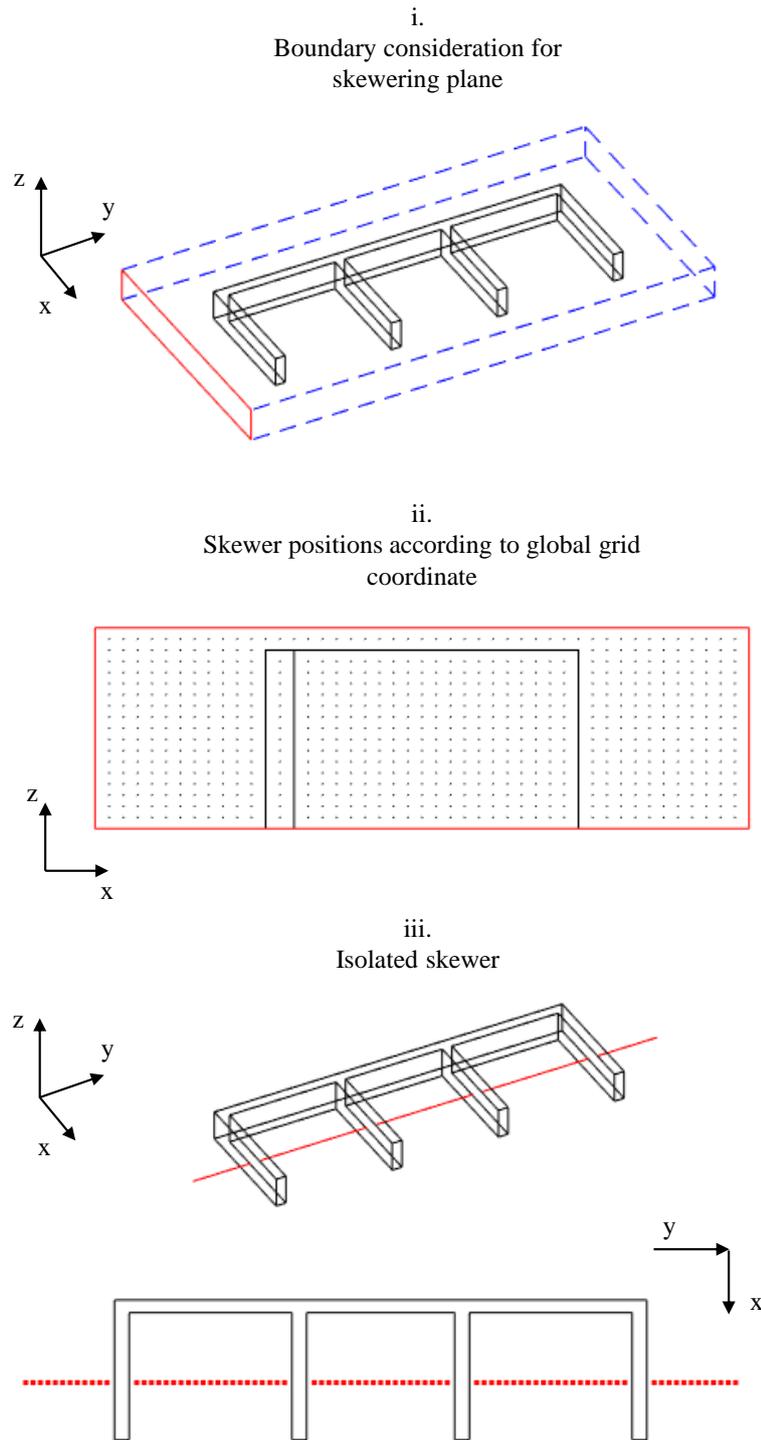


Figure 5.12: Generating free nodes using a 3D line intersection check approach.

Optimising this ‘skewering method’ involves only projecting lines (skwers) in the axis combination that will produce the fewest lines for the chosen mesh spacing. Plot i shows that this corresponds to the  $x$  and  $z$  axes for this example, with plot ii providing a side view of each skewer position. For all other axis pairs, the number of skewers would be larger and so more computationally expensive line intersection checks would be required.

When checking each skewer for intersections in the domain, the following equations are used to determine if the vector of each skewer intersects with the planes of each surface. Barycentric coordinates, discussed in Section 5.3.1, are then used to establish if the point rests within the surface boundaries.

Considering a skewer defined by the coordinates  $(x_0, y_0, z_0)$  and  $(x_1, y_1, z_1)$ , the direction vector of the line can be given as,

$$v = [x_0 - x_1, y_0 - y_1, z_0 - z_1] \quad (5.25)$$

Then, using the vector line equation the intersection point between a vector and surface,  $P$ , can be defined as,

$$P = P_0 + vb \quad (5.26)$$

Where,  $P_0$  is the position of a point on the line,  $v$  is the direction vector and  $b$  is a scalar parameter. With,

$$P_0 = (x_0, y_0, z_0) \quad (5.27)$$

$$b = \frac{P_s \cdot v - P_0 \cdot v}{P_s \cdot v} \quad (5.28)$$

Where  $P_s$  is a point on the surface being checked for an intersection. If  $b$  is undefined, the vector and surface are parallel with no intersection. In all other cases, barycentric coordinates identify if the intersection is within the surface constraints.

Determining if the nodes added to the gaps formed by the intersection points would be free or internal involves consideration of the number of intersections by the skewer considering all surfaces. Figure 5.13 presents two cases for a given geometry where an odd number of intersections from a point signifies it is internal, and an even count shows it to be external. Using this geometric rule, if a gap has an even number of intersections on one side, it is external and a series of free nodes should be defined along its length according to the mesh spacing used in the algorithm. This is shown in Figure 5.12 plot iii, where red nodes are added outside of the geometry.

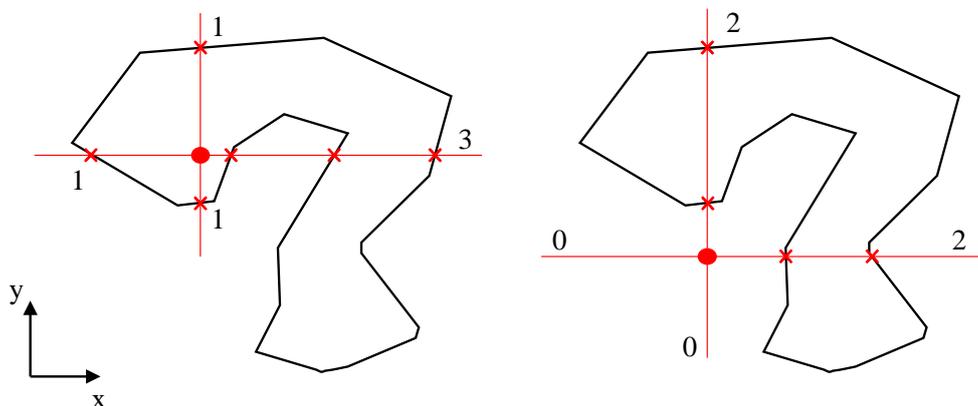


Figure 5.13: Example of how a point can be classified as internal or external by projecting a line with sufficient length in any direction, checking the number of obstacle intersections, and determining if the count is odd or even. An odd number of intersections relates to internal points (left), even for external (right). This rule applies in 2D and 3D.

### 5.6.4 Connection accuracy: Clipping

It is essential that the possible travel paths for the wave are accurate to the physical problem being modelled. Connections passing through rigid obstacles are therefore not possible and should not be included in the graph being analysed using Dijkstra's algorithm.

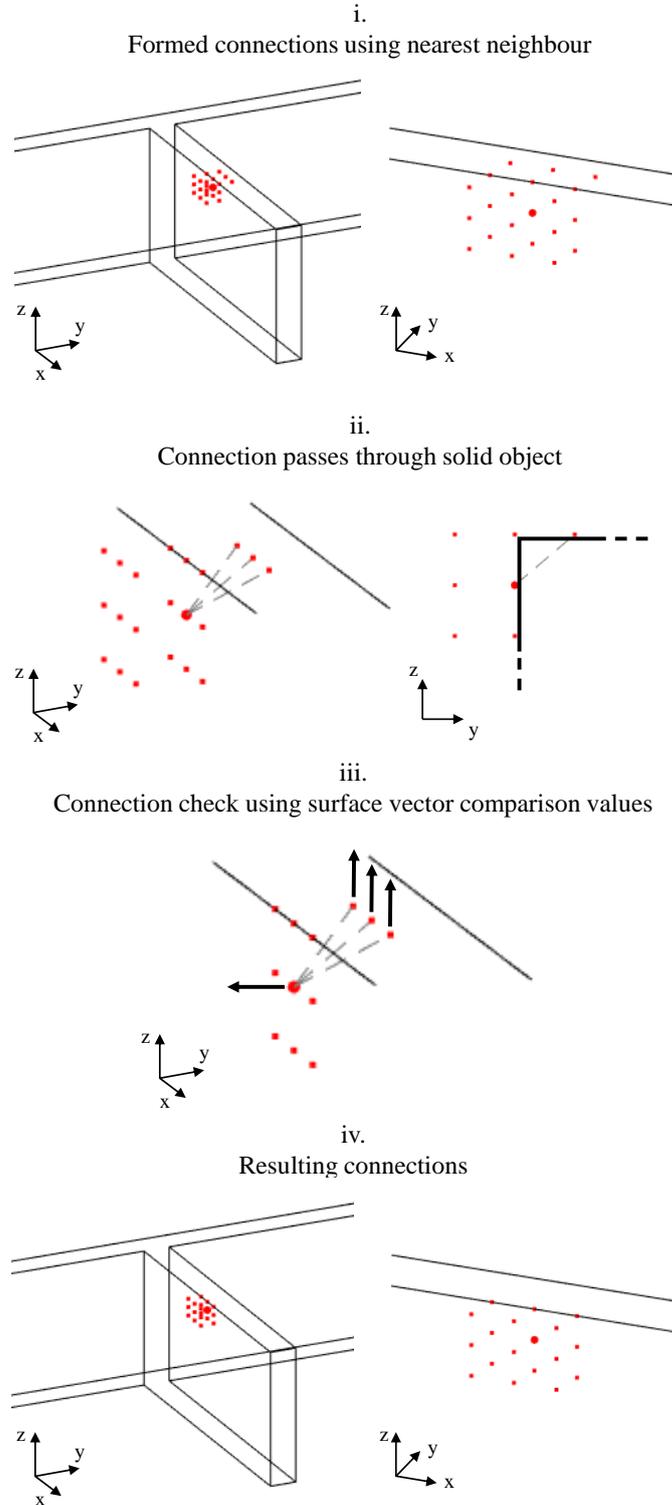


Figure 5.14: Method of preventing connections passing through a rigid obstacle.

An issue that arises due to the use of the nearest neighbour connection approach is shown in Figure 5.14 plot i. It shows that an influence on one surface can form a connection to another influence on an adjacent surface by passing through the corner of the structure, because the distance between both points satisfies Equation 5.24.

To resolve this ‘clipping’ issue, the outward normal surfaces vectors, stored with every surface influence entry, can be compared. If two connected surface nodes share the same surface vector, the connection is formed. If they do not, it is removed. Plots iii and iv of Figure 5.14 show how this affects the connections in this example to preserve a physically accurate geometric representation of the domain.

### 5.6.5 Path optimisation

Despite successful implementation of the SPA method for generating wave travel paths to each influence, Figure 5.15 highlights how the identified routes can include obscurities associated to the traversable node grid spacing. In this case, sudden  $90^\circ$  movements are noticeable as the wave path approaches the target surface influence.

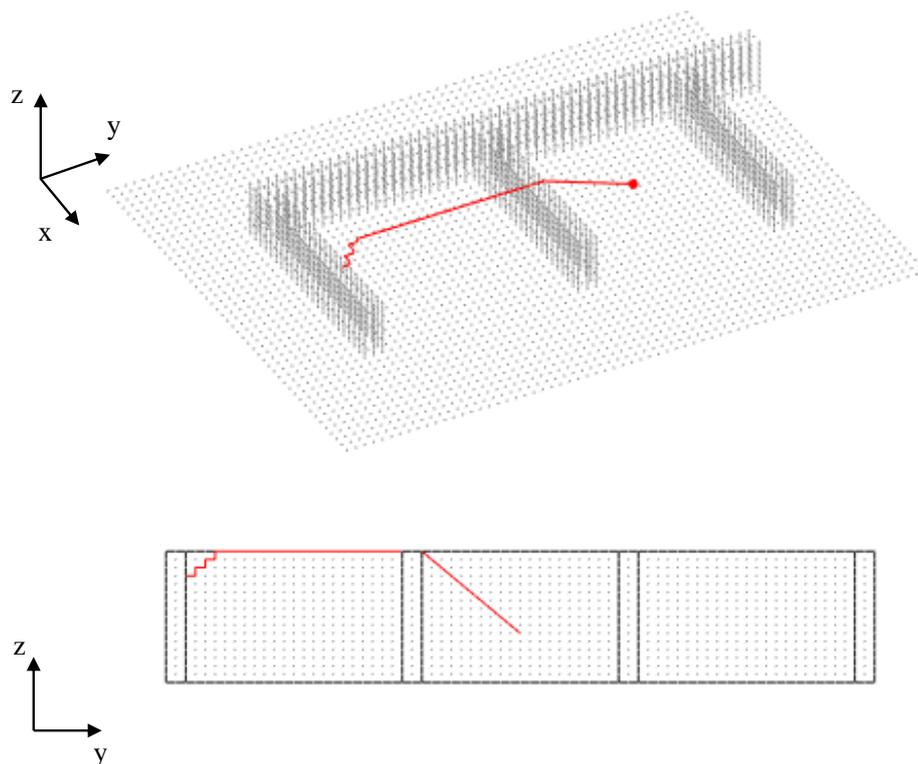


Figure 5.15: Unoptimised path obtained using Dijkstra’s algorithm. Charge shown as red circle. Influences shown as grey dots. Path shown as red line.

Two features for optimising the Dijkstra’s shortest paths to produce more accurate shortest paths have been explored in the following sections. The first being named ‘type run’ and the second, ‘intermediate path storage’.

### Type run optimisation

Type run optimisation of the shortest paths requires each visited node type to be stored so that any uninterrupted series of the same type can be enhanced. Node types include free nodes and obstacle or boundary influences (vertex, edge and surface), with Figure 5.16 presenting these classifications for the unoptimised path shown previously in Figure 5.15.

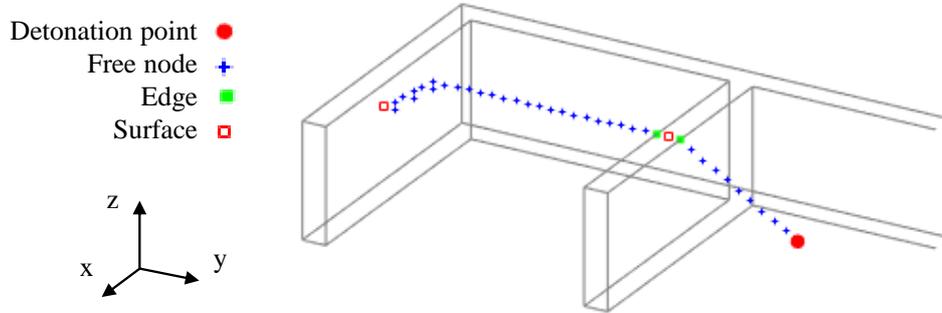


Figure 5.16: Unoptimised path influence types.

If three or more of the same type are included in a sequence in any identified route, then there must be a direct line of sight between the first and last points of the run. This is because the node connection rules defined in this Chapter prevent any node from forming paths beyond its 26 surrounding nodes, hence preventing an identified path from skipping over nodes in-between the target and the charge.

This means that for any run of the same node type, only the first and last nodes need to be included when considering the wave travel distance. Figure 5.17 shows how optimisation of the example path helps to remove the angular movements in the free node section of the route. Ultimately providing a more accurate path direction and reducing the wave travel distance to be closer to the true minimum value, as shown by Figure 5.18.

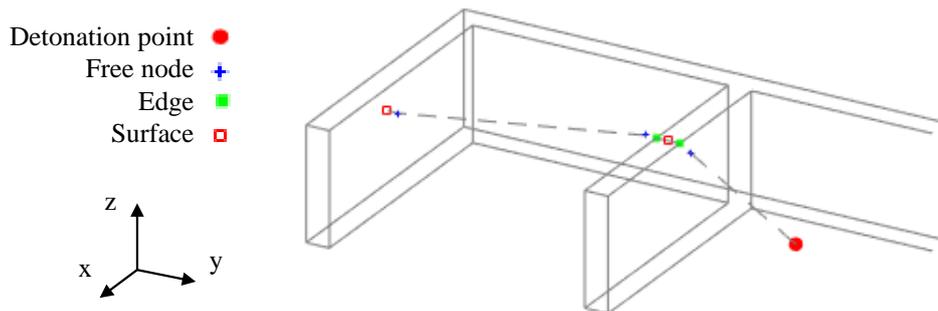


Figure 5.17: Optimised path influence types.

### Intermediate path storage

The second feature included with generating the comparison metric for 3D blast models is intermediate path storage. Figure 5.19 shows an example of a type run optimised path for the wave being tracked to a target surface influence. On the route to this node, the wave progresses over the top of the containment structure, visiting two edges and one surface influence. With intermediate path storage, the routes and distances up to these

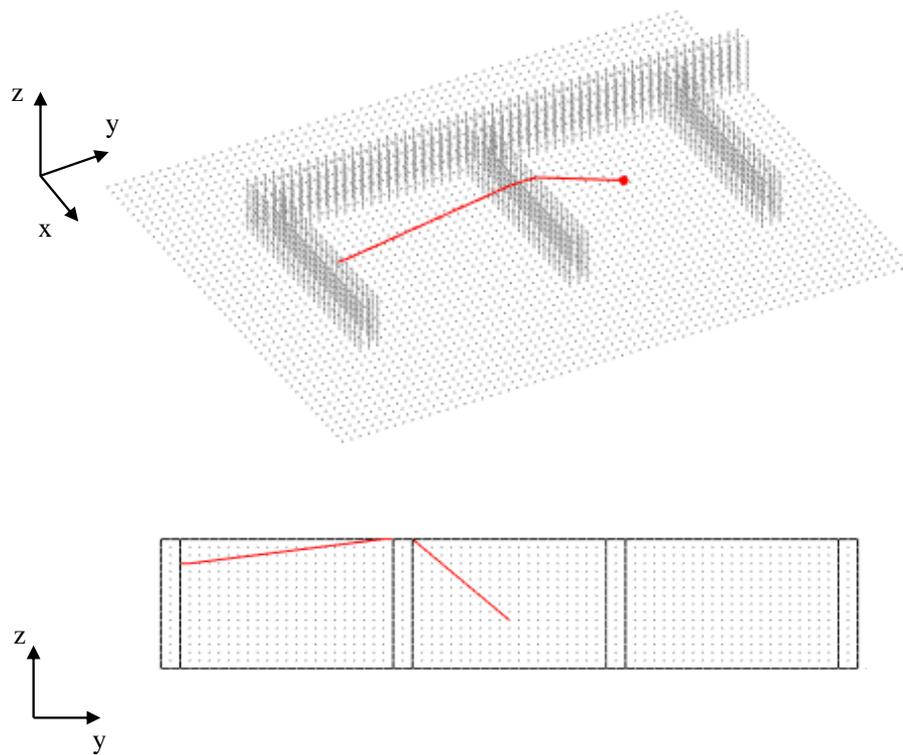


Figure 5.18: Output path obtained using Dijkstra's algorithm and type-run optimisation.

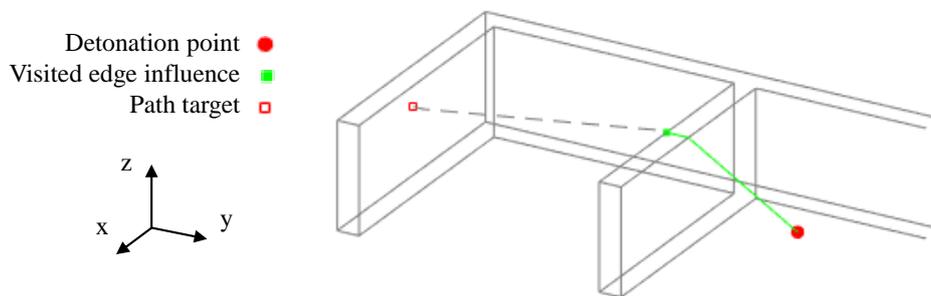


Figure 5.19: Example of a shortest path for a visited influence that is identified when assessing the optimised path of a different influence.

nodes are stored as the comparison metrics for these influences during the analysis of the original target surface. Hence, preventing the need to analyse the same sections of Dijkstra's output multiple times. The green line therefore shows the part of the optimised path that is saved for the second edge that is visited.

For additional computational efficiency, the path optimisation methods are applied to the influences that have the greatest Dijkstra's shortest path first, as this ensures a high number of intermediate routes are found and stored.

## 5.7 Alternative methods to represent a domain

This section introduces a number of ideas that were tested as alternatives to the adopted approaches discussed so far in this chapter. They provide key benefits that make them suited to rapid implementation, however, issues related to accuracy and robustness have limited their development for use with the BA. Despite this, the concepts may be suitable for use with other rapid analysis tools.

### 5.7.1 Comparison metric: Ray tracing

Ray tracing is a graphical technique commonly used in many modern video games and animated films that involves tracing a ray of light around the plane of an image to render a virtual obstacle with realistic lighting (Christensen et al. 2018). It is a process that considers various effects such as reflection, refraction and diffraction to mimic the physical process that light experiences with the aim of generating a photo-realistic image of a given obstacle or scene (Christensen et al. 2006).

Aside from image rendering, ray tracing has also been adapted for applications such as X-ray diffraction tomography and multi-axis machining simulations (Ulseth et al. 2019, Jachym et al. 2019). Not only this, but it has been used successfully in blast wave analyses for assessing the impact of explosively driven fragments on electric substations, and calculating pressure waveforms in urban environments (Roybal et al. 2009, Frank et al. 2008).

However, considering the potential application of the technique for this thesis, the complexity of the fully developed approach that is used to form pressure profiles in these articles is not required. Instead, the ideas of ray projection, intersection and reflection could be modified to assess the path that a wave would take in reaching a given point in a model's domain.

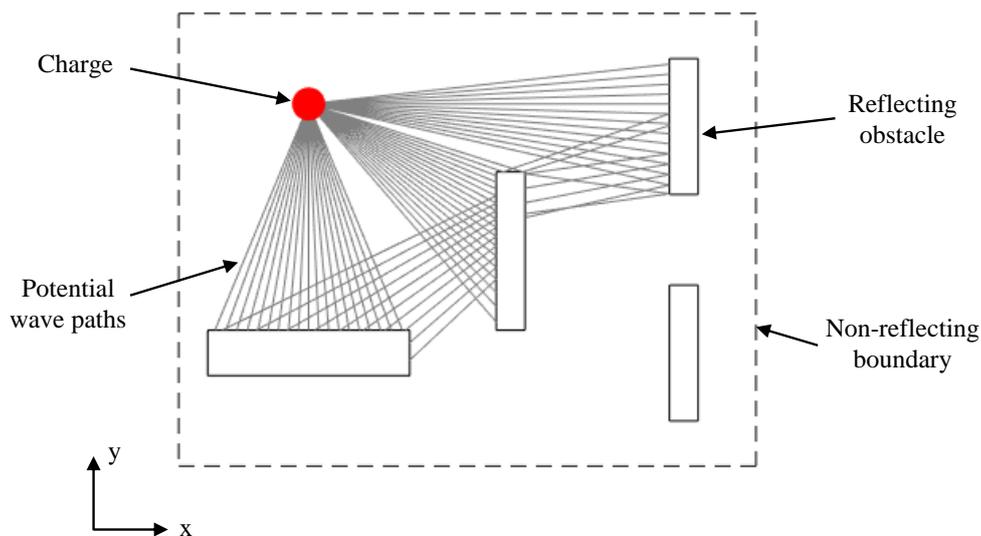


Figure 5.20: Example of how ray tracing in 2D can be used to identify wave travel paths.

To determine the applicability of this idea, Figure 5.20 displays the preliminary results from the experimental implementation of ray tracing in 2D with the charge acting as the origin of all potential wave paths. Rays were projected with vectors that were incremented by one degree to emulate the omnidirectional expansion of the blast wave. Then, if a ray intersected with a rigid surface, it was cut and a new part of the ray was formed with a vector that represented a perfect reflection. If no intersection occurred, the ray was removed.

Seeing as this process does not consider more complex interaction processes such as diffraction and refraction, it is fast running and can be used to effectively represent the immediate surroundings of the charge. This could be used to inform the degree of confinement of the environment, or to relate blast effects to a point of interest based on its location. Additionally, distances between surfaces that are not in direct view of the charge can be derived in a way that does not compromise the physical constraints of the obstacles in the domain.

However, it is clear that not every point on each obstacle is reached by a ray. For a 3D application of the BA, it is essential that each discretised influence is assigned a relevant comparison metric, but with ray tracing this cannot be guaranteed. Increasing the number of rays from the charge, by decreasing the angular spacing of the vectors projected from it, could improve the number of assigned travel distances, but some influences will still be omitted. The progression of this approach to 3D was therefore not tested considering that these issues would remain prevalent.

### 5.7.2 Domain representation: Voxelisation and skewering

Octrees are a hierarchical data structure that is used to spatially sub-divide a geometry into 3D ‘voxels’, with applications in autonomous robot navigation (Meagher 1982, Hornung et al. 2012, 2013). In simple terms, voxels are cubes that represent if the space inside of it is empty, filled or partially filled. Using this information a robot is able to decide if it can move into a space, or if the path would be blocked. This section explores the application of voxelisation of domains used with the BA to determine if computation time can be saved when discretising a 3D geometry.

In this adaptation, the same influence types discussed in Section 5.2 must be assigned to each voxel being defined in each geometry. Similarly, free voxels must be included to enable the wave travel distance comparison metric to be generated using the shortest path approach. Figure 5.21 shows how a domain could be represented by filled voxels using an adapted skewering method based on Section 5.6.3. This Figure is presented in 2D, using a large voxel size to clearly show the simplification effect.

In addition to using the skewers to find the external sections of the model, they are also used to categorise the internal and surface points depending on the position in the domain relative to the free space. If a voxel is identified to be on the interface between external and internal points, it is given the surface or edge designations. Any voxels in inside obstacles are not included.

This representation is also similar to how numerical models commonly mesh a given domain using a set cell size with the model geometry being fit to the cell positions. This results in the model being slightly shifted in some cases if it does not coincide with the cell structure, however the alterations are often so small that no significant impact is observed in the analysis.

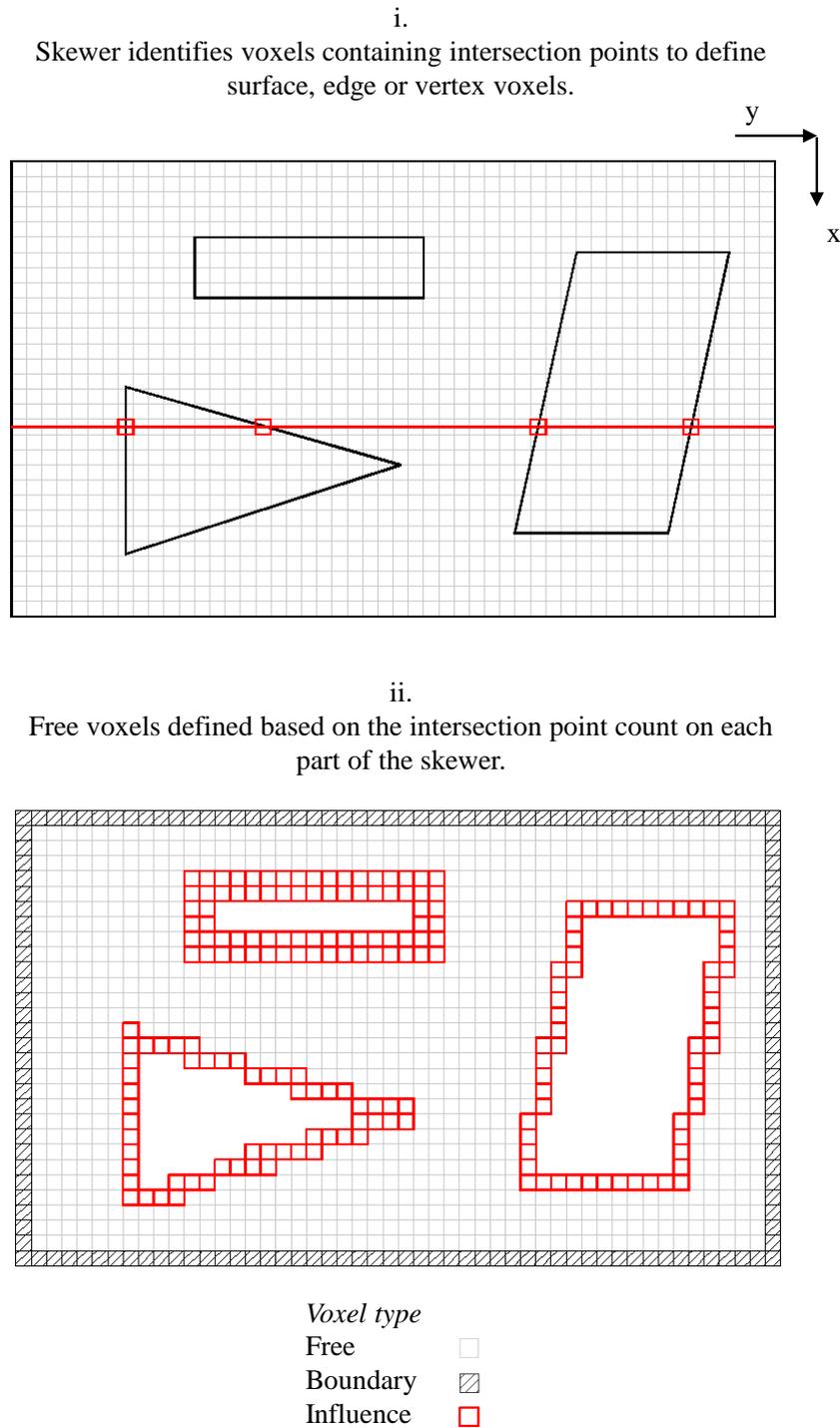


Figure 5.21: Voxelisation using the skewering approach discussed in Section 5.6.3. Plot ii utilises the internal/external point check shown in Figure 5.13 to define free voxels.

For the BA, the processing steps required to generate the comparison values means that to keep the code fast running, larger cell sizes, or mesh spacing as it is termed in this thesis, is required. Simplifying the geometries of each model therefore acts as both a positive and negative of this approach, because there will be a loss of geometrical complexity. This can lead to incorrectly identified deviation points in models where surfaces are in the same location relative to the charge, but the voxel representations differ, as shown in Figure 5.22. In this example, the panels of each domain have equal stand-off distances from the same charge, the only difference is the position of the arrangement in each respective domain. The shortest path analysis of the wave travel paths tracks to the centre of each voxel and so using a voxelised approach to simplify the geometry leads to differing voxel representations which in turn gives incorrect deviation conditions.

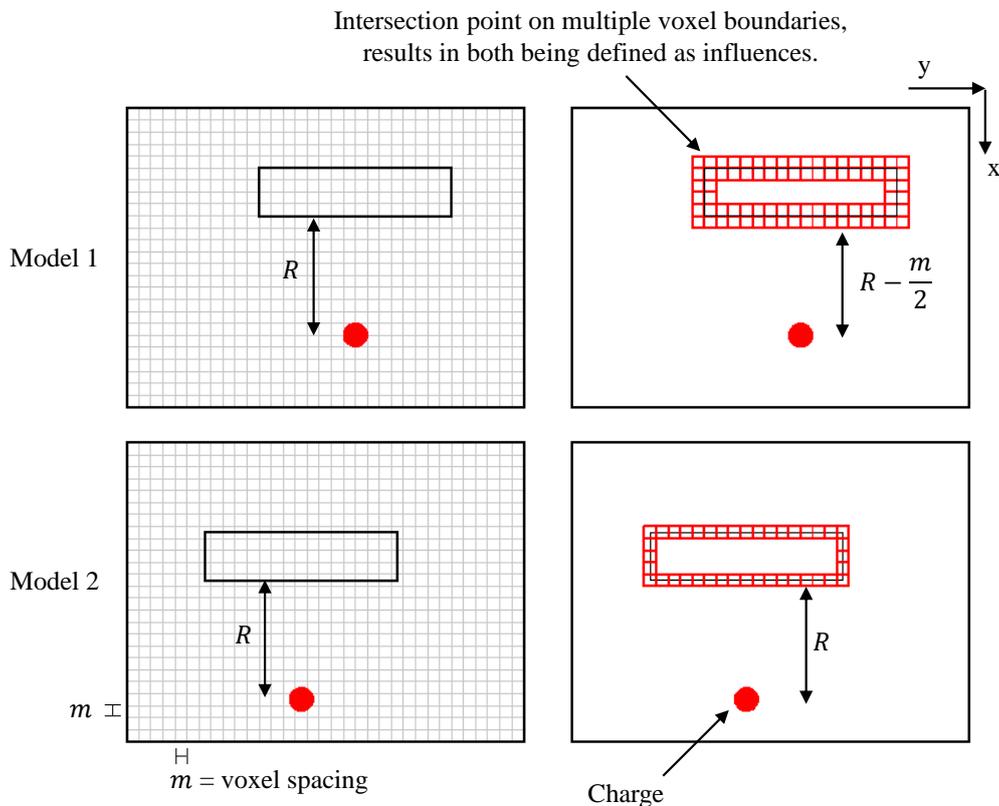


Figure 5.22: Voxel spacing results in different wave travel distances even through the models have the same stand-off in the un-voxelised form. Leads to a deviation at the incorrect time, reducing the BA efficiency. Free voxels omitted for clarity.

Similar to the discrete node method presented in this chapter, the voxels themselves are defined in a global sense. Every model of the batch includes voxels in the same positions of the domain. However, Figure 5.22 shows that once they are categorised, they can lose the global benchmark when the model geometry and charge are not aligned with the voxel boundaries in a specific way. The discrete node method averts this problem by defining influences globally in a dominant axis only, leaving the remaining axes to accurately position the influences relative to the charge.

In this example the deviation of model 1 from 2 as the wave hits the closest influence voxel would not result in physically incorrect parameter fields in either domain, however it ignores a large portion of repeated calculation steps that could have been removed through more considerate data mapping. The algorithm runtime improvement that could be achieved by implementing the voxelisation method is likely to be cancelled out by this effect. Thus, no further development of this idea has taken place, yet the concept of voxelisation provides a means of rapid domain discretisation that may benefit alternative analysis tools.

## 5.8 Interfacing with the Branching Algorithm

The BA is coded in the open access programming language Python, with all functions being written using free to download packages. It is initiated by a single text file of a standard layout (example shown in Appendix A), with the user only being required to provide the name of the file to the script that executes the sorting process.

This ‘read-in’ approach is similar to other computational software such as APOLLO Blast-simulator, where the user is given the option to create a model using the graphical user interface, or with text inputs alone (Fraunhofer EMI 2018). Combining the latter with the windows command line brings the advantage of simulating a series of models one after another with each new text file’s simulation conditions being used to initiate various models from birth.

An example of the output from the BA is given by Table 5.2. Each row contains a model number and its associated deviation condition, in addition to parent model number and any dependants. This simplified output limits the amount of information that needs to be passed to the numerical solver whilst also allowing for the controlling script to easily handle incoming data that will need to be merged based on the progression of each simulation.

Table 5.2: Example output mapping table from the BA for use with the numerical solver.

Model No.	Parent model	Deviation location [ $x$ , $y$ , $z$ ]	Dependants
1	4	[ 0 , 8.5 , 2.5 ]	~
2	5	[ 0.5 , 2.5 , 0 ]	~
3	~	~	5
4	5	[ 0 , -8.5 , 2.5 ]	1
5	3	[ 0 , 2.5 , 0 ]	2,4

## 5.9 Additional features

### 5.9.1 Probabilistic grouping

Seeing as models will produce varied parameter fields from birth if various input conditions are different, probabilistic grouping allows for multiple trees to be created when running the BA. This ensures that the maximum amount calculation steps can be removed from the overall analysis, utilising informed data mapping in as many domains as possible.

It is expected that most uses of the algorithm will only require one probabilistic group. However there may be analysis cases where multiple explosives need to be considered when assessing the risk to a given target and so the results from all tests may need to be gathered in one dataset produced by the algorithm for further risk assessments.

### 5.9.2 Domain scaling

Hopkinson-Cranz scaling is supported by the BA to allow models with different charge sizes to benefit from informed data mapping provided that the charge composition remains the same. This aims to maximise the amount of repeated calculation steps being removed from the batch of simulations.

Using this relationship between blast parameters, multiple model domains can be scaled to the size associated to a 1 kg mass of explosive. The BA and all simulations can then run with the scaled parameters before the outputs are unscaled for use in additional analysis.

### 5.9.3 Viper::Blast integration

As discussed in Section 3.2.3, the numerical solver Viper::Blast features trigger gauges that can be positioned in a domain with specific conditions allowing them to either terminate the current model, or output a remap files of the parameter space as the gauge is contacted. With the implementation of these features, the BA is able to function seamlessly with the solver since gauges can be positioned in each relevant domain at the specified deviation locations when evaluating the output from the BA. Integration with alternative solvers may require changes to the coded method to ensure that files are output at the correct time steps.

## 5.10 Application: Containment structure analysis

### 5.10.1 Problem scenario and model specification

The following application of the BA in 3D is included to show that the developments explained in this chapter can lead to meaningful reductions in the computation time required to analyse batches of models.

Figure 5.23 provides a plan view of the five geometries that feature in a batch of 20, 3D models that aims to replicate a study where the effect of reducing the material use of geometry 1 is evaluated in terms of the overpressure recorded at surrounding locations following an accidental detonation. Each containment structure is 4 m tall, with base dimensions covering  $18 \times 5.5$  m.

The locations of the gauges used to monitor the pressure variations are presented in Figure 5.24 alongside the four charge locations that are used independently with each geometry. Furthermore, the simulation domain that was adopted for use in the chosen solver, Viper::Blast, is also provided. The floor is defined as the only reflecting surface with all others being able to transmit the blast wave outside of the domain.

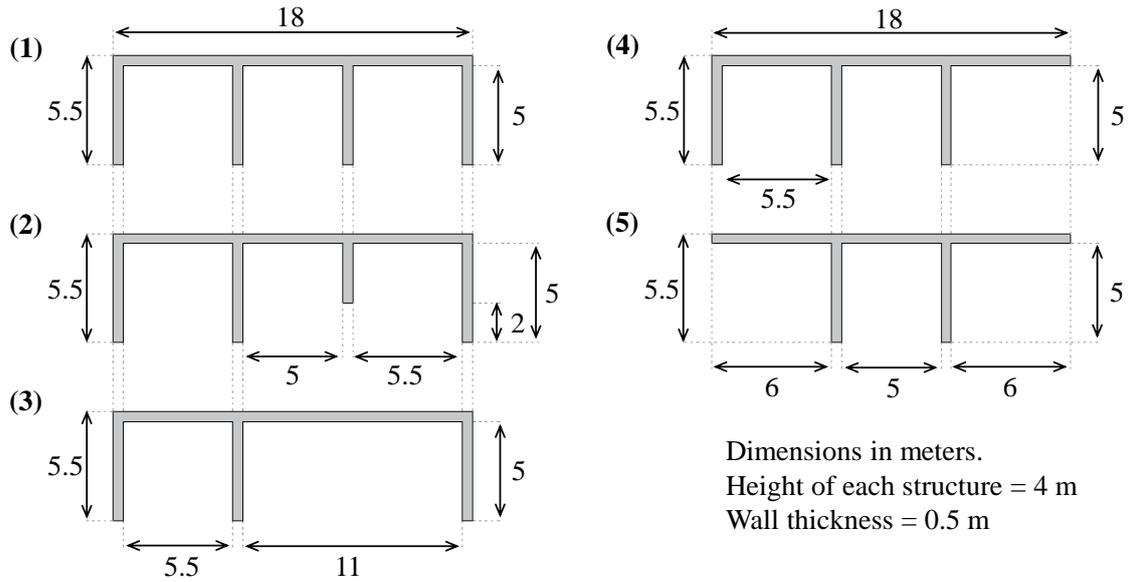


Figure 5.23: Plan view of the five containment structures included in the example analysis.

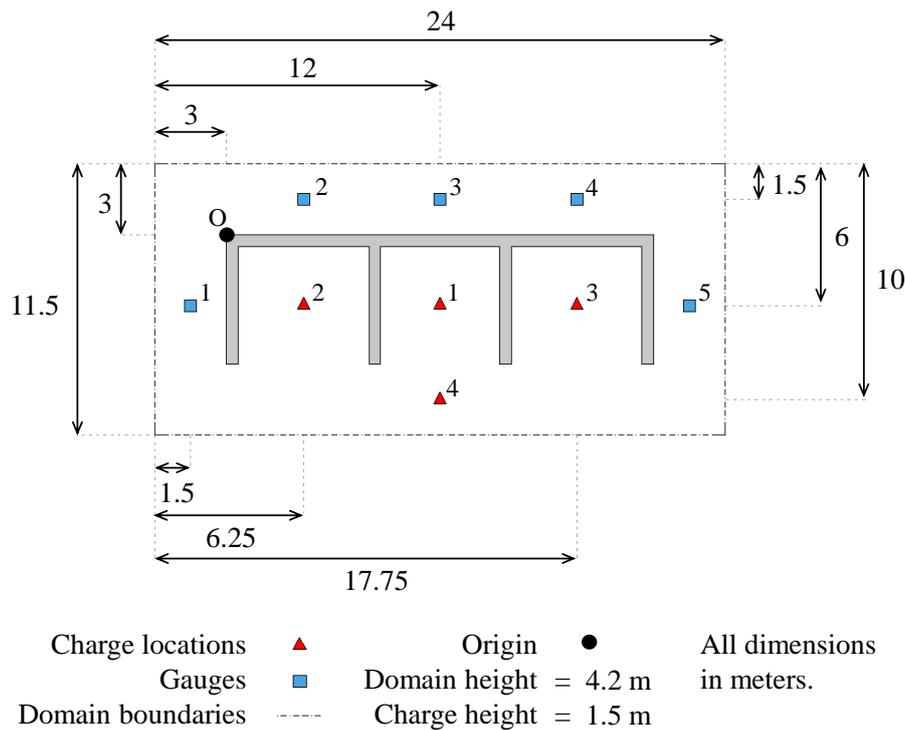


Figure 5.24: Plan view of the modelling domain, showing the origin point of the containment structures in Viper::Blast, four potential charge locations, and five gauges used to record pressure-time profiles.

In each case, the Viper::Blast models are simulated using an ideal gas calculation with a 5 kg, TNT, spherical charge positioned 1.5 m above the ground and a termination time of 55 ms. Ambient pressure and temperature are 101325 Pa and 288 K respectively. Simulations including the initial detonation utilised 1D to 3D mapping that takes place one cell before the wave reaches a rigid surface.

For all models, the element size of the 1D stage is specified as 0.002 m to provide  $\sim 45$  elements across the spherical charge radius for the simulation of the initial detonation. This stage is evaluated until the shock wave is one cell away from the rigid boundary of each domain at  $z_{\min}$ . The element size is increased to 0.04 m for the 3D solving stage to comply with the requirement outlined in Section 3.2.6 for a maximum 1D cell size of 0.0045 m, and maximum 3D cell size of 0.045 m. Whilst it is important to respect these mesh restrictions it is the overall computation times and outputs between branched and non-branched methods that will be compared. Therefore, provided that conditions remain the same for both approaches, conclusions will remain valid.

### 5.10.2 Algorithm output

Figure 5.25 displays the output from the BA., where four groups are considered based on the charge location. Red lines on each model show the path of the blast wave that emanates from the charge to the uniquely identified deviation points. For lines that pass through the containment walls, the wave is progressing up and over the 4 m structure. Domains including no red line are the trunk models of each group.

It should be noted that it is the final deviations being shown for each model in each tree. For example, reviewing the output for the first tree, model 5 deviates from the trunk model as the blast wave reaches the right hand wall of the central bay. No other model shows this to be a deviation point, however every other model also deviates from the trunk model at this step as shown by the resulting deviation tree. Similarly, models 1 and 4 would both deviate at the same time step, however symmetrically opposite influences are identified to cause the deviations, resulting in an additional sub-trunk model that does not impact the algorithm's performance.

### 5.10.3 Simulation times

The overall computational saving is given by Table 5.4, with Table 5.3 showing that full simulations of all 20 models requires 20086 seconds when using an Intel i5-8265u processor, 8 GB of RAM and a Nvidia GTX 1650 dedicated graphics card. Utilising the BA and informed data mapping reduces this requirement to 16228 seconds, providing a computation time saving of around 20%.

For this complex scenario, a saving of around 20% means that an additional geometry could be assessed with all four charge locations at no extra cost when compared to the standard birth to termination simulation approach. Accordingly, an additional geometry could be explored that may benefit from the knowledge gained from the first batch at no additional computational cost.

When discussing the reported time saving it is also important to consider how the individual computation requirements for each model are variable depending on the termination time and mesh density used in the numerical calculation. If these models used a higher resolution of elements, the relative cost of the algorithm would be reduced. This ultimately drives the total percentage saving to be closer to when the algorithm cost is ignored, in this case, moving 19.2% closer to 20.8%.

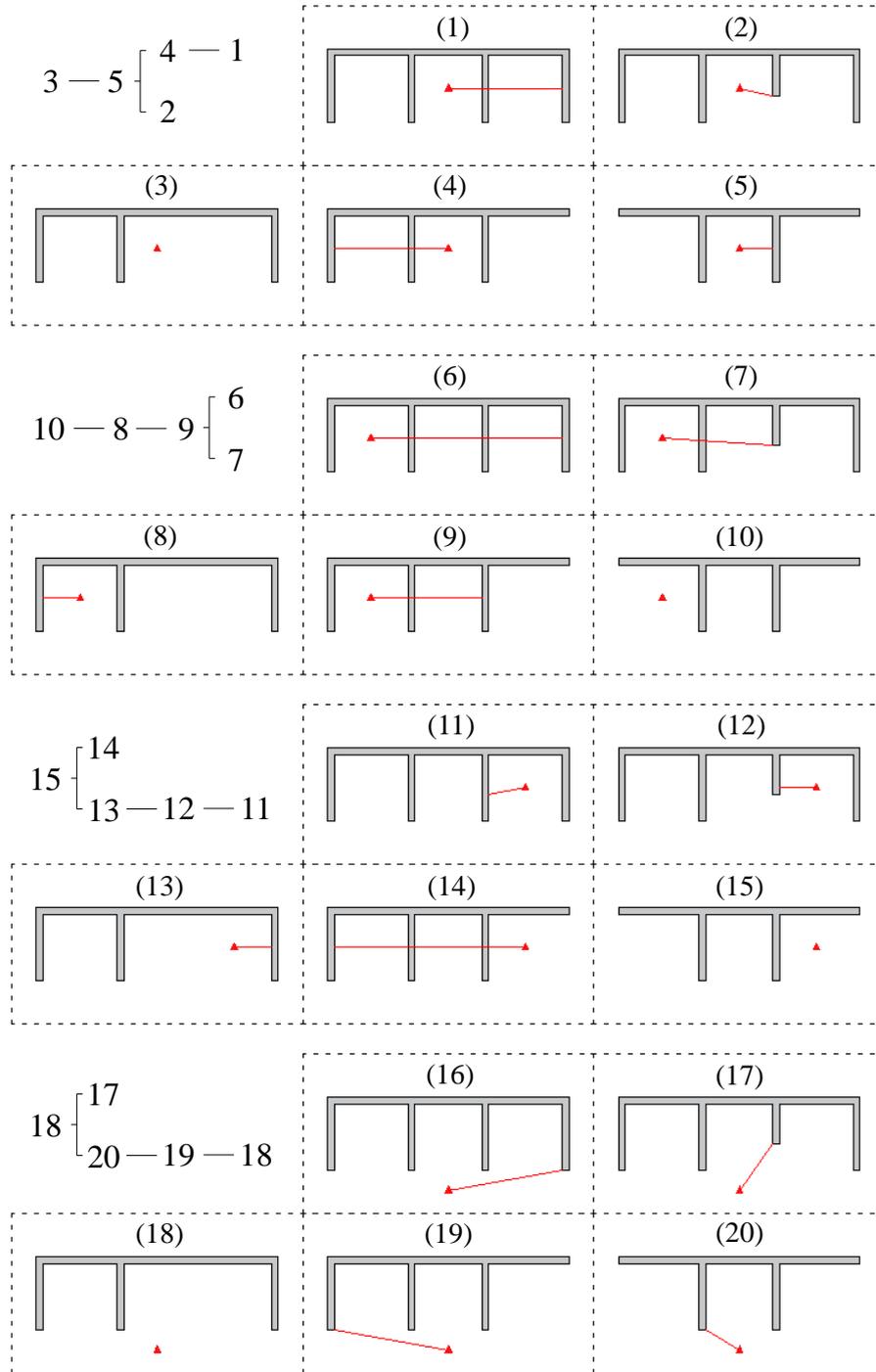


Figure 5.25: Generated deviation trees for each group and visual representations of the deviation points for each model. Red lines show the blast wave travel path to the first influence that results in a parameter field that is no longer identical to the parent model.

Furthermore, this sample study only contains 20 models. If instead, it featured upwards of 100 different arrangements with various combinations of geometry and charge size, it would be possible for a greater amount of saving to be observed with each model having a higher chance of sharing certain simulation steps with another in the batch.

Table 5.3: Comparison of model computation times.

<b>Model No.</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Tree 1</b>
Without branching (s)	1039	1011	992	1030	1030	5102
With branching (s)	595	977	992	594	1001	4159
Saving (s)	444	34	0	436	29	943
Saving (%)	42.7	3.4	0	42.3	2.8	<b>18.5</b>
<b>Model No.</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>Tree 2</b>
Without branching (s)	1015	1009	1015	1013	986	5038
With branching (s)	351	599	971	603	986	3510
Saving (s)	664	410	44	410	0	1528
Saving (%)	65.4	40.6	4.3	40.5	0	<b>30.3</b>
<b>Model No.</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>Tree 3</b>
Without branching (s)	1013	1014	994	993	987	5001
With branching (s)	974	968	954	334	987	4217
Saving (s)	39	46	40	659	0	784
Saving (%)	3.8	4.5	4.0	66.4	0	<b>15.7</b>
<b>Model No.</b>	<b>16</b>	<b>17</b>	<b>18</b>	<b>19</b>	<b>20</b>	<b>Tree 4</b>
Without branching (s)	987	990	992	990	986	4945
With branching (s)	634	843	992	630	930	4029
Saving (s)	353	147	0	360	56	916
Saving (%)	35.8	14.8	0	36.4	5.7	<b>18.5</b>

Table 5.4: Comparison of model computation times for the batch.

Full simulation computation time	20086 s
Algorithm computation time	313 s
Branched method computation time (exc. algorithm)	16228 s
Total saving (exc. algorithm)	20.8%
Branched method computation time (inc. algorithm)	15915 s
Total saving (inc. algorithm)	19.2%

Similarly, the solver Viper::Blast utilises Nvidia GPU processing in addition to various optimisation methods that significantly reduce the time required to analyse any given model when compared to alternatives exclusively featuring CPU processing power. Viper::Blast is therefore already likely to be a less time consuming approach to conducting batch analyses and so the savings noted here are expected to be far greater if less optimised solvers were to be used instead.

Regarding the implementation of the BA and data mapping, the algorithm could have used all models in one tree rather than splitting them into four separate trees. This would have increased the number of simulation steps being removed as the initial detonation of the charge, that was common to all models, would only need to be simulated once whereas here it was done four times. However, the sorting process used by the algorithm runs more efficiently with smaller trees because fewer model to model comparisons are required when defining the trunk model. Ultimately, this improved algorithm efficiency outweighs

the potential saving of modelling the detonation once because the 1D solution used for this stage is already rapid in its execution.

#### 5.10.4 Mapping results

Assessing the output from the BA in Figure 5.26 shows that the branched output for gauge 2 in model 9 provides a Mean Absolute Error (MAE, Equation 6.2) of 0 kPa when compared to the output from the same gauge from a simulation that runs without informed data mapping. The reduction in simulation time from 1013 seconds to 603 seconds in this instance is therefore achieved with no detrimental impact to the simulation accuracy. This conclusion remains true for all domains in the batch due to how mapping takes place just before each associated deviation would occur in the corresponding parent model.

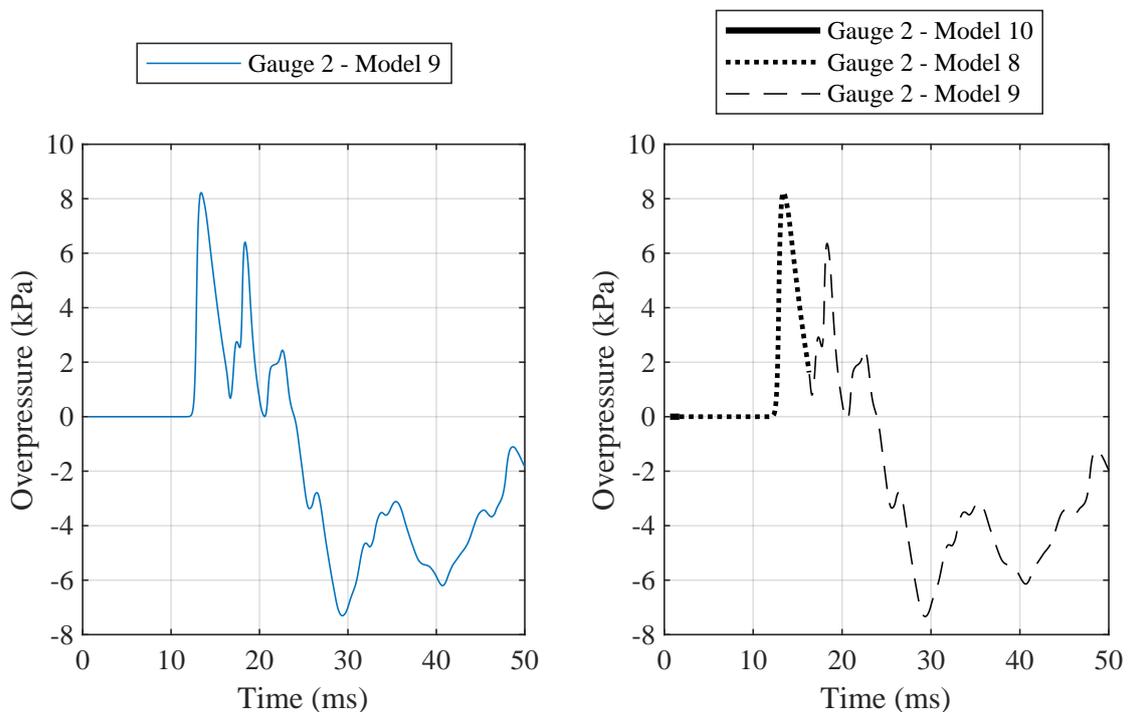


Figure 5.26: Pressure-time history recorded by gauge 2 when simulating model 9 entirely in its own domain (left) compared to when informed data mapping is used (right). Each line type shows which domain the data was record in.

Whilst it was not the aim of this paper to draw conclusions about the containment structures considering the magnitudes of the simulated results, with the focus instead being placed on proving the computation time saving potential of the BA, it is important to appreciate the scenario where the batch analysis approach is useful. Table 5.5 therefore provides the peak overpressure at each gauge for each containment structure (shown in Figure 5.23) considering all four potential charge locations when defining the peak overpressure at each gauge.

An assessment of these readings clearly shows how the variation is small for gauges 2, 3 and 4. However, gauges 1 and 5 highlight how structures 4 and 5 may result in an unacceptable risk of increased overpressure either side of the containment model. Containment 3 appears

Table 5.5: Comparison of the peak overpressure recorded at each gauge for each containment structure considering all charge locations.

Containment Structure	Peak Overpressure (kPa) at gauge number				
	1	2	3	4	5
1	18.53	8.22	11.13	8.22	18.19
2	18.53	8.22	9.91	8.22	18.19
3	18.53	8.22	8.51	8.22	18.19
4	18.53	8.22	11.13	8.22	82.58
5	79.16	8.22	11.13	8.22	82.58

to present the best compromise between peak values and material use, however no human vulnerability thresholds or fatality assessments, such as the ones featured in studies by Alterman et al. (2019) and Marks et al. (2021), have been consulted or conducted in making this judgement. Including such analyses would be required in the next phase of this analysis so that the data from all 20 models is utilised to make more informed risk based decisions or in designing a new alternative structural arrangement that benefits from these observations.

## 5.11 Summary

This chapter has developed new influence definitions that allow for the Branching Algorithm to be applied to batches of 3D domains. By considering the shortest wave travel distance, vertices, surface nodes and edges, the algorithm is able to discern when each simulation deviates from the others using a logical sorting process provided in the previous chapter. It is shown that when applying this method to a complex set of 20, 3D containment structures, the computation time can be reduced by approximately 20%. This is achieved, once again, with no loss of output accuracy, resulting in *Objective 3* of this study being met.

A key advantage of utilising this method is therefore that it provides the researcher with the ability to conduct a more robust analysis of various problems with the same computation expense. Thus benefiting the analysis of probabilistic frameworks, or development of machine learning training datasets to ultimately promote more effective design optimisation and exploration. The ability to remap data from one domain to another relative to the centre of an explosion lends itself well to transferring data at time steps where a deviation in the outputs will occur, however, the magnitude of the improvement to the required computation time will vary depending on the application and complexity of the batch of models being simulated.

For example, the application included in this chapter aimed to provide representative time savings for a complex structural layout with charges placed at relatively low stand-off distances. Yet, scenarios featuring larger stand-off distances and a greater amount of structural similarity, such as those discussed in Chapter 4, have been shown to save up to 50% of the computation time. Nevertheless, each batch that is processed by the method will result in time savings that compound with each additional use.

The key contribution of this chapter is that it provides a robust, reliable and repeatable method for formalising efficient remapping of numerical modelling, by identifying a main ‘trunk’ model and sorting each subsequent model according to when its results are expected to deviate from that model. Whilst this, and the previous, chapter focusses on blast analyses, future applications could look to tailor and develop the algorithm for different numerical problems, where batches of models are equally as important for understanding the processes being models. This includes vehicle design and safety assessments (e.g. Lu et al. (2020); Wu et al. (2020)), search and rescue tools (e.g. Coppini et al. (2016)), pressurised water pipe failure (e.g. Barr et al. (2020)) and structural topology optimisation.

It should be noted that this chapter and the previous were combined, and condensed, to form the published article titled ‘A branching algorithm to reduce computational time of batch models: Application for blast analyses’.

# Chapter 6

## A Direction-encoded Framework for Machine Learning Tools

### 6.1 Introduction

It was discussed in the literature review of existing FREMs (Section 2.6) that Artificial Neural Networks (ANNs) can be successful in predicting values associated to various task-specific blast events. However, the inputs of these tools often limits their use to a select range of pre-defined scenarios that do not allow for probabilistic studies to explore the impact of varied charge locations or domain geometries.

Consequently, the challenge for improving this approach for blast wave analysis is to engineer an input pattern that represents the relevant domain geometry in addition to the prediction point and charge location. Thus, preventing information from being trapped in the tuned network parameters. Using knowledge of how a blast wave interacts with the surroundings on its journey towards the point of interest (POI), the Direction-encoded framework for Machine Learning tools, introduced in this chapter, aims to solve this problem.

The initial idea is firstly tested using an architecture that matches existing literature, before each step of the feature engineering process is explained and tested. This aims to highlight the importance of understanding blast wave mechanics when constructing an ML tool whilst also showing how each input adaptation builds upon the predictive accuracy of the previous model. Once the inputs are confirmed, the hyperparameters are optimised.

### 6.2 Introduction to the Direction-encoded Framework

Considering the various mechanisms associated to how blast waves interact with obstacles, including channelling, clearing, and reflection, it is clear that the cause of varied blast properties is linked to the path that the blast wave must take to reach a POI, rather than the shape and size of the domain or the entire topology.

The input pattern utilised by the Direction-encoded Framework is therefore informed by the underlying physical processes, and structured to include the shortest wave travel path, the influence of surrounding obstacles according to 8 directional ‘lasers’ on a 2D plane, and the laser direction that points towards the charge. This initial feature set is shown in the example given by Figure 6.1, where the inputs are applied to a neural network.

Taking inspiration from robotic vacuum cleaners and how they navigate their surroundings, each directional input acts as a range finding tool that informs the model about the proximity of rigid surfaces around the POI. Robot vacuum cleaners use these distances to identify where they can go, where they cannot go, and where needs to be cleaned, allowing them to be used in any room (Chiu et al. 2009, Kang et al. 2014). However, in this application distances to rigid obstacles are calculated for each laser with a input value of 0 being assigned to those that reach the ambient boundary without obstruction.

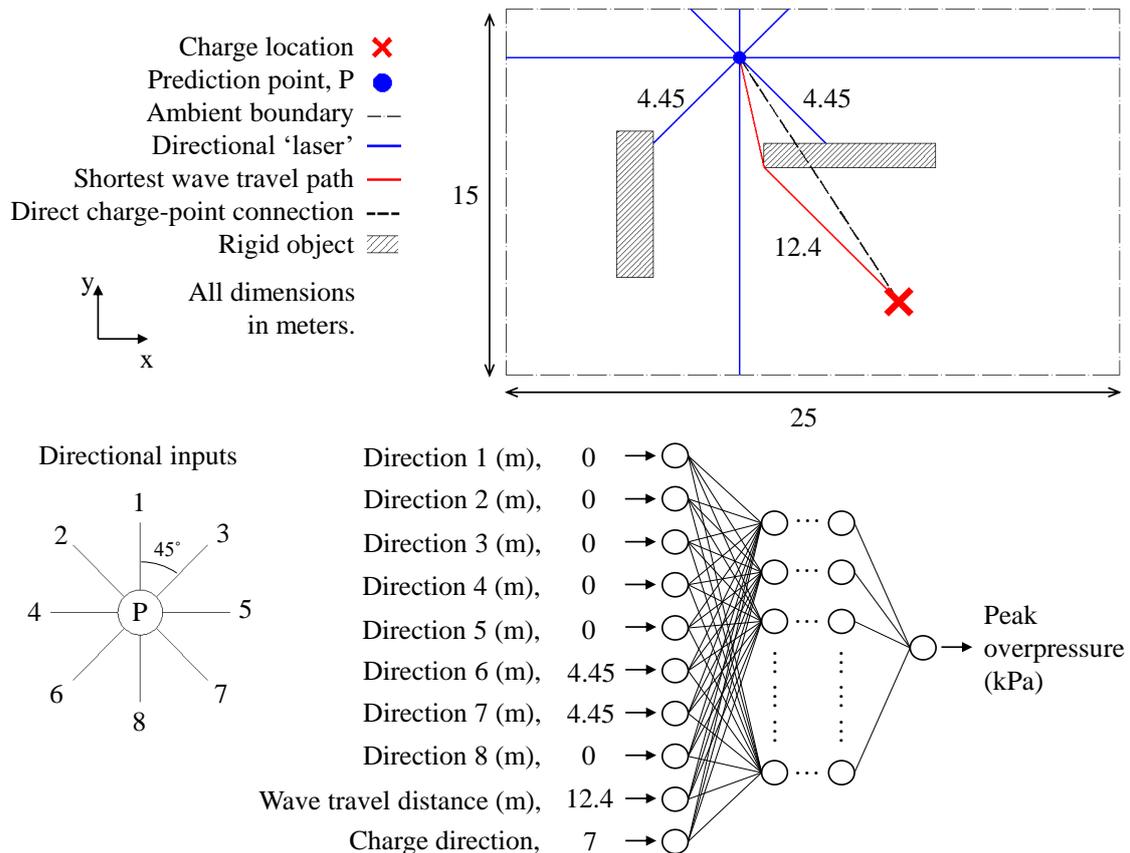


Figure 6.1: Proposed approach for generating blast load predictions using an ANN. Reference to the prediction point is made relative to the surroundings and the wave's journey.

The charge direction input relates to the directional input that is closest to the direct line drawn between the charge and prediction point. In the example shown by Figure 6.1, direction 7's laser has the smallest angular difference to the direct connection, resulting in the input of 7. This aims to show the model that obstacles being detected by lasers 5, 7 and 8 are likely to shield the POI from the wave, leading to a reduction in the peak overpressure that should be predicted.

Shortest path analysis is used to calculate the wave travel distance with a similar approach to the BA in 3D (Chapter 5). Since a training dataset needs to be generated, it is beneficial to discretise each domain being evaluated, with each point providing an independent input–output combination. This network of nodes can then be traversed using Dijkstra's shortest path algorithm to find the wave travel path to each POI. In combination with the directional inputs, this feature set aims to replicate the versatility of the automated robots when predicting the peak overpressure in domains featuring various, movable obstacles.

Development of the framework is restricted to inputs–output combinations associated to the detonation of a 1 kg spherical TNT charge, with these composition details being omitted from the model’s feature selection. This allows other domains, featuring different masses or charge materials, to be compatible with the ANN by being scaled according to equivalency factors and Hopkinson-Cranz scaling laws (Hopkinson 1915, Cranz 1926).

### 6.2.1 Adapted wave travel distance calculation

It was discussed in Section 5.6.5 that the shortest path between the charge and a POI can be calculated using Dijkstra’s algorithm and type-run optimisation for a discretised domain. This provided an efficient way for the BA to identify when points in various domains would experience a change in ambient conditions caused by the arrival of a blast wave, signifying a potential deviation from the parameter space of a parent (trunk) model.

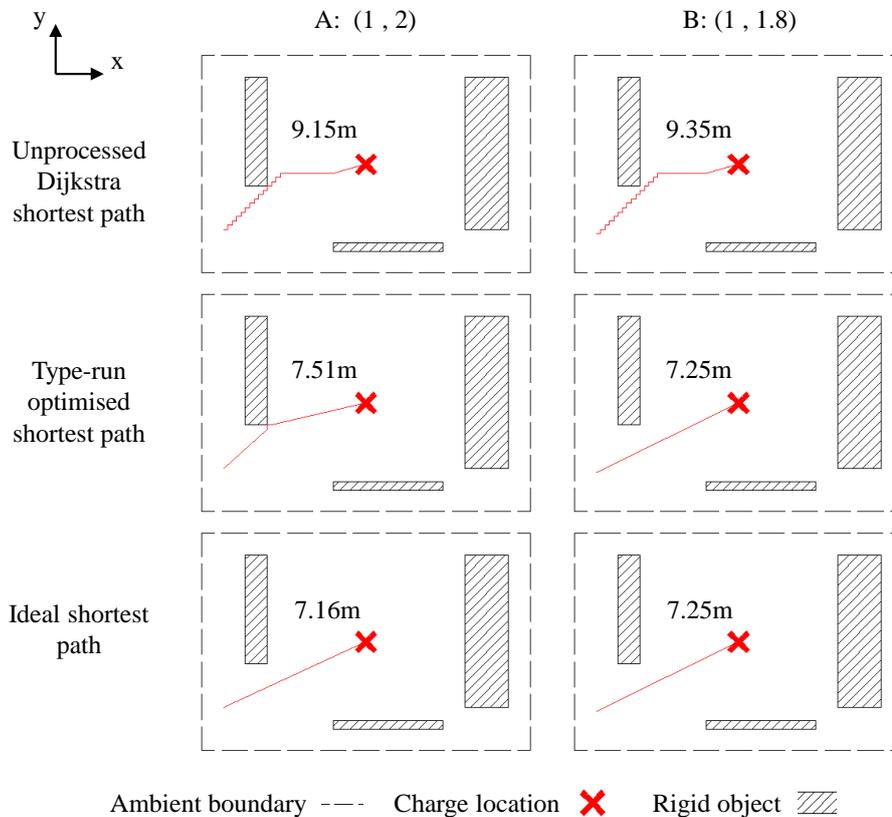


Figure 6.2: Example of the variation in wave travel distance obtained when using Dijkstra’s shortest path algorithm with type-run optimisation (introduced in Section 5.6.5). Coordinates given in  $(x,y)$  to show relative distance between points A and B for a charge positioned at  $(7.5,5)$ .

For the BA, the accuracy of this approach proved to be suitable in multiple applications. However, the arbitrary geometry, shown in Figure 6.2, highlights a potential issue with this method as points A and B both have a direct line of sight to the charge, but only point B is optimised to form this path. Instead, the output from Dijkstra’s algorithm forces point A’s path to visit a rigid panel vertex and since paths are only optimised to straight lines if every node along the section shares the same node type (free, edge/surface, vertex), the error is not corrected in the optimisation process.

Since the accuracy of ML tools is highly dependant on the quality of the inputs in the training dataset, it is essential that any non-physical or inconsistent wave travel paths are corrected. Solving this issue therefore requires the an additional smoothing function to optimise wave travel paths for all nodes in a given domain:

1. Sort nodes in the domain by the wave travel distance, from smallest to largest.
2. For each node (defined as base node):
  - (a) Find the connected nodes and the connection distances from the base node.
  - (b) For each connected node,
    - i. Calculate the potential wave travel distance by adding the corresponding connection distance to the base node and the wave travel distance to the connected node.
    - ii. If the the potential wave travel distance is smaller than the stored distance for the base node, update the stored value.
3. Repeat steps 1 and 2 until no nodes are updated.

All node connections are physically valid, i.e. no connections pass through obstacles, so this process does not reduce shortest paths to values that are below the true optimum for each node. Instead the process takes another step towards this goal without adding unreasonable levels of computational complexity.

Figure 6.3 provides the wave travel distances throughout a domain featuring a charge at the centre point, before and after type-run optimisation. It is clear that this step is vital for removing many physical inconsistencies, yet, many regions with steep gradient changes are still present, e.g. at (5.5,0.5).

Figure 6.4 shows the reductions generated by using the smoothing function as a continuation of Figure 6.3. In localised areas, corrections of up to 0.7 m are applied, ensuring that consistency is maintained throughout the entire parameter space.

Using this additional robustness measure when calculating the shortest wave travel distance, the remainder of this chapter will test and develop the Direction-encoded Framework to determine how, and when, it could be used to replace current approaches for blast wave analysis.

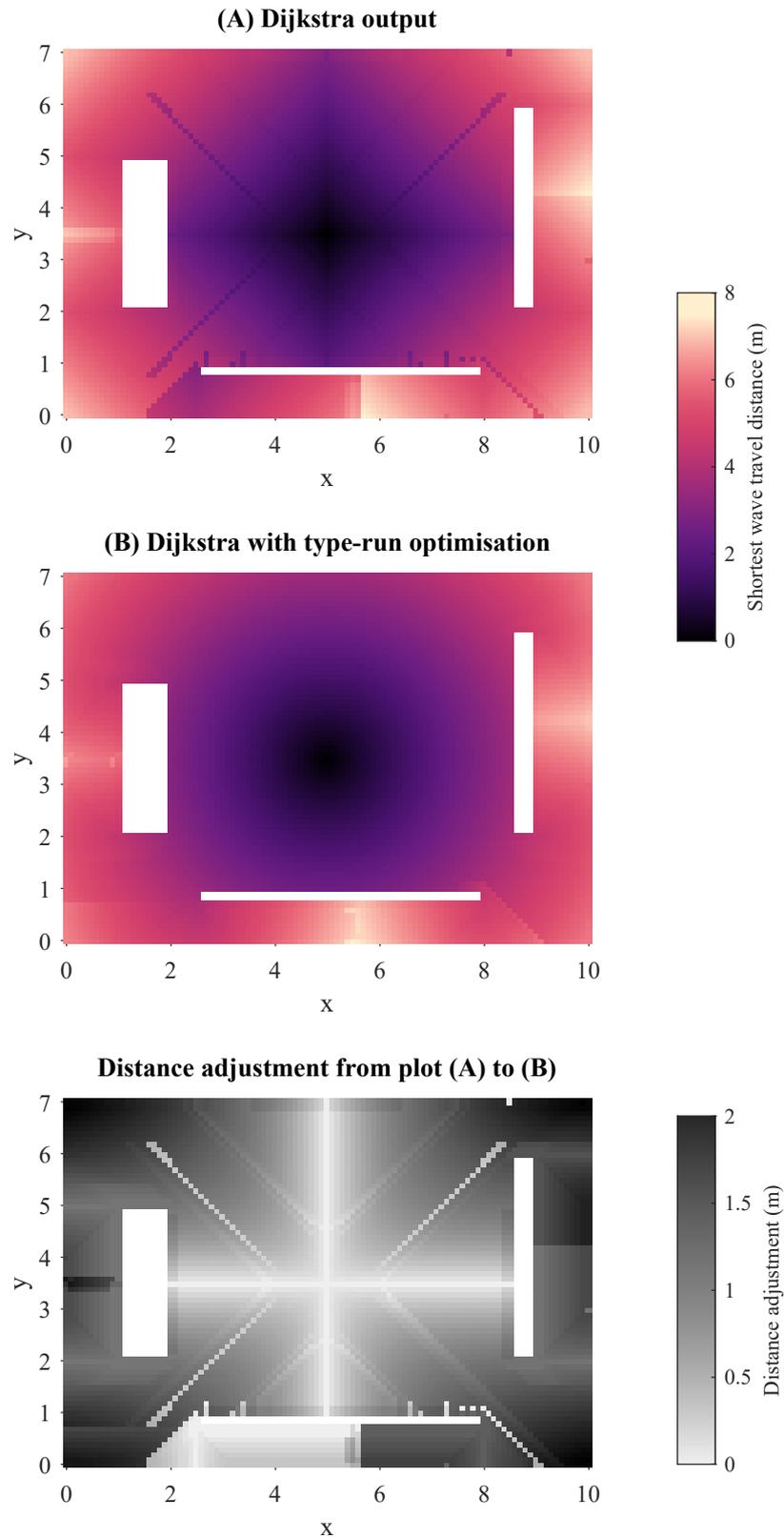


Figure 6.3: Distribution of shortest wave travel paths from the charge to each POI, showing how type-run optimisation removes most inconsistencies in the domain caused by Dijkstra's algorithm. Charge centre positioned at  $x = 5, y = 3.5$ , white voids are rigid obstacles.

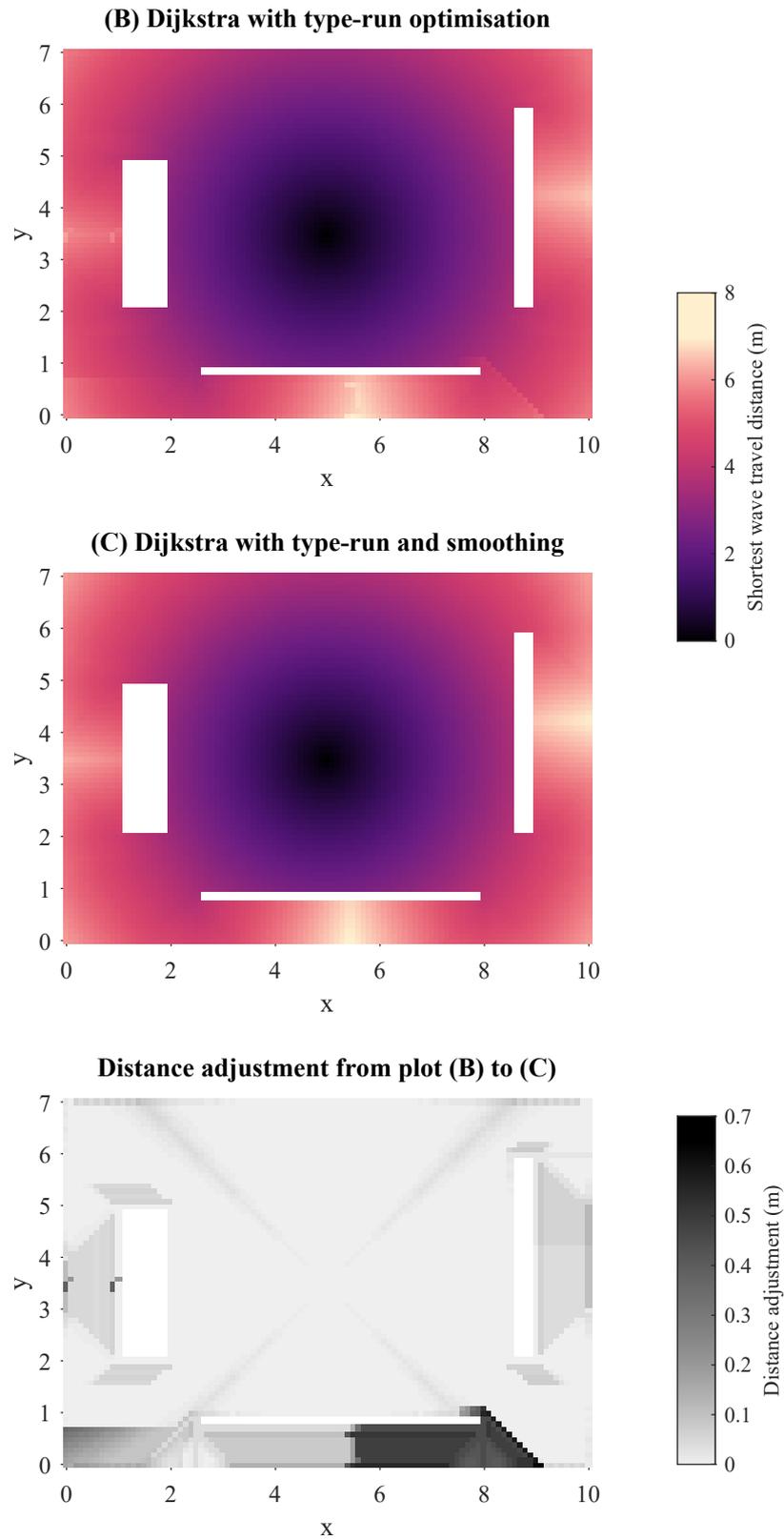


Figure 6.4: Additional path reduction caused by smoothing the output from type-run optimisation. Charge centre positioned at  $x = 5, y = 3.5$ , white voids are rigid obstacles.

## 6.3 Dataset development and training approach

### 6.3.1 Initial model architecture

As a result of the paper by Dennis et al. (2021) showing that good performance can be achieved for regressive blast analysis problems using a multilayer perceptron, a similar network architecture has been adopted for an initial testing phase of the Direction-encoded Framework. This aims to provide insight into how the input pattern can be redeveloped to benefit the generalisation capabilities, and predictive accuracy of the approach. For the remainder of this thesis, this tool is defined as the Direction-encoded Neural Network (DeNN).

Table 6.1 provides an overview of the network architecture that is coded in the Python programming language using the Tensor Flow and Keras packages for Machine Learning. Training is allowed to continue for up to 500 steps unless the early stopping criteria is met. This being that there is no improvement in the validation loss for 10 steps, where ‘no improvement’ includes loss variations of 1 kPa<sup>2</sup> or less. The network is only saved after each training step if it provides the best performing validation loss, replicating the process implemented by Bakalis et al. (2023). This ensures that if the training performance continues to improve despite a decline in validation performance, the weights and biases of the network are not saved. Thus, preventing overfitting and allowing for good generalisation with unseen inputs.

Table 6.1: Fixed network parameters.

Hidden network structure	500 500
Activation function	ReLU (Linear at output)
Loss function	Mean squared error
Optimiser	AdaGrad
Learning rate	0.01
Training steps	500 (with early stopping if validation loss does not improve for 10 steps)
Batch size	100
Dropout rate	0.1
Regularisation	L2
Weight initialiser	Glorot Normal
Bias initialiser	Zeros
Cross validation folds	4

Four fold cross validation is utilised to ensure that the network performance is evaluated for the entire training dataset (introduced in the next section). This process involves splitting the dataset into four equal sections, with four separate networks being trained on a different 75/25 training/validation subset. Performance is reported as the mean average of all the folds so that bias in the validation data split is removed whilst also ensuring that every point in the dataset is predicted by a network that was trained without considering that point, thus allowing for more robust performance assessment. Following cross validation, the final model is trained for the number of training steps equal to the average from all folds considering the early stopping criteria, using all the training data.

As with the study by Dennis et al. (2021), the AdaGrad (Adaptive Sub-gradient Descent) gradient descent algorithm is used for training since the dataset features localised effects and wide variations in outputs (Duchi et al. 2011). The variable learning rate is well suited to this as common features have smaller impacts on the updates whilst rare features have larger impacts (Hadgu et al. 2015).

It should be noted that this array of variables is only fixed for initial testing and development phases of the DeNN. Tuning of the key hyperparameters will be included once feature set has been developed throughout the following sections.

### 6.3.2 Training dataset

A key part of developing a Machine Learning tool that can be useful in practice is related to the quality and quantity of training data points. To develop a suitable dataset to analyse the performance of the DeNN, 25 randomly generated Viper numerical models, shown in Figure 6.5, have been simulated. This aims to provide a range of unbiased blast scenarios that include obstacles positioned in a wide range of locations so that the networks can generalise from the training process and provide predictions with suitable accuracy regardless of the prediction domain.

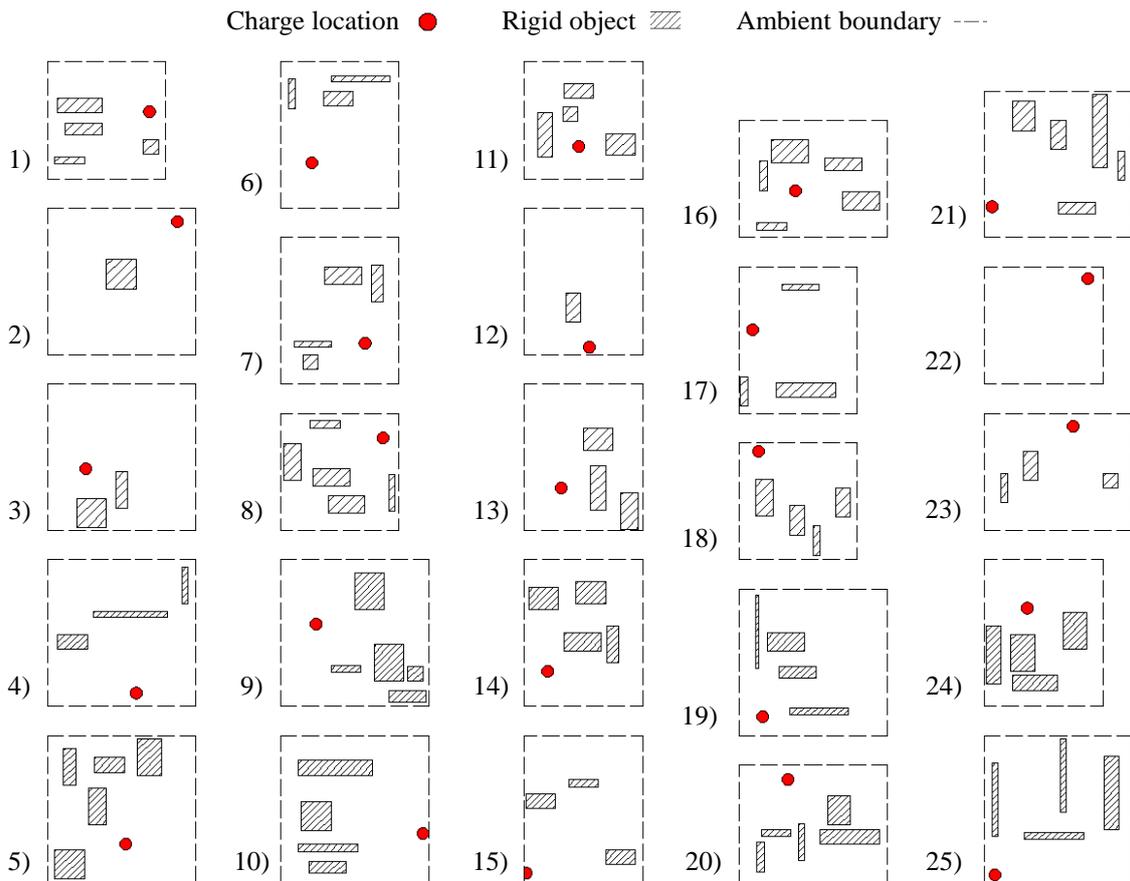


Figure 6.5: Randomly generated training models used to develop the DeNN.

The random generation process applied limits on the number of obstacles (0–5) and their size (0.01–5 m<sup>2</sup>), their shape (square/rectangle) and orientation (0° or 90° to the horizontal), and the size of the overall domain (10×10 m, 10×8 m, 8×10 m, 8×8 m). Each obstacle is also set to a height of 2 m.

The 1 kg TNT spherical charge was positioned at 1.5m above a rigid reflecting ground surface in every domain with a grid of gauges also being specified at this height with regular spacing of 0.1 m. Gauges within 1.5 m of the charge were then removed, and will not be predicted, as this 1.5 m sphere around the charge is a true free-air case. Standard rapid analysis methods, such as ConWep, can therefore be used if information about this region is required. Furthermore, peak overpressures are also expected to be far greater in this region than in any other part of the domain and so their removal reduces the range of values that need to be predicted by the ANNs. This will have the same impact as the specific impulse limit employed by Dennis et al. (2021), where performance was improved for predictions of lower magnitudes once the rare, especially large, values were omitted. Overall, this methodology provides 177277 unique datapoints for use in network development.

The size and shape of the domains used in the training process have little effect on the ultimate performance of the DeNN due to how the inputs are derived with reference to the surroundings instead of the domain itself. It is more important to provide a wide range of target peak overpressures and a range of activated (non-zero) directional input combinations. In this case, the 25 models provide the magnitude ranges shown in Table 6.2. These form the allowable bounds of inputs associated to future scenarios requiring prediction. It should be noted that the low mean value associated to the directional inputs is caused by the number of ambient (0 magnitude) inputs.

Table 6.2: Training dataset variable statistics.

Variable	Units	Min	Max	Mean	Std. Deviation
Directions	m/kg <sup>1/3</sup>	0	11.03	0.67	1.45
Wave travel distance	m/kg <sup>1/3</sup>	1.52	13.93	5.39	2.37
Peak Overpressure	kPa	4.72	672.34	51.63	51.20

At each gauge location, pressure histories are recorded by Viper. The peak reading is extracted and aligned with the directional input patterns and wave travel distances that are calculated using the discretised domain representation given by the 0.1 m gauge grid. Thus allowing the network to be trained considering inputs with a known target output from the validated solver. Details of the Viper models are given in section 6.3.5.

### 6.3.3 Testing models

In order to test the performance of the trained ANNs, two additional models have been simulated to enable a real world assessment of the predictive performance. These independent tests are not restricted by the aforementioned randomisation requirement and are developed with the aim of replicating some expected domain layouts that could be seen in practice. By assessing the performance for these unseen inputs, the impact that the randomised training dataset has on the generalisation capabilities can be observed.

Figure 6.6 provides the dimensions for both testing models, with all input parameters falling within the bounds of the training dataset shown previously in Table 6.2. T1 aims to test the network’s ability to predict the peak overpressure in a simple scenario with various blast panels, whereas T2 requires greater appreciation for more complex wave interaction effects, with channelling and shielding replicating a scenario more closely aligned to a cityscape layout, albeit a simple one. Both models include 1 kg spherical TNT charges positioned at 1.5 m above the rigid reflecting floor, and obstacles that are 2 m in height.

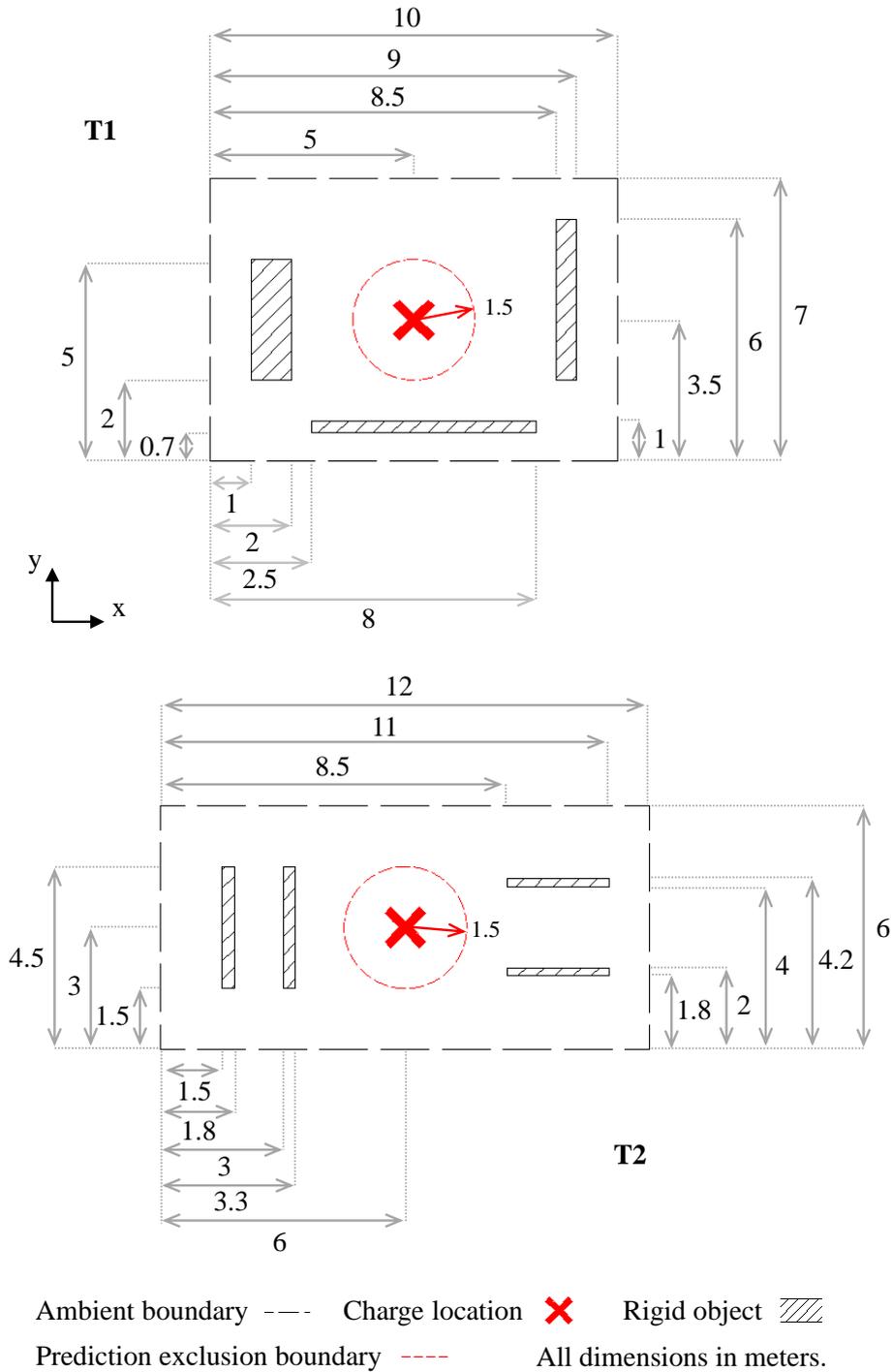


Figure 6.6: Selected testing models to be used for unseen ANN performance assessment.

A key feature of both testing domains is that they do not share equal size or shape with any of the models used to form the training dataset. Seeing as the DeNN references the surroundings of each prediction point and not the domain itself, predictions can still be generated on a 0.1 m gauge grid, positioned 1.5 m above the rigid ground plane. It should be noted that it is possible for the user to change this predictive grid spacing in future use cases if required.

A large range of input patterns and corresponding outputs are generated for training, and only specific patterns will also feature in real world applications. In some cases the accuracy may be better than expected, and in some it may be worse. For example, the average error of the network may be 10%, however this could include a 1% average error for points being shielded, and a 30% error for those where channelling will have the largest impact on the peak parameter. By generating predictions for these specifically designed testing models, it will highlight these variations.

### 6.3.4 Analysis method

The schematic shown in Figure 6.7 presents the data splitting process used throughout training, with 4-fold cross validation and the two independent testing models. As noted in Section 6.3.1, the final model used to predict the testing data is trained using all training data for the average number of steps required by the cross validation process.

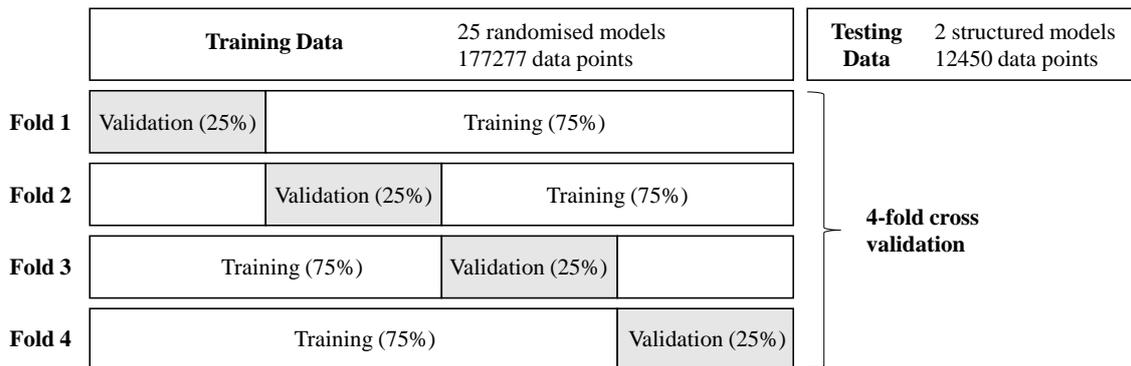


Figure 6.7: Representation of how data is split for K-fold cross validation during training, showing that this is independent of the two additional models that have been devised for testing the trained DeNN. Adapted from Pannell et al. (2022).

### 6.3.5 Viper::Blast modelling

Each model in the training dataset will be simulated using the chosen numerical solver, Viper::Blast. Following its validation in Section 3.2, this solver provides the functionality required to generate peak overpressure readings throughout each domain at the specified grid spacing.

Setup parameters are provided by Table 6.3 and in each case all boundaries are set to be ambient (transmit) aside from the rigid reflecting floor. The simulations utilise 1D–3D mapping with the 1D stage running until the blast wave has reached one cell away from this boundary at 1.499 m from the detonation point. Whilst the DeNN will predict for a 2D plane at 1.5 m elevation, it will be trained with data from a more comprehensive modelling

process, carried out in 3D with a domain height of 2 m. This matches the obstacle height to replicate scenarios where the building height is so large that the clearing effects caused by the wave reaching the top edge do not influence the blast loads at the ground level, such as along city streets. To remove the potential influence of ambient boundary effects on the peak overpressures recorded by gauges at the boundaries, each boundary in the  $x$  and  $y$  plane is extended by 0.5 m.

Table 6.3: Viper::Blast training model parameters.

Solving method	Ideal gas
Charge size (kg)	1
Charge composition	TNT
Charge density (kg/m <sup>3</sup> )	1600
Charge energy (J/kg)	$4.52 \times 10^6$
Mapping	1D-3D
1D cell size (m)	0.001
1D CFL	0.5
3D cell size (m)	0.02
3D CFL	0.4
Ambient temperature (K)	288
Ambient pressure (Pa)	101325
Termination time (s)	0.05

### 6.3.6 Performance metrics

Comparisons between each network variation will be made using three metrics. The first is the Young's Correlation Coefficient, calculated using Equation 6.1.

$$R_t^2(o, m) = 1 - \frac{\sum_{n=1}^N (m_n - o_n)^2}{\sum_{n=1}^N o_n^2} \quad (6.1)$$

Where  $R_t^2$  is the correlation coefficient,  $m_n$  is the predicted peak overpressure,  $o_n$  is the target peak overpressure and  $N$  is the total number of data points. An  $R_t^2$  of 1 shows that every prediction equalled every target, values close to 1 show high correlation between the two variables and close to 0 shows little correlation.

The Mean Absolute Error is also used, calculated using Equation 6.2, to assess the average magnitude of error in the predictions using the associated units of kilo pascals.

$$MAE = \frac{1}{N} \sum_{n=1}^N |m_n - o_n| \quad (6.2)$$

Finally, the average percentage error is calculated using Equation 6.3. This metric removes the influence of magnitude from the error assessments.

$$Avr\%E = \frac{1}{N} \sum_{n=1}^N \frac{|m_n - o_n|}{o_n} \times 100 \quad (6.3)$$

## 6.4 Initial performance assessment

To assess the performance of the DeNN using the input pattern design shown in Figure 6.1, the 4-fold cross validation technique is used. Mean performance statistics are reported in Table 6.4 for the train and validation stages.

Table 6.4: Mean performance metrics from 4-fold cross validation of the DeNN.

Development stage	$R_t^2$	MAE (kPa)	Avr%E
Training	0.9654	5.84	17.9
Validation	0.9647	5.87	18.0

Correlation coefficients of around 0.96 show good promise for the approach, however, Remennikov & Rose (2007) achieved a coefficient of 0.997 for peak pressure predictions behind blast barriers using their bespoke ANN. It is therefore clear that further developments are needed, especially considering the average errors around 18% are also outside of what is typically defined as ‘good enough’ for blast applications, this being less than 10% (Rigby et al. 2014). Despite this, there is minimal overfitting as the validation performance is only slightly worse than training in each metric. The dataset is shown to be generalised consistently with different groups of data being held out in four separate training processes. As a result, the network used for testing is trained using all of the data with performance statistics from predicting the outputs for both tests shown in Table 6.5. For this stage, the number of training steps used equalled the average from each fold during cross validation.

Table 6.5: Performance metrics for both testing domains, simulated using the DeNN, compared to a Viper model.

Testing model	$R_t^2$	MAE (kPa)	Avr%E
1	0.9202	16.45	51.0
2	0.9594	10.65	44.2

There is a stark decrease in performance from validation to testing, highlighting that the testing models include points that are similar to those in the randomised training dataset that are not predicted with good accuracy. Figures 6.8 and 6.9 provide visual representations of each prediction that forms each domain in addition to Viper outputs and the absolute error between both modelling approaches. This shows that wave reflections are largely ignored by the DeNN, in particular around the areas of pressure build up in front of the rigid obstacles, the network is not able to amplify its predictions. The use of only 8 directional lasers may also mean that some input patterns omit the presence of obstacles in the domain. It is likely that the wave travel distance has dominated the magnitudes of each prediction, with some angularity being present to also ignore any domain symmetry (as highlighted by the absolute error plots). Performance is therefore not yet at an acceptable level, despite another testing model potentially matching, or exceeding, the validation performance, depending on the distribution of points that are included.

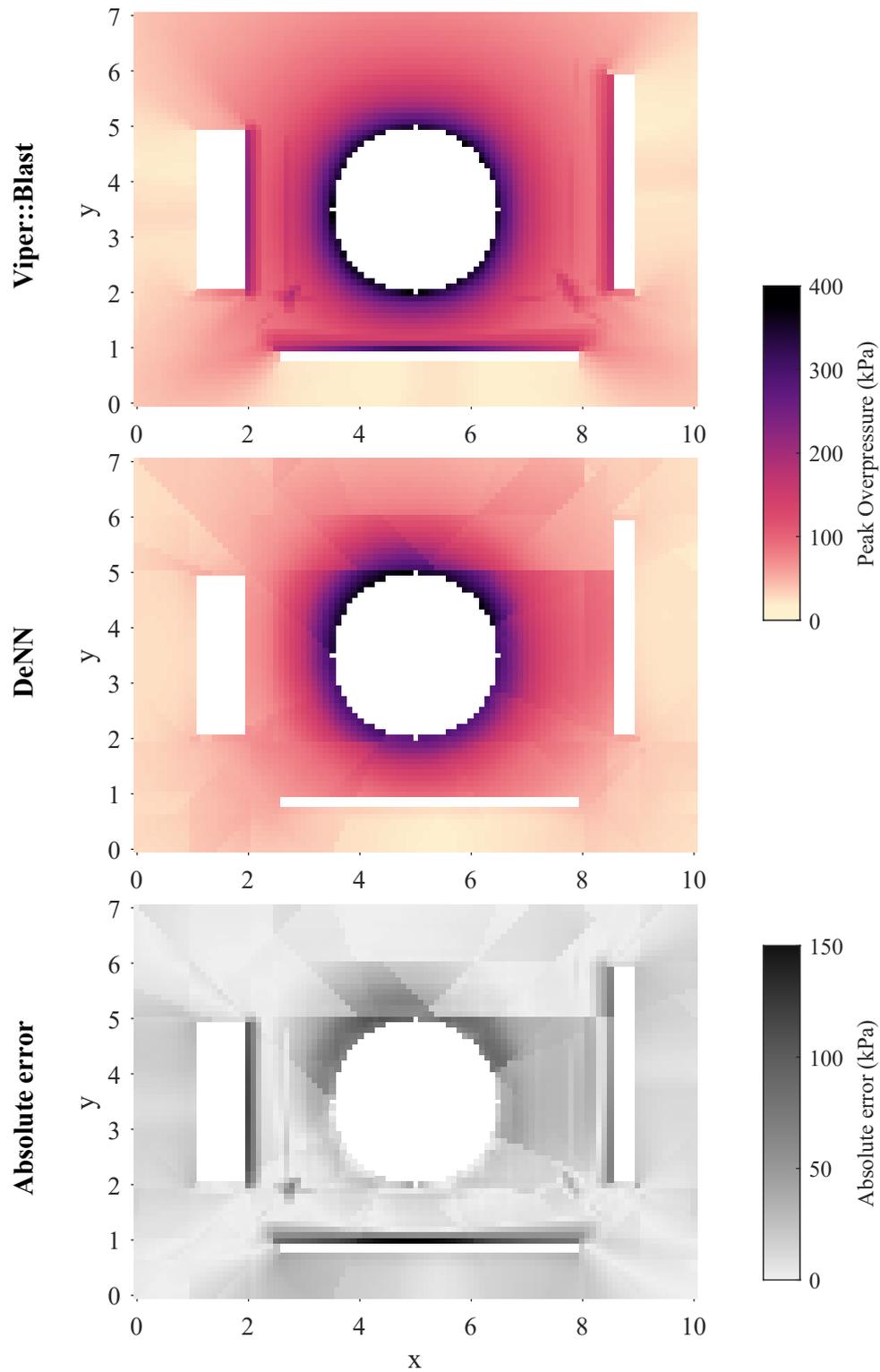


Figure 6.8: T1 peak overpressure targets generated by Viper::Blast, predictions from the DeNN, and the resulting absolute errors. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone.

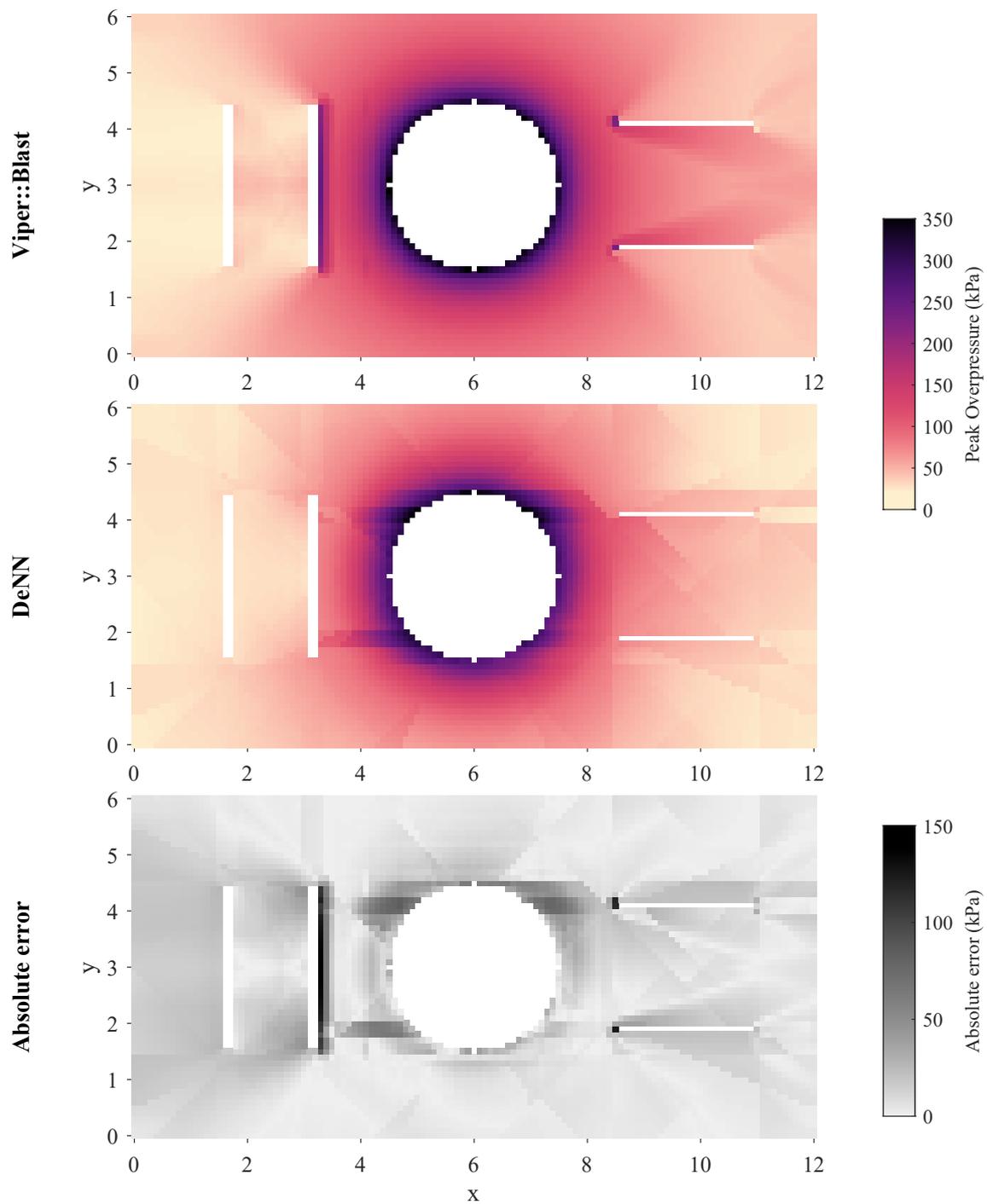


Figure 6.9: T2 peak overpressure targets generated by Viper::Blast, predictions from the DeNN, and the resulting absolute errors. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone.

The following list outlines the five main issues identified in this assessment:

1. Angularity, predictions formed with clear reliance on the angular spacing of the directional lasers.
2. Wave reflections near surfaces are not considered.
3. Domain symmetry is not respected.
4. Large variations in values are not consistently predicted.
5. Obstacles are missed by the directional lasers that would influence the predictions.

In the following section each area for improvement is addressed by independent, creative input parameter adjustments before a final feature set for the DeNN is evaluated to determine the capabilities of this approach.

## 6.5 Feature engineering and development

### 6.5.1 Introduction

Throughout this section various adaptations are made to the input pattern of the DeNN. This feature engineering process is tested incrementally to highlight the benefit of each development. Fair comparisons are ensured through use of a set randomisation seed in Python, TensorFlow and Keras. This ensures that each fold used in cross validation remained the same for every network. Similarly, consistency was maintained when initialising and updating the weights and biases, with each training process selecting the same points in each batch of each step. This removes random variation, so any performance improvements are directly linked to the creative feature selection decisions that are explained throughout the remainder of this section.

### 6.5.2 Rotating laser directions and a mirrored dataset

Including the closest charge direction input provides a regression network with an input that is not continuous. Whilst this is valid and predictions may be suitable with this approach, it is likely that removing this variable by incorporating it into the network itself would allow for better generalisation. For example, in a scenario where a switch node is used to take the value of 0 when a criteria is met and 1 when it is not, use of two separate networks could lead to greater performance because each network can be tailored to each distinct case.

For the DeNN, a split for each charge direction is not feasible considering the size of the dataset that would be required to sufficiently train each network based on all 8 possible closest directions. Instead, the proposed development allows the directional lasers to rotate such that direction 1 always faces the charge centre. The value of the input associated to this direction is restricted to the wave travel distance to ensure that obstacles behind the charge are not translated to the DeNN, thus having no impact on the predictions.

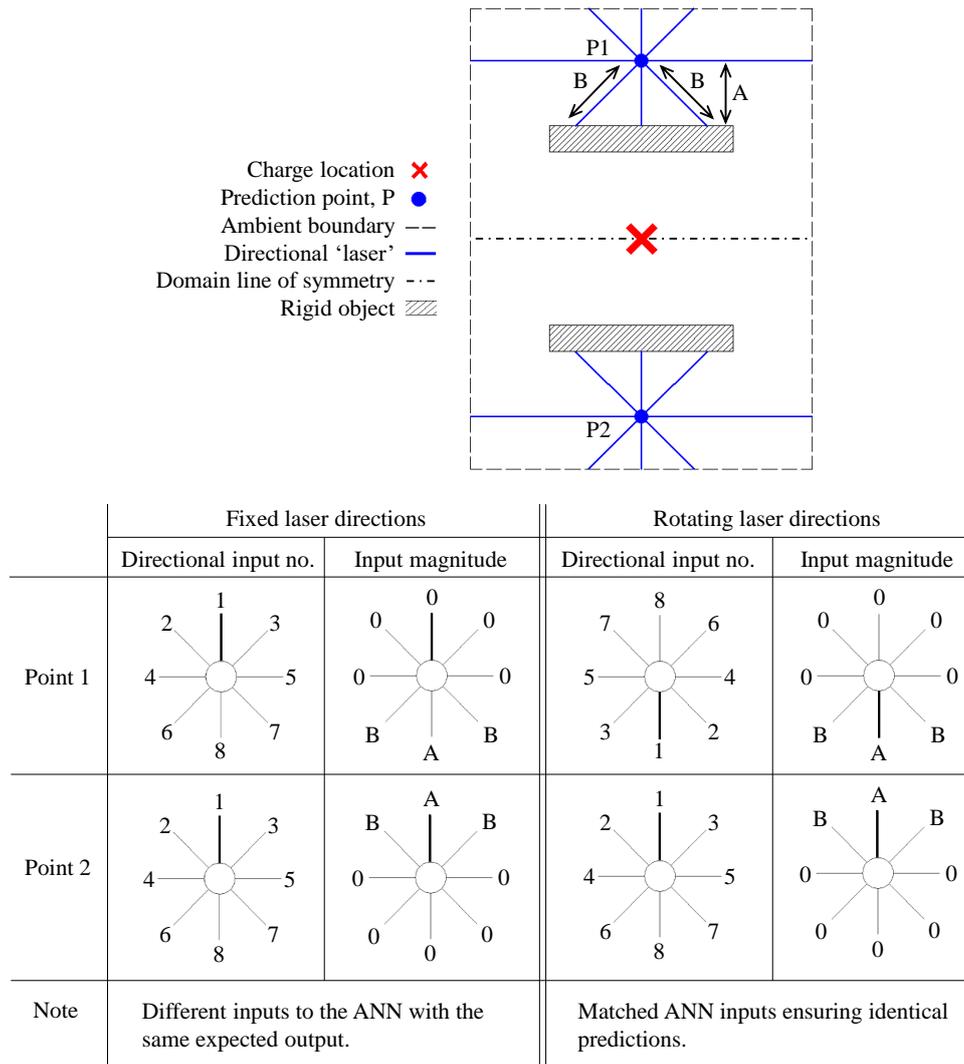


Figure 6.10: Example of how the network could be provided with differing inputs for the same expected output prediction. Use of rotating laser directions ensures identical input patterns to maintain consistency in the predictions.

Including rotating directional lasers and removing the charge direction input is also necessary to help with generating consistent predictions for points that are expected to have identical outputs. Through analysing the absolute error from the initial DeNN tests, shown in Figure 6.9, symmetrical consistency is rarely preserved. This is because in the fixed laser approach, the input patterns used for two points, P1 and P2, shown in Figure 6.10, are different despite the outputs needing to be identical due to symmetry. P1 activates (non-zero magnitude) lasers 6, 7 & 8, whereas P2 activates 1, 2 & 3. Rotating lasers corrects for this error by ensuring that direction 1 points towards the charge and the same input nodes are activated in both instances. Providing the same input pattern to the network ensures that the same output will be generated.

However, maintaining consistent input patterns is not possible throughout the entire domain as the angle of rotation can change which side of the directional laser rosette will be activated. Considering this in a simple case that builds upon the previous example, Figure 6.11 shows the differing input patterns are produced for two points that should share

equal predictions. The directional input rosettes include a distance ‘B’, at input number 2 for P1, but at input number 3 for P2. The tuning of weights and biases associates to each direction will therefore be inconsistent, resulting in bias towards one side of the rosette that is likely to cause symmetrical inconsistencies.

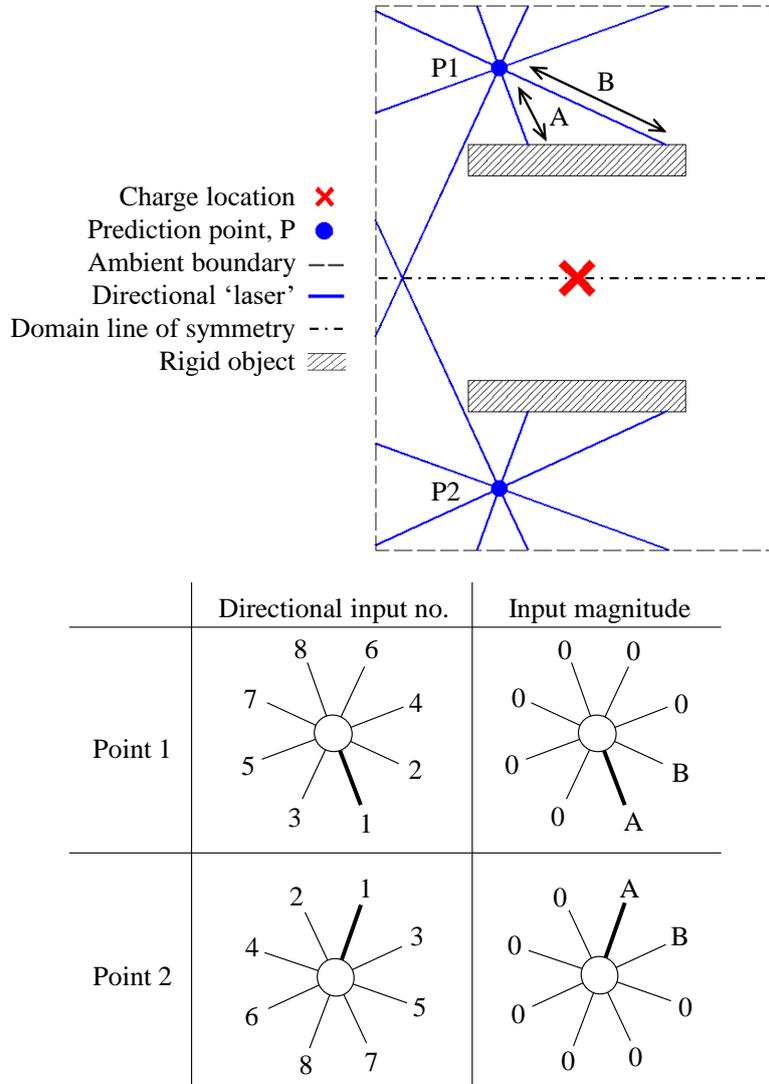


Figure 6.11: Proof of how mirroring the dataset can expand the number of points in training without duplicating input patterns.

To reduce this issue, the dataset can be mirrored by swapping directional inputs 2/4/6 for 3/5/7 whilst retaining the other inputs and the target output, to double the number of valid data points can be made available for training without breaking any physical constraints of the problem. Using this approach, the training dataset increases to 354554 unique data points, allowing the networks being trained to have access to input patterns that relate to obstacles on both sides of direction 1 in equal quantities.

Table 6.6 notes the training and validation performance metrics from the initial feature set compared to when rotational lasers and the mirrored dataset are used. The MAE and average percentage errors decrease slightly, but the correlation coefficient is reduced. Performance is also shown for T2, where improvements are made for all performance

metrics, and symmetrical consistency is achieved with a prediction comparison shown in Figure 6.12. Despite this improvement, average errors remain above 10% with wave reflections, clearing and channelling not being correctly considered around the rigid obstacles.

Table 6.6: DeNN performance comparison between the initial feature set, presented in Section 6.4, to when rotating laser directions and a mirrored dataset is incorporated into input generation.

Improvement method (s)	Development stage	$R_t^2$	MAE (kPa)	Avr%E
Initial - N/A	Training	0.9654	5.84	17.9
	Validation	0.9647	5.87	18.0
	T2	0.9594	10.65	44.2
Rotating laser directions and mirrored dataset	Training	0.9626	5.64	15.9
	Validation	0.9625	5.65	16.0
	T2	0.9696	9.33	32.1

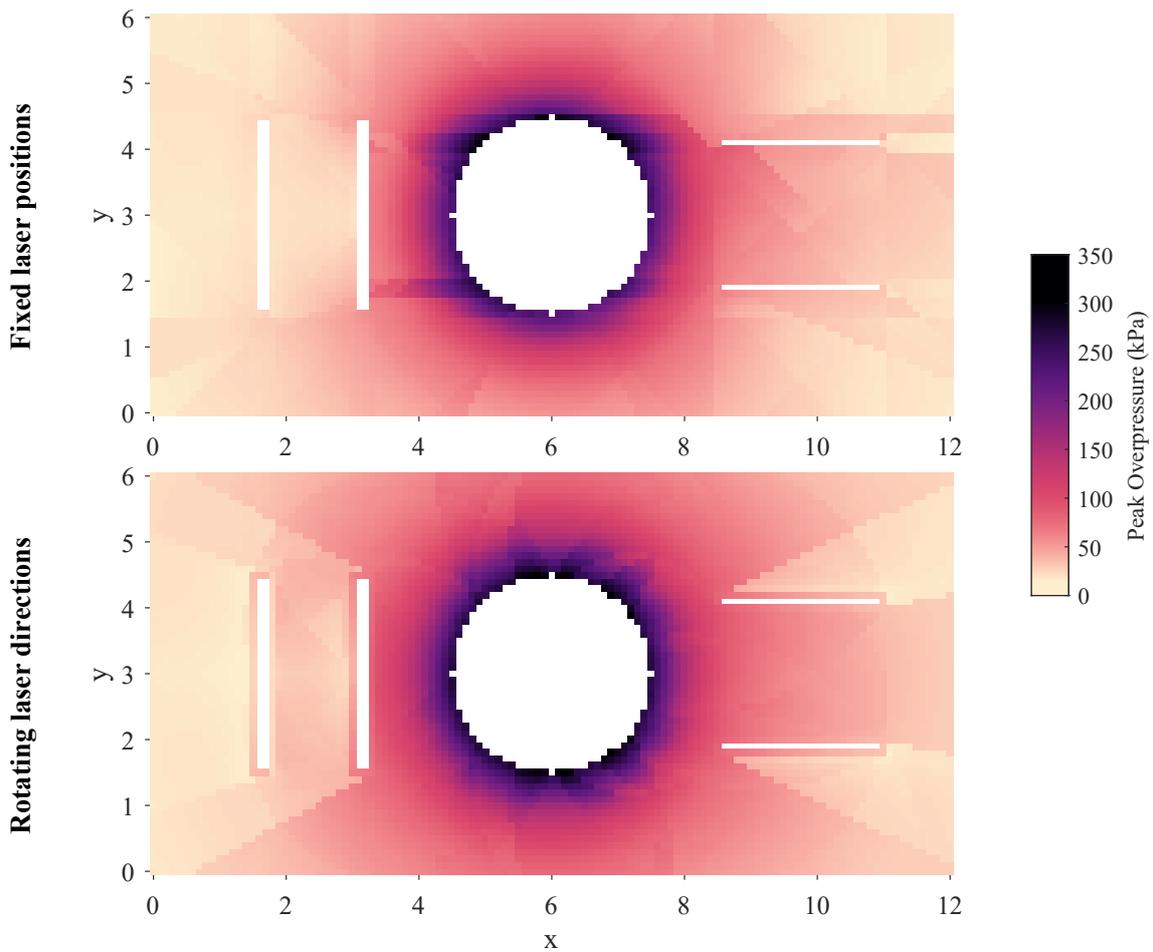


Figure 6.12: DeNN predictions showing when the directional inputs are fixed in position relative to the domain compared to when a mirrored dataset is used in addition to allowing the directional inputs to rotate such that direction 1 points towards the charge centre.

### 6.5.3 Wave reflections with superposition

The next method for improving generalisation potential is related to how wave reflections, shielding and channelling effects are considered. At present, the directional lasers will provide the network with values associated to any obstructions in specified directions from the POI. For directions 1, 2 and 3, a small input therefore suggests that there is an obstacle between the charge and the POI that will lead to a reduced peak reading due to shielding. Similarly, a low input for directions 6, 7 and 8 suggests that there is an obstacle immediately behind the POI, leading to amplified blast parameters due to the wave reflection.

In both cases, a low input relates to a large contribution of contrasted blast wave mechanics that will heavily alter the predicted values. Setting an ambient directional input as 0 places an intended ‘no effect’, in between the largest effect in terms of amplification and reduction. As a result, it is likely that the network will not truly differentiate between points that should be predicted with larger/smaller peak pressures based on what the blast wave is experiencing, instead focusing on the wave travel distance as the main reason to alter its output.

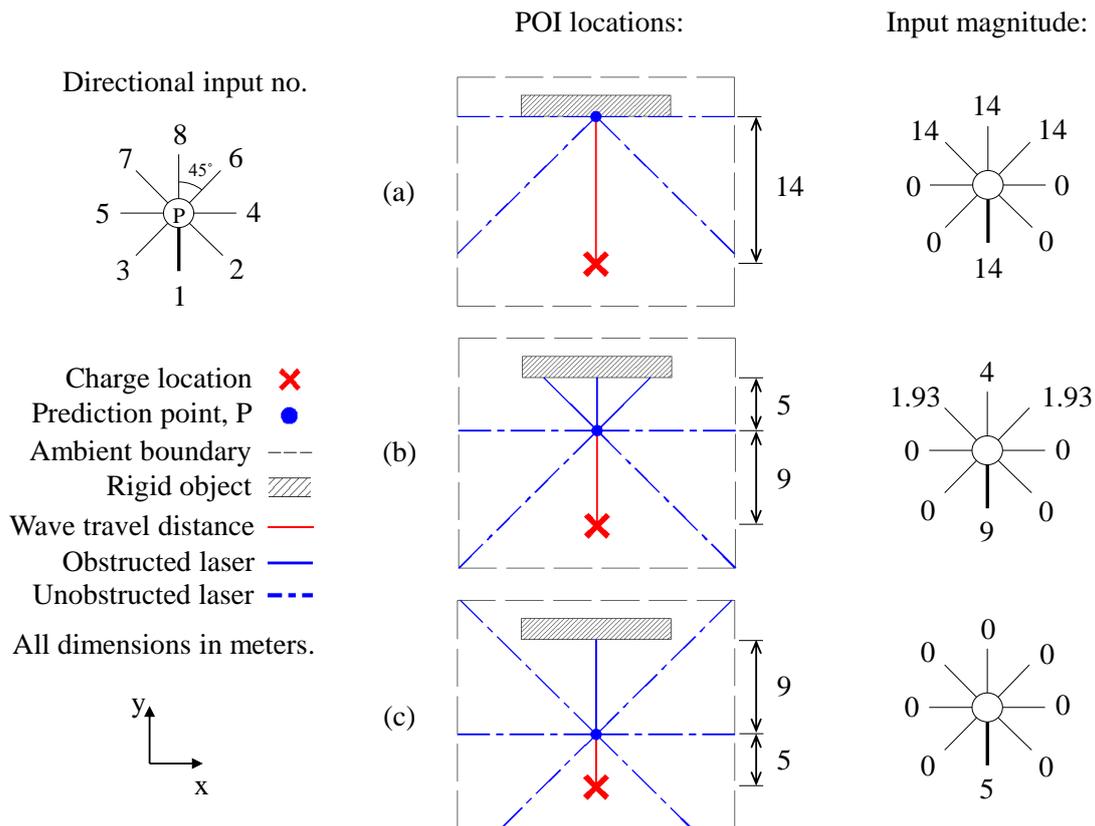


Figure 6.13: Example of how wave reflections are represented by the input pattern. Larger values for directional inputs 4, 5, 6, 7 and 8 imply a larger reflection effects.

Figure 6.13 presents the proposed approach for improved wave reflection consideration with plot a, showing that when a point is on an obstacle surface its directional input is taken to be the wave travel distance. Then, as the point moves away from the obstacle (b), the larger laser distance is subtracted from the wave travel distance to provide an

input closer to 0. Finally, plot c shows that at a sufficient stand-off, the obstruction is deemed to be sufficiently far away from the point that it is essentially having no effect on the prediction, i.e. acting like an ambient boundary with a 0 input.

Each directional input is calculated using this approach, shown in Equation 6.4, if a obstruction is identified. Otherwise the value remains as 0 for an ambient interaction.

$$\text{Directional input} = \max(\text{Wave distance} - \text{Obstruction distance}, 0) \quad (6.4)$$

This superposition equation is inspired by how existing fast running methods utilise multiple charge superposition to model blast wave reflections, with a charge at an imaginary source behind a wall providing the wave that amplifies the predictions at rigid surfaces (Pope 2011). In this application, this has the effect of defining a ‘zone of influence’ around the POI such that any obstacle interactions outside of the zone are deemed to be insignificant for the peak pressure prediction i.e. acting like an ambient boundary with a 0 input. A similar concept is shown to apply to the design of continuous beams (Gallet et al. 2023). Figure 6.14 provides the zones of influence for two example points in a domain. For point A, the close proximity to the charge results in a small zone of influence, showing how the blast wave will not be obstructed by the presence of obstacles outside of this region as it approaches the POI. The zone for point B is much larger since shortest wave travel path must wrap around the horizontal obstacle. The directional lasers must therefore consider obstacles that could contribute to wave coalescence along this path.

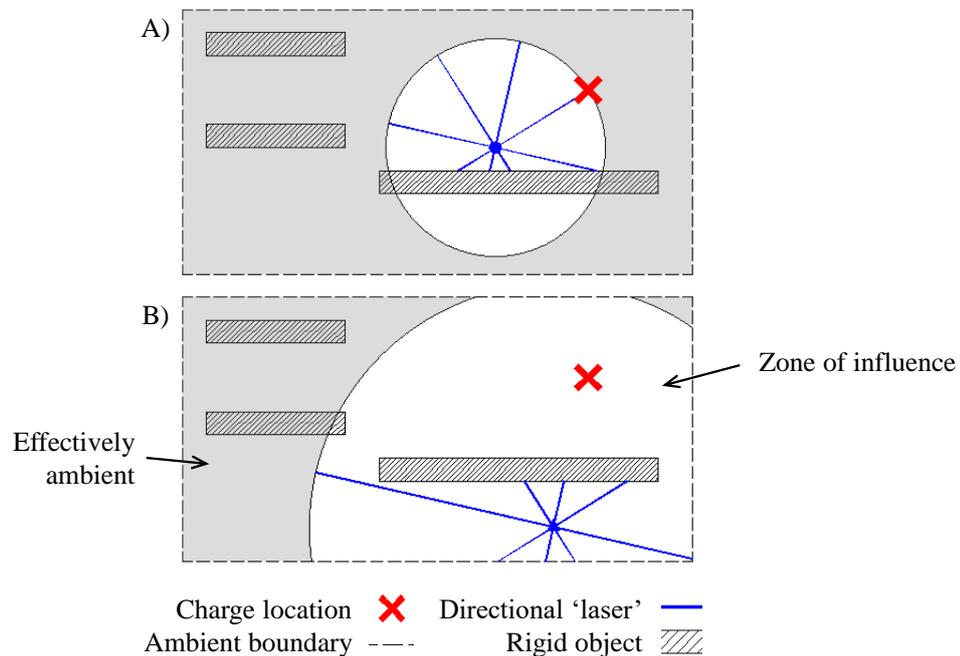


Figure 6.14: Zone of influence examples showing the regions where directional inputs would be treated as ambient at stand-off distances greater than the wave travel distance in accordance to Equation 6.4.

In theory, the polarity of inputs to an ANN does not affect its predictive performance as long as the network parameters are optimised through sufficient training steps and remain consistent throughout development and use. The weights and biases of the first layer of connections can be adjusted to account for positive or negative input values. However, the magnitudes of each input and their relationships do affect performance. Equation 6.4 enables larger or smaller interaction effects from obstacles that are closer or farther from the POI to be translated with correspondingly larger or smaller values relative to the shortest wave travel distance. However, the network is still responsible for learning that the forward directions (1, 2, 3) will relate to shielding and channelling, whereas the backwards directions (6, 7, 8) are associated to reflections, both having different impacts on amplification or reduction.

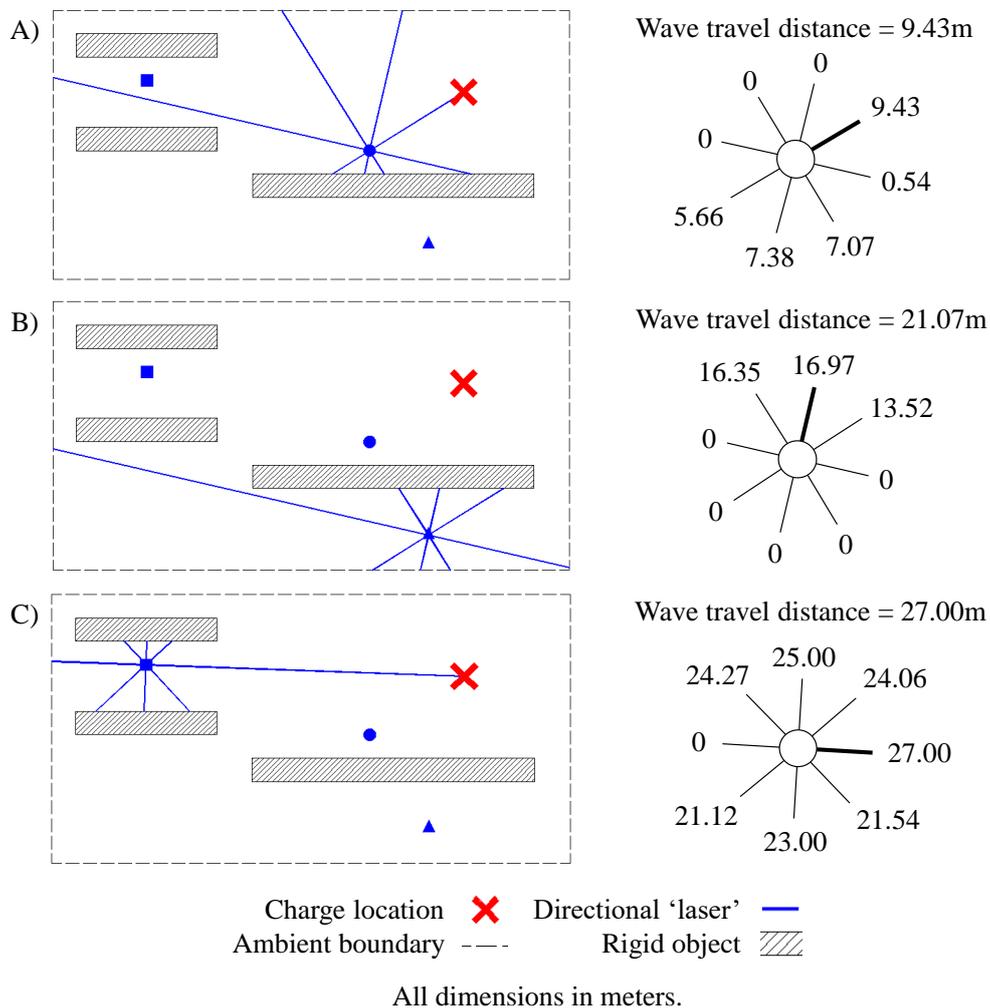


Figure 6.15: Additional example directional inputs for various points. Thick black line in the directional rosette indicates direction 1.

Figure 6.15 provides three examples of how some common input patterns are represented using this approach. Point A includes a wall behind the POI and so the inputs in the backwards direction are closer to the wave travel distance. The contrast with lower values in the forward positions helps the network to understand that blast wave reflection should be considered when forming the prediction. Next, point B is being shielded by a wall resulting

in larger forward inputs and low backwards ones. Finally, point C is between two rigid bodies, causing large values to feature in all side directions of the input pattern. In every case, a distinct combination of directional inputs are activated, thus providing different routes for information to be passed through the DeNN's connections when predictions are formed, ultimately allowing for differing wave processes to be represented.

Following training using Equation 6.4, Table 6.7 shows that performance significantly improves in all metrics with correlation coefficients surpassing 0.985 and average errors reducing below 15%. This network design will therefore be used for future comparisons as further improvements are trialled.

Table 6.7: DeNN performance comparison between the current best performing feature set, presented in Section 6.5.2, with when the superposition equation is also incorporated in input generation.

Improvement method (s)	Development stage	$R_t^2$	MAE (kPa)	Avr%E
Rotating laser directions and mirrored dataset	Training	0.9626	5.64	15.9
	Validation	0.9625	5.65	16.0
Rotating laser directions, mirrored dataset <b>and superposition equation</b>	Training	0.9894	4.00	13.3
	Validation	0.9892	4.02	13.3

#### 6.5.4 Multiple neural networks

When considering the peak pressure distribution through a domain featuring various obstacles, larger magnitudes of peak overpressure are often in positions where no shielding from obstacles is provided. These positions are dominated by free air blast waves that can be amplified by channelling or reflections from obstacles behind the POI, whereas POIs behind obstacles experience clearing and diffraction effects leading to reduced readings. Requiring a single neural network to learn about the processes associated with both of these regions may therefore needlessly restrict prediction accuracy. Since a distinction can be made by considering each POIs position relative to the charge, the DeNN model is applied as two separate ANNs with identical input-output structures. One network (ANN-1) is used for POIs with a direct line of sight to the charge, and the other (ANN-2) is used when an obstruction is present. A flow chart of the splitting process for each POI is given by Figure 6.16, with Figure 6.17 providing the resulting distribution for T1.

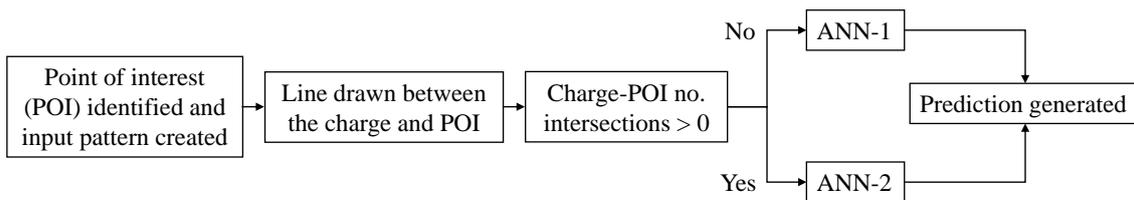


Figure 6.16: Flowchart showing how all data points are distributed into each ANN.

While this approach may not fit the definition of ensemble learning, whereby the outputs from multiple models trained on the same task are combined to improve overall performance, the concept aligns with the broader idea of reducing the complexity of the problem space to enhance the predictive accuracy. It also creates a framework that is similar to RAINet, presented by Zaleski & Prozument (2018), where inputs are sorted into one of five ANNs that provide predictions for varying groups of the problem.

Separating each set of points allows ANN-1 to form strong connections with the wave travel distance as this is the most critical variable being provided in determining the peak overpressure. Conversely, ANN-2 is able to form more balanced connections, as the proximity of surrounding obstacles will be more influential with the wave effects that must be considered. Although not done in this thesis, separating the simple from the more complex settings in this way facilitates the implementation aspects of transfer learning. For example, ANN-1 could be replaced with simple empirical predictions, or different models to account for TNT equivalence (Grisaro et al. 2021, Pannell et al. 2023).

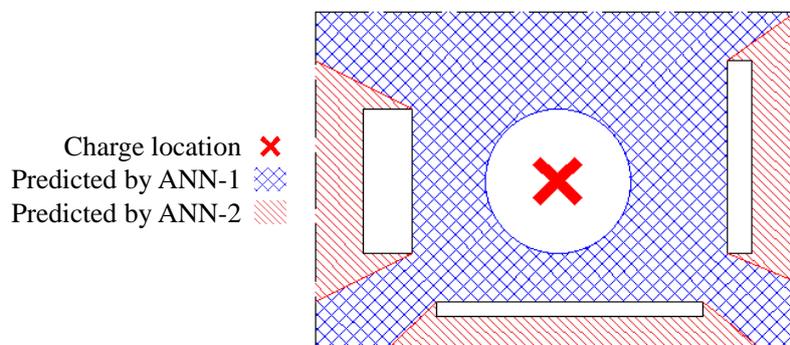


Figure 6.17: Distribution of points predicted by each ANN for T1. Includes 1.5 m un-hatched region around the charge that is not predicted.

Table 6.8: DeNN performance comparison between the current best performing feature set, presented in Section 6.5.3, with when the multiple neural networks are used.

Improvement method (s)	Stage	ANN num.	$R_t^2$	MAE (kPa)	Avr%E
Rotating laser directions, mirrored dataset and superposition equation	Training	—	0.9887	4.03	13.2
	Validation	—	0.9884	4.06	13.3
Rotating laser directions, mirrored dataset, superposition equation <b>and multiple ANNs</b>	Training	1	0.9922	3.77	7.9
		2	0.9412	3.83	22.0
		Avr.	0.9763	3.79	12.3
	Validation	1	0.9920	3.79	8.0
		2	0.9404	3.84	22.1
		Avr.	0.9759	3.81	12.4

Table 6.8 shows the performance of each network when the multiple ANN approach is used, including an average based on the number of points predicted by each ANN. This allows for a comparison to when a single ANN is used to show that a reduction in the correlation coefficients in training and validation is clearly impacted by the inability of the ANN-2 to predict the overpressure to the same standard as ANN-1. However, the overall MAE and percentage error metrics improve suggesting that the correlation coefficient reduction is likely to be caused by a greater number of points being over or under predicted as opposed to there being an even split of over and under estimation that would improve the  $R_t^2$  score. The performance when using multiple ANNs is therefore deemed to be better than the previous approach, meaning all iterations of the DeNN in the remainder of this thesis includes two independent networks.

### 6.5.5 Expanded input patterns

Aside from being useful to limit the number of calculations required to produce the input patterns, there was no physical reasoning for specifying 8 directions in the original DeNN input design. Figure 6.18 shows how this decision could lead to some obstacles being missed in the inputs of POI that will be influenced by their presence. It is shown in scenario a that the inclusion of 16 lasers can be useful in detecting obstacles that would otherwise be missed when only 8 are used (thin obstacle to the left of the POI). Similarly, scenario b may benefit from additional laser interactions with each obstacle in the domain as more directional inputs are provided.

Through providing the network with more data that is relevant to the problem it is trying to model, the predictive performance is expected to improve. Twelve directional inputs are also trialled to explore the balance between overloading the DeNN with information and providing enough to enable good generalisation of the problem.

Table 6.9: Performance comparison between when 8, 12 and 16 directional inputs are provided in the input pattern of the DeNN. Rotating laser directions, the superposition equation, mirrored dataset and multiple ANNs are included in input generation.

Directional laser count	Stage	ANN num.	$R_t^2$	MAE (kPa)	Avr%E
8	Training	1	0.9922	3.77	7.9
		2	0.9412	3.83	22.0
	Validation	1	0.9920	3.79	8.0
		2	0.9404	3.84	22.1
12	Training	1	0.9944	3.32	6.8
		2	0.9553	3.30	18.8
	Validation	1	0.9941	3.35	6.8
		2	0.9539	3.33	19.0
16	Training	1	0.9950	3.26	6.7
		2	0.9620	3.04	17.3
	Validation	1	0.9946	3.30	6.7
		2	0.9600	3.09	17.6

Table 6.9 shows how all performance metrics improve as the number of lasers increases. This culminates in 16 lasers providing correlations of 0.96 and above, with a MAE around 3.15 kPa. As the increase from 8 to 12 lasers is larger than from 12 to 16, it is likely that the network performance will converge after a set number of lasers are added. Ideally, an infinite number of lasers would be used so that the entire surroundings are translated to the ML tool. However, adding lasers linearly increases the computation time for a tool that is intended to function rapidly. Consequently, 16 lasers are used for the remainder of this document.

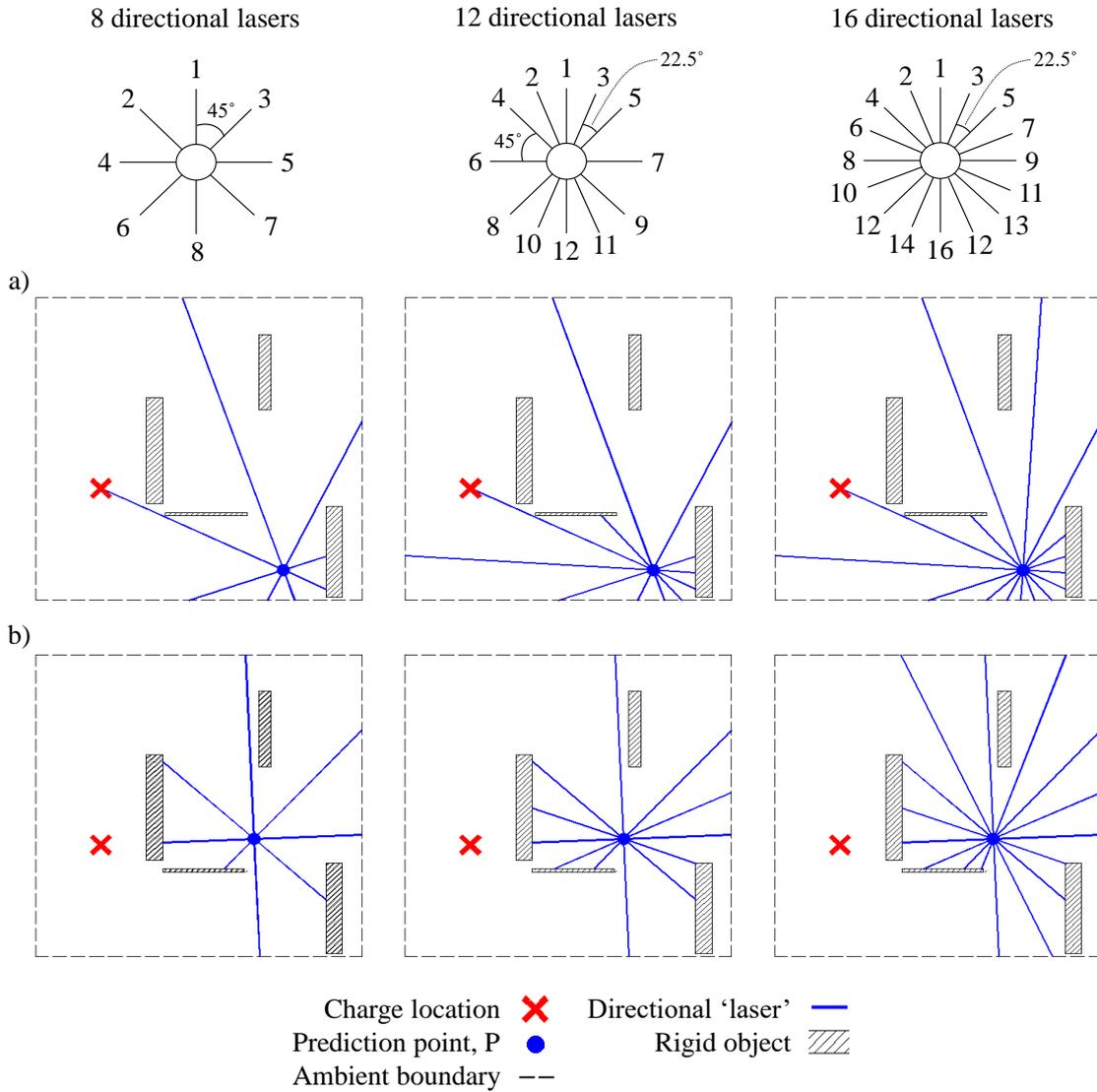


Figure 6.18: Expanded directional input options.

### 6.5.6 Kingery and Bulmash pressure input

In the literature review of ML applications in blast (Section 2.6), it was discussed that Pannell et al. (2022) improved the predictive performance of an MLP by using a physically informed loss function. The authors utilised the knowledge that blast waves decay with increasing stand-off, allowing them to apply a penalty to the network when predictions were made that disobeyed this law. Applying the same approach is not possible with the DeNN due to how it is used to predict values in complex domains, featuring many reflections and confinement effects. However, taking inspiration from Pannell et al. (2023), transfer learning using the free air incident overpressure, calculated using the Kingery and Bulmash method at a stand-off distance equal to the associated shortest wave travel distance, could help to improve performance of the DeNN. In this case the overpressure value replaces the wave travel distance in the input pattern used to provide the network with a base value that can be scaled appropriately depending on the directional inputs.

Figure 6.19 shows the KB predictions for T2, alongside Viper targets and the absolute error. This indicates where the DeNN will need to apply amplification and reductions in the peak overpressure to appropriately quantify the impact of the various obstacles in the domain. In essence, the ANNs are required to predict the absolute error plot to transform the KB predictions into the Viper outputs.

Table 6.10 shows the training and validation results for when the KB input is used in place of the wave travel distance. Although the difference is minimal, use of the KB inputs reduces performance in nearly every metric for both ANNs. The distribution of the wave travel distance is similar to the KB distribution in relative terms, and since the order of magnitude of these distances is similar to the directional inputs, this standardised input set is likely to benefit the optimisation of the weights and biases. The following section will therefore explore if normalisation of all inputs changes the outcome of this assessment when using the KB input.

Table 6.10: DeNN performance comparison between using the wave travel distance as an input and when this is translated to an equivalent free air incident overpressure generated by the KB method. Rotating laser directions, the superposition equation, mirrored dataset and multiple ANNs are included in input generation. Best statistics shown in bold.

DeNN inputs	Stage	ANN num.	$R_t^2$	MAE (kPa)	Avr%E
16 directional lasers, shortest wave travel distance	Training	1	<b>0.9950</b>	<b>3.26</b>	<b>6.7</b>
		2	<b>0.9620</b>	<b>3.04</b>	17.3
	Validation	1	<b>0.9946</b>	<b>3.30</b>	<b>6.7</b>
		2	<b>0.9600</b>	<b>3.09</b>	17.6
16 directional lasers, KB incident overpressure prediction at the shortest wave travel distance	Training	1	0.9937	3.38	6.8
		2	0.9582	3.05	<b>16.6</b>
	Validation	1	0.9935	3.41	6.8
		2	0.9563	3.11	<b>16.8</b>

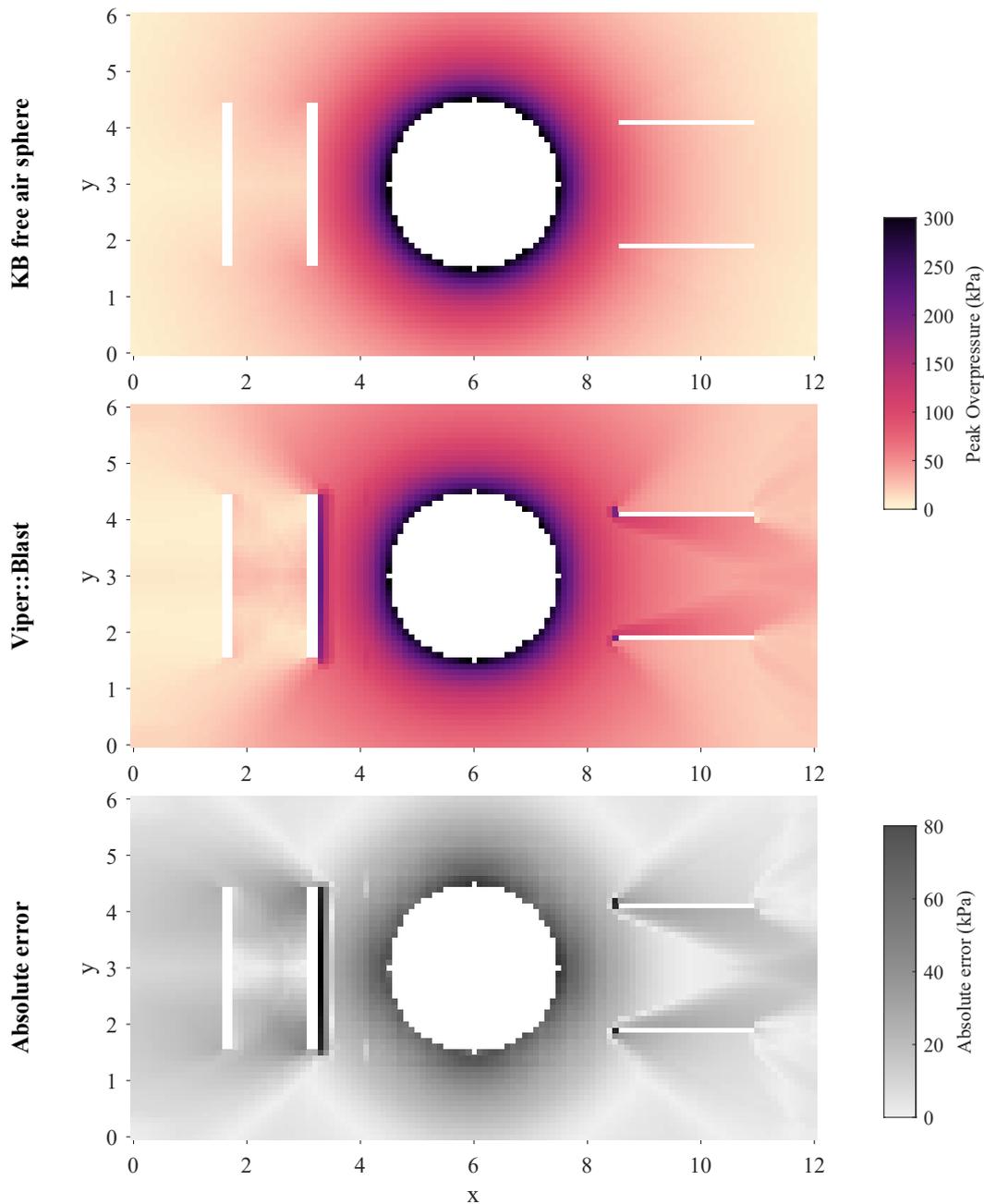


Figure 6.19: Comparison between the peak incident overpressure from a spherical, free air, 1 kg TNT charge calculated for each point using the shortest wave travel distance as the stand-off in the KB method, and Viper::Blast, where the domain geometry is included to account for wave coalescence and reflections.

### 6.5.7 Input and output normalisation

As discussed in Section 2.4.4, a common method of improving training times and/or performance of ML tools involves normalisation and standardisation of inputs and outputs. When providing an input pattern to the network, the magnitudes of each variable will alter how the weights and biases of each layer are adjusted to understand the process being modelled, and this could reduce the time required by the iterative process to find a global minima that corresponds to the lowest predictive error.

To explore this, the following equation shows the power transformation that is applied to all inputs and the peak overpressure output to normalise the data prior to training and validation.

$$x' = \ln(x + 1) \quad (6.5)$$

Where  $x$  is the original value and  $x'$  is the normalised value. The constant is applied within the natural log function to allow for 0 values to be processed. There are many different forms or power transformation, however, it is not practical to test every option, nor is it the focus of this study. The chosen method replicates the Yeo-Johnson transformation introduced by Yeo & Johnson (2000), when  $\lambda = 0$ .

To assess the skew of each input and output variable, Figure 6.20 displays histograms before and after Equation 6.5 is applied. Skewness values, calculated using the following equation, are given for each plot to highlight how each distribution is normalised.

$$s = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^3}{\left(\sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}\right)^3} \quad (6.6)$$

Where  $s$  is skewness,  $n$  is the number of samples in the dataset,  $\bar{x}$  is the mean and  $x$  is an individual sample. Positive skew is present if the value of  $s$  is positive, meaning the data is shifted to the left of the mean. Conversely, negative skew corresponds to negative  $s$  values, with a value of 0 indicating that the distribution is perfectly normal.

Figure 6.20 shows that each potential input to the DeNN is transformed to have a distribution with a skewness closer to 0. Each variable is also defined by smaller ranges that are of the same order of magnitude. Most directional inputs are omitted from the plot since they have a similar profile to Direction 2, where the large number of 0 inputs remains an issue with obtaining a truly normalised profile.

Table 6.11 provides the training and validation metrics for when no normalisation is applied, when normalisation is applied to a network that uses the wave travel distance, and finally when normalisation is applied to a network using the KB incident overpressure. Overall, normalisation has a detrimental effect to the DeNN in its current form, however, using the normalised KB input results in improved predictions when compared to normalising the wave travel distance. This suggests that standardising each variable has benefit the inclusion of a physically derived value that can be scaled by the network to account for various obstacles and wave coalescence effects in the domain.

It is likely that the optimum performance from these trials is observed without normalisation since the ReLU activation function is used for all hidden neurons in both networks forming the DeNN. Any variations in the scales of the inputs are therefore handled by adjustments to the weights and biases associated to the first layer of neurons. In theory, this enables the networks to find optimal parameters provided that enough training steps are implemented. Conversely, for networks using activations such as the hyperbolic tangent, standardisation becomes critical as the vanishing gradient problem could prevent some weights from changing their values, thus preventing an optimum network from being identified.

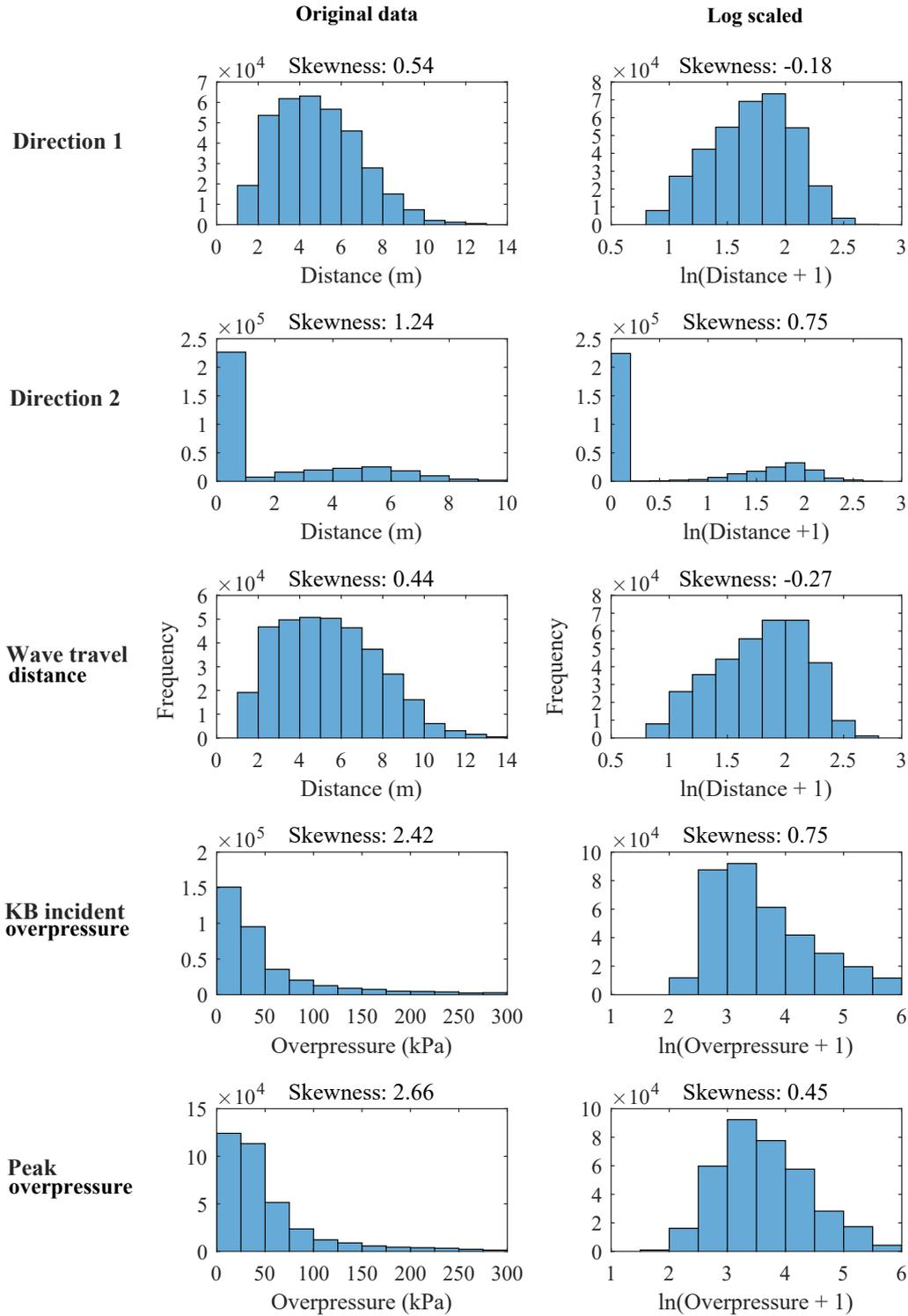


Figure 6.20: Histograms showing the data distributions for three inputs and the peak overpressure output, before and after a log transformation.

As before, a detailed comparison between each potential network architecture is not provided as this is beyond the scope of this study. Instead, focus is placed on the feature engineering process to prove the concept of the Direction-encoded neural network that will now be used without normalisation or the KB input.

Table 6.11: DeNN performance statistics comparing when no normalisation is applied to inputs or outputs versus when log based normalisation is used. Rotating laser directions, the superposition equation, mirrored dataset, multiple ANNs and 16 directional lasers are included in input generation. Best statistics shown in bold.

DeNN inputs	Stage	ANN num.	$R_t^2$	MAE (kPa)	Avr%E
16 directional lasers, shortest wave travel distance	Training	1	<b>0.9950</b>	<b>3.26</b>	<b>6.7</b>
		2	<b>0.9620</b>	<b>3.04</b>	<b>17.3</b>
	Validation	1	<b>0.9946</b>	<b>3.30</b>	<b>6.7</b>
		2	<b>0.9600</b>	<b>3.09</b>	<b>17.6</b>
<b>Normalised:</b> 16 directional lasers, shortest wave travel distance	Training	1	0.9713	5.88	9.6
		2	0.9028	4.35	22.8
	Validation	1	0.9713	5.89	9.6
		2	0.9027	4.36	22.8
<b>Normalised:</b> 16 directional lasers, <b>KB incident overpressure</b>	Training	1	0.9750	5.34	9.2
		2	0.9115	4.25	22.4
	Validation	1	0.9750	5.34	9.2
		2	0.9113	4.26	22.4

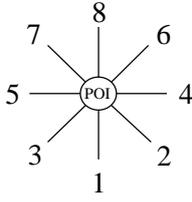
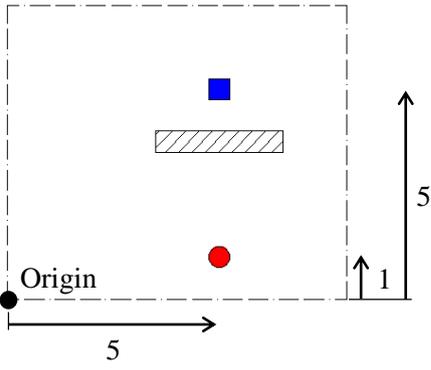
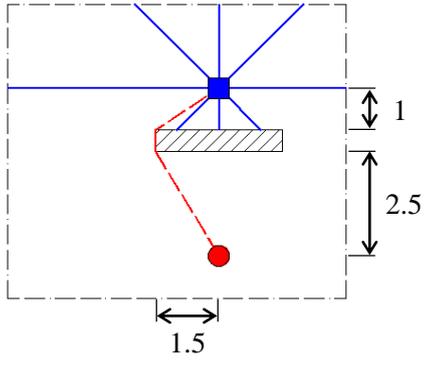
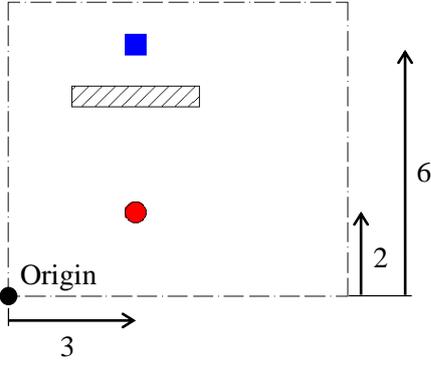
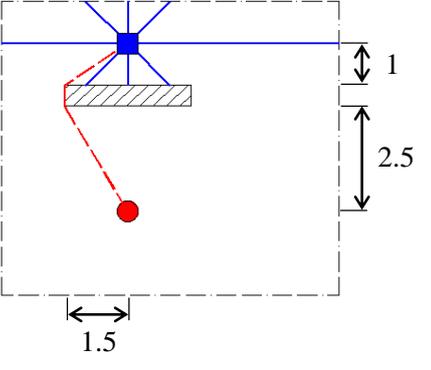
### 6.5.8 Summary of performance improvements

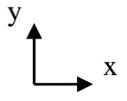
In summary, the feature engineering process conducted for this thesis has developed the initial DeNN architecture to include inputs that represent the physics of the wave coalescence problem with a greater ability for generalisation with unseen inputs. Notably this includes a superposition equation that accounts for wave reflections in front of, and around, various obstacles that are present in any given domain. The progression of the approach, showing the adopted developments, is shown in Table 6.12 in terms of testing performance to highlight the impact of each feature alteration on points outside of the training dataset.

Table 6.12: Performance progression with testing models throughout the feature engineering process for the DeNN. Predictions of peak overpressure.

DeNN feature progression	T1		T2	
	MAE (kPa)	Avr%E	MAE (kPa)	Avr%E
Initial feature design	16.45	51.0	10.65	44.2
+ Rotating lasers and mirrored dataset	10.28	31.8	9.33	32.1
+ Superposition equation	5.53	22.3	5.68	27.1
+ Multiple ANNs	5.19	21.4	5.64	27.6
+ 16 directional inputs	4.84	18.2	5.04	24.5

A notable improvement of this method when compared to a Cartesian based approach is shown in Figure 6.21, where it is shown that a Cartesian network fails to provide equal input patterns for two points that should be predicted with identical output predictions.

	<b>Cartesian structured network</b>	<b>Direction-encoded Neural Network</b>
<b>Inputs</b>	POI x coordinate, POI y coordinate, charge x coordinate, charge y coordinate.	8 directional inputs, shortest wave travel distance 
<b>Point A</b>	 $I_A = [ 5, 5, 5, 1 ]$	 $I_A = [ 4.2, 3.8, 3.8, 0, 0, 0, 0, 0, 5.2 ]$
<b>Point B</b>	 $I_B = [ 3, 6, 3, 2 ]$	 $I_B = [ 4.2, 3.8, 3.8, 0, 0, 0, 0, 0, 5.2 ]$
<b>Comp.</b>	$I_A \neq I_B$	$I_A == I_B$



- Charge location ●
- Prediction point, P ■
- Directional laser —
- Ambient boundary - - -
- Shortest wave travel path - - -
- Rigid object ▨

All dimensions in meters. All rigid objects are 0.5 m wide.

Figure 6.21: A comparison of the input patterns that are generated for the prediction of peak overpressure when using the DeNN and a Cartesian based network.

Conversely, the DeNN assigns the same inputs regardless of domain position, thus allowing for its use in domains of varied sizes, with movable obstacles. It should be noted that the procedure for ANN development discussed in this document is one that was devised by the author to allow for efficient analysis of the various options. Clearly there are a multitude of other approaches and functions that could achieve improved performance, however, it is not feasible to test every combination given that the focus of this study is to prove the concept of the DeNN.

## 6.6 Hyperparameter tuning

With any machine learning application there are a number of different hyperparameters that can be tuned to provide an optimal network structure for each unique application. For ANNs, this includes the number of hidden neurons, number of hidden layers, dropout rate and the optimiser learning rate. Conducting a grid-search to test every possible combination would be very time consuming and computationally expensive, so for this application a range of setup variables, shown in Table 6.13, are fixed throughout the tuning process. Each parameter is matched to those used by Dennis et al. (2021), since this study showed that they are suitable for generating accurate predictions for a similar blast application.

Table 6.13: Fixed network parameters.

Output Activation function	Linear
Loss function	Mean squared error (MSE)
Training steps	500 (with early stopping if validation loss does not improve for 10 steps)
Batch size	100
Regularisation	L2
Weight initialiser	Glorot Normal
Bias initialiser	Zeros
Cross validation folds	4

The KerasTuner is applied using the hyperband optimisation process to identify an optimal combination of the hyperparameters being trialled in this study (O'Malley et al. 2019, Li et al. 2018). Table 6.14 provides each variable with the associated sampling method, or step size, and potential values. The ranges of each variable were reduced based on initial optimisation tests and in the case of hidden neurons and layers, limitations are applied to prevent prohibitively large training times and overfitting due to excessive model complexity. It is appreciated that this could prevent the performance of the approach from being fully realised, however, as mentioned previously, it is not practicable to explore all hyperparameter combinations.

The AdaGrad (Adaptive Sub-gradient Descent) optimiser remains as an option for training since it has proved to work effectively for blast applications so far in this chapter. Similarly, Zahedi & Golchin (2022) notes that the Adam optimiser tends to perform well for most studies, providing a useful alternative.

Tuned parameters are provided in Table 6.15 for ANN-1 and ANN-2.

Table 6.14: Tuned hyperparameter options, ranges and sampling methods.

Variable	Tuning range / options	Step size / sampling method
Hidden layers	2–4	1
Hidden neurons	500–1000	50
Activation function	ReLU, ELU, SELU	Random
Optimiser	AdaGrad, Adam	Random
Learning rate	0.001–0.1	log
Dropout rate	0–0.2	linear

Table 6.15: Tuned hyperparameters for the developed DeNNs.

Variable	ANN-1	ANN-2
Hidden layers	4	4
Hidden neurons	550   900   550   800	800   650   950   600
Activation function	ReLU	ReLU
Optimiser	AdaGrad	AdaGrad
Learning rate	0.0170	0.0033
Dropout rate	0.0290	0.0139

## 6.7 Developed performance assessment

### 6.7.1 Training analysis

Mean performance statistics from the 4-fold cross validation technique are reported in Table 6.16 for the training and validation stages of both ANN-1 and ANN-2 using the tuned network parameters.

Table 6.16: Mean performance metrics from 4-fold cross validation of the tuned, developed DeNN.

Development stage	ANN number	$R_t^2$	MAE (kPa)	Average Error (%)
Training	1	0.9973	2.63	5.6
	2	0.9766	2.38	13.6
Validation	1	0.9967	2.74	5.6
	2	0.9734	2.49	14.0

Considering points that are not used to iteratively update the weights and biases, the average error for points that are unobstructed (ANN-1) is 5.6%, corresponding to a MAE of 2.74 kPa. On the other hand, the error of obstructed points (ANN-2) is higher at an average of 14.0%, yet this results in a similar MAE of 2.49 kPa (suggesting a higher propensity for lower magnitude pressure values for this network, as discussed previously). Correlation coefficients of around 0.997 for ANN-1 are comparable to those achieved by Remennikov & Rose (2007) for peak pressure predictions behind blast barriers using a

bespoke network structure. However, ANN-2’s correlations around 0.975 suggest that future work should focus on this network’s ability to replicate the relevant wave coalescence effects, especially considering the average errors around 14% are also outside ‘typical’ variations for blast scenarios, being less than 10% (Rigby et al. 2014).

Despite this, there is minimal overfitting as the validation performance is only slightly worse than training in each metric. The dataset is shown to be generalised consistently with different groups of data being held out in four separate training processes.

An assessment of the variation in error from this training and validation process is displayed in Table 6.17. It shows that lower magnitudes are commonly predicted with higher errors compared to those with larger magnitudes, despite the dataset featuring more points in the lower overpressure ranges. ANN-2, used for predictions of obstructed points, handles a larger number of these lower magnitude targets, hence increasing its average error and decreasing the reported performance.

Table 6.17: Variation of tuned network predictive performance during validation relative to the target peak overpressure magnitude.

Target overpressure range (kPa)	Number dataset points	Percentage of validation dataset points predicted within percentage error range (%)			
		$E < 5$	$5 \leq E < 10$	$10 \leq E < 30$	$E \geq 30$
<b><math>P &lt; 25</math></b>	124128	29.3	21.9	39.0	9.9
<b><math>25 \leq P &lt; 50</math></b>	113342	51.8	23.5	22.9	1.8
<b><math>50 \leq P &lt; 100</math></b>	75042	81.3	9.3	8.9	0.4
<b><math>100 \leq P &lt; 200</math></b>	31090	88.7	7.3	3.7	0.3
<b><math>200 \leq P &lt; 300</math></b>	10498	86.1	12.1	1.8	0.1
<b><math>P \geq 300</math></b>	454	75.3	15.0	5.5	4.2

The table also shows how there are only 454 (0.13% of the dataset) targets over 300 kPa, corresponding to around 10% of points having errors over 10% in this overpressure range. The low number of input patterns associated to this range means that the DeNN does not update its weights and biases to suit predictions of this magnitude very often, restricting its ability to accurately account for the needed pressure amplification. However, for points between 50 kPa and 300 kPa, performance is generally very good with average errors of less than 5% for over 81% of the points in each these ranges. Furthermore, over the same targets, less than 0.5% of points are predicted with errors over 30% giving confidence that the DeNN can account for wave interaction effects appropriately.

### 6.7.2 Testing analysis

Performance metrics obtained when using the tuned and developed DeNN to predict both testing domains are provided in Table 6.18. There is a decrease in performance compared to the validation and training metrics, yet this is expected considering how the testing models were structured without the same restrictions that were applied to the randomised training dataset.

An average MAE around  $\sim 4.5$  kPa and correlation coefficients over 0.99 for both domains proves that the DeNN has been able to successfully use its training to generalise for unseen, and independent, inputs. However, average errors throughout both domains are over the 10% target, due to ANN-2 large error contribution of over 40% in both instances. As before, these errors are coupled with low absolute errors suggesting that the points being predicted by this network are of very low magnitude.

Table 6.18: Performance metrics for both testing domains, simulated using the DeNN, compared to a Viper model.

Testing model	ANN number	$R_t^2$	MAE (kPa)	Average Error (%)
1	1	0.9961	4.18	5.4
	2	0.8989	4.48	42.1
	Overall	0.9952	4.26	16.0
2	1	0.9971	3.90	7.1
	2	0.8070	6.83	57.6
	Overall	0.9938	4.86	23.6

Figure 6.22 presents heat maps for T1, showing the Viper outputs, DeNN predictions and the resulting absolute errors. The wave superposition equation and rotating lasers are proved to work effectively since the DeNN achieves good symmetrical consistency and the magnitude of pressure amplification, caused by wave reflections in front of rigid obstacles, is replicated appropriately across the majority of each surface. As mentioned previously, absolute error are generally very low, aside from the regions where multiple surfaces are close to one another. Here, errors approach 80 kPa, yet this does not prevent to DeNN from qualitatively representing the distribution of peak overpressure with high accuracy.

Conversely, Figure 6.23, which shows heat maps T2, highlights that channelling is only partially considered. Errors are low where  $x = 10$  and  $y = 3$ , but beyond this as  $x$  reaches 12, the amplification effects are not as accurate. Additionally, the shape of the high magnitude regions of the DeNN heat maps display some angularity, suggesting that use of 16 lasers contributes to slight local inconsistencies when obstacles are not captured in the input pattern correctly for adjacent points. Despite this, once again, the domain is qualitatively represented by the DeNN with good accuracy, indicating regions where pressure is reduced due to shielding and clearing, whilst amplifying the predictions in front of surfaces.

Table 6.19 shows that the trends within the data are consistent from training to testing when assessing the variation in prediction errors. It is clear that the smaller overpressures are being predicted with the largest percentage errors, but lower absolute errors and that ANN-2 is responsible for the majority of the lower magnitude predictions. Future developments should therefore focus on this network's ability to handle shielding and clearing effects that can lead to reductions in the peak overpressure when compared to a free air prediction. Targets in the range of 50–200 kPa are once again predicted with good accuracy since the percentage of points with less than 5% error is around 80%, and round 90% are predicted within the desired  $<10\%$  bracket.

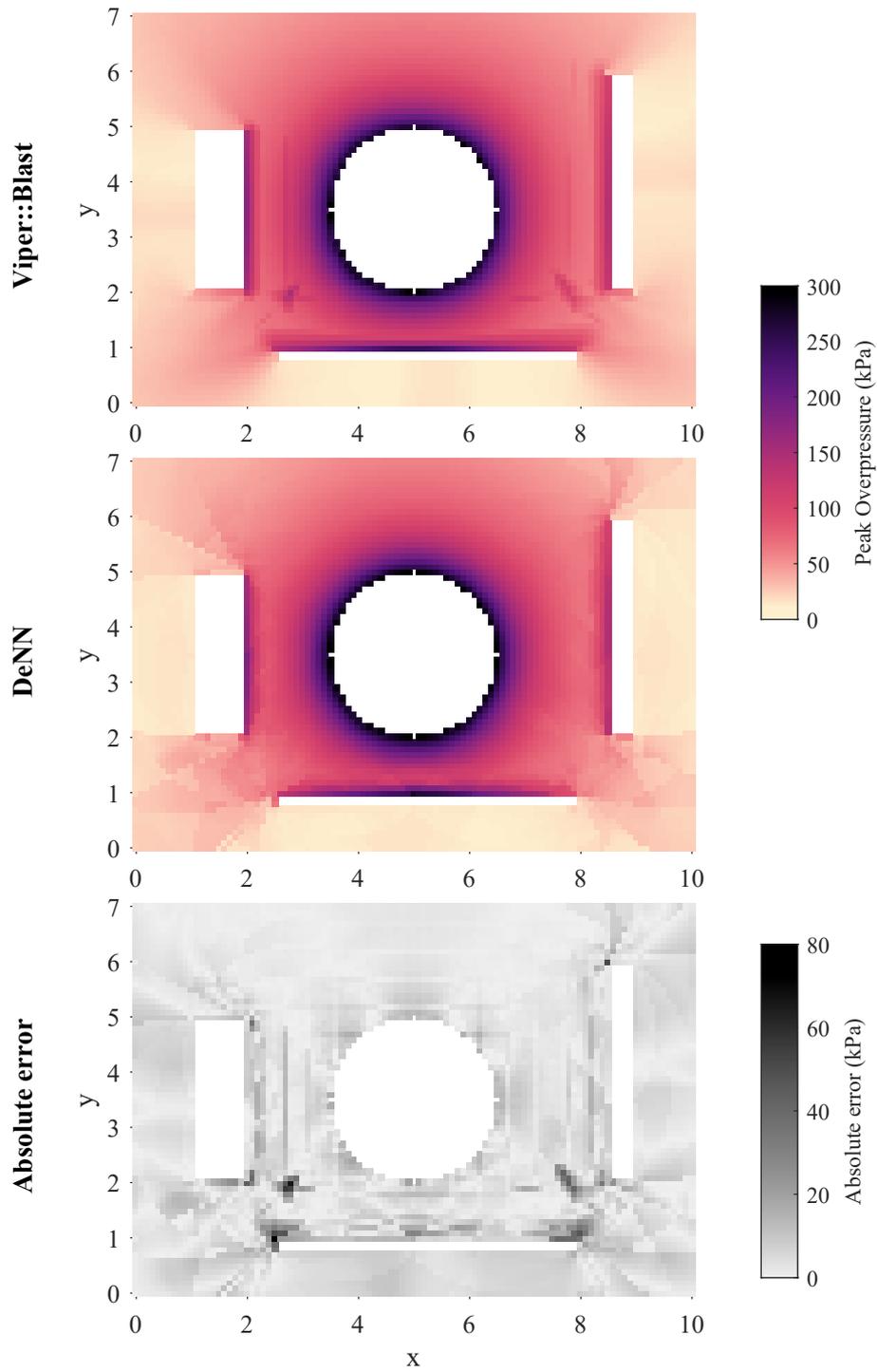


Figure 6.22: T1 peak overpressure targets generated by Viper::Blast, predictions from the DeNN, and the resulting absolute errors. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone.

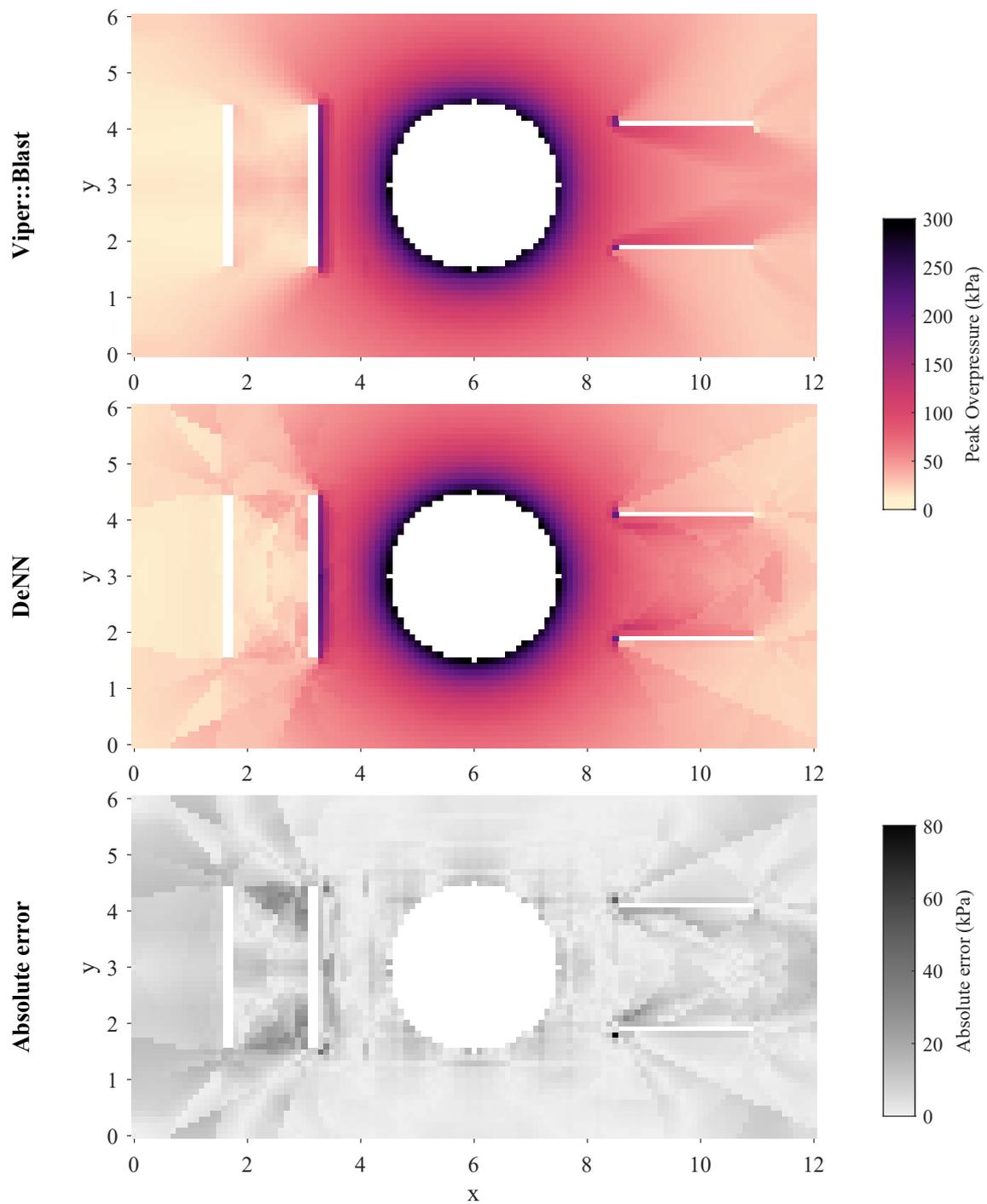


Figure 6.23: T2 peak overpressure targets generated by Viper::Blast, predictions from the DeNN, and the resulting absolute errors. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone.

Table 6.19: Variation of network predictive performance for both testing models relative to the target peak overpressure magnitude.

Testing model	Target over-pressure range (kPa)	Number dataset points	Percentage of points predicted within percentage error range (%)				
			$E < 5$	$5 \leq E < 10$	$10 \leq E < 30$	$E \geq 30$	
1	$P < 25$	1560	10.4	9.7	24.7	55.1	
	$25 \leq P < 50$	1160	40.3	19.4	36.0	4.3	
	$50 \leq P < 100$	1773	74.3	13.2	11.4	1.0	
	$100 \leq P < 200$	1136	80.0	10.7	7.8	1.5	
	$P \geq 200$	310	85.5	14.5	0.0	0.0	
2	$P < 25$	1925	7.7	9.6	19.3	63.4	
	$25 \leq P < 50$	1407	26.9	16.6	50.6	6.0	
	$50 \leq P < 100$	1906	77.9	11.4	10.4	0.3	
	$100 \leq P < 200$	972	88.4	7.1	4.5	0.0	
	$P \geq 200$	301	87.0	13.0	0.0	0.0	

Overall, the achieved performance highlights that physics-informed structuring of the input data provided to machine learning tools can produce accurate FREMs that can be applied to a range of domains. An appreciation for the specific application of the model being developed is essential for understanding how various features of the problem should be represented so that the ANNs can effectively learn from the process and replicate the complex interaction processes.

The next section will explore how this level of performance can be leveraged to allow the the DeNN to be used for rapid human injury assessments in its current form.

## 6.8 Application: Ear drum rupture

To highlight a potential use case for the DeNN, eardrum rupture is predicted according to the rupture levels given by Table 6.20. This information was compiled by Denny et al. (2021*a*), and has been used in assessments presented by Denny et al. (2021*b*, 2022).

Table 6.20: Overpressure eardrum rupture limits (Denny et al. 2021*a*).

Rupture level	Overpressure (kPa)
Threshold	35
50% chance	103
100% chance	202

This criteria is chosen as it relies on peak overpressure, the only parameter involved in the development of the DeNN so far. It is also the injury criteria that will indicate the regions where other injuries are likely to be experienced, since overpressures below 35 kPa will not cause injury from the blast wave itself.

As shown by Figure 6.24, the DeNN provides a very good qualitative representation of the various injury zones for T1 when compared to Viper. Only slight variations in output are observed as 95% of points are predicted in the correct rupture category. The remaining 5% are predicted with only one level of error. Regions of shielding are predicted with a ‘no rupture’ designation and transition zones (where diffraction occurs) are predicted with a threshold rupture level as the overpressure begins to increase with reduced stand-off distance. The 50% and 100% chance regions are formed in the correct locations, in front of the rigid surfaces and directly around the charge.

Rupture predictions for T2 are shown in Figure 6.25. Here, 93% of points are predicted correctly by the DeNN and the remaining 7% are predicted with only one level of error. The aforementioned issue related to incorrectly amplifying pressure due to channelling as  $x = 10$  and  $y = 11$  is captured in the DeNN’s predictions, and some further inconsistencies are present between the two rigid obstacles to the left of the charge. However, again, the domain is qualitatively well represented.

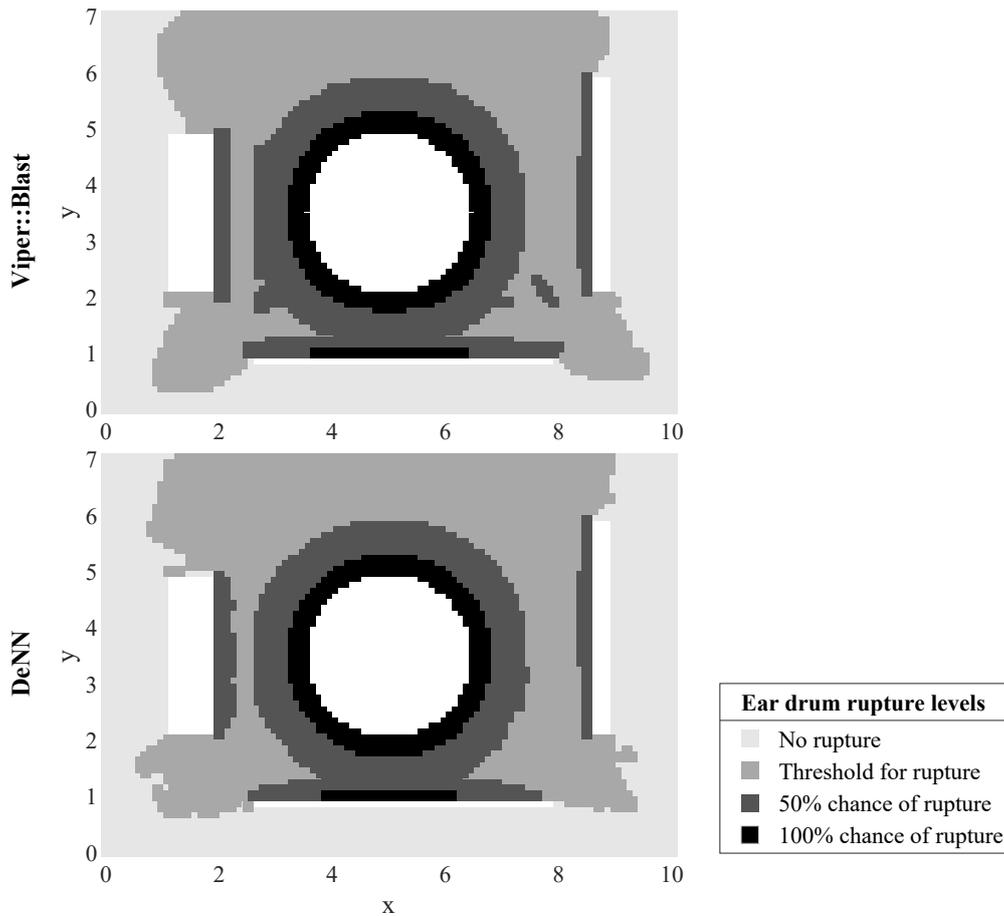


Figure 6.24: Eardrum rupture levels for T1 calculated by the DeNN and Viper. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone.

This shows that the DeNN, unlike previous task-specific ML tools, could be used in probabilistic assessments that require varied charge and obstacle positions to be evaluated. Its flexibility, achieved using inputs that reference the surrounding and not the domain itself, allows for obstacles to be moved, added or removed, with predictions being generated in under 60 seconds for an entire domain. Compared to Viper, this allows for upwards of 30 unique layouts to be evaluated in the time that it would take to run a single numerical model from birth to termination. Ultimately, this enables decisions to be made rapidly regarding the structural layout of an area, or the regions to focus a response to an explosive event.

There are, of course, other applications where the DeNN cannot be used in its current form for probabilistic analyses, and numerical models should be evaluated using Viper or similar solvers. These include if time histories are required, or if non-rectangular or frangible/non-rigid obstacles are present. Conversely, it would be possible for the current form of the DeNN to be retrained for charge and output locations at different elevations, or for the prediction of values relating to another blast wave parameter that is required by more robust human injury assessments (see next section). However, these alterations may require the generation of a new training dataset to train new networks.

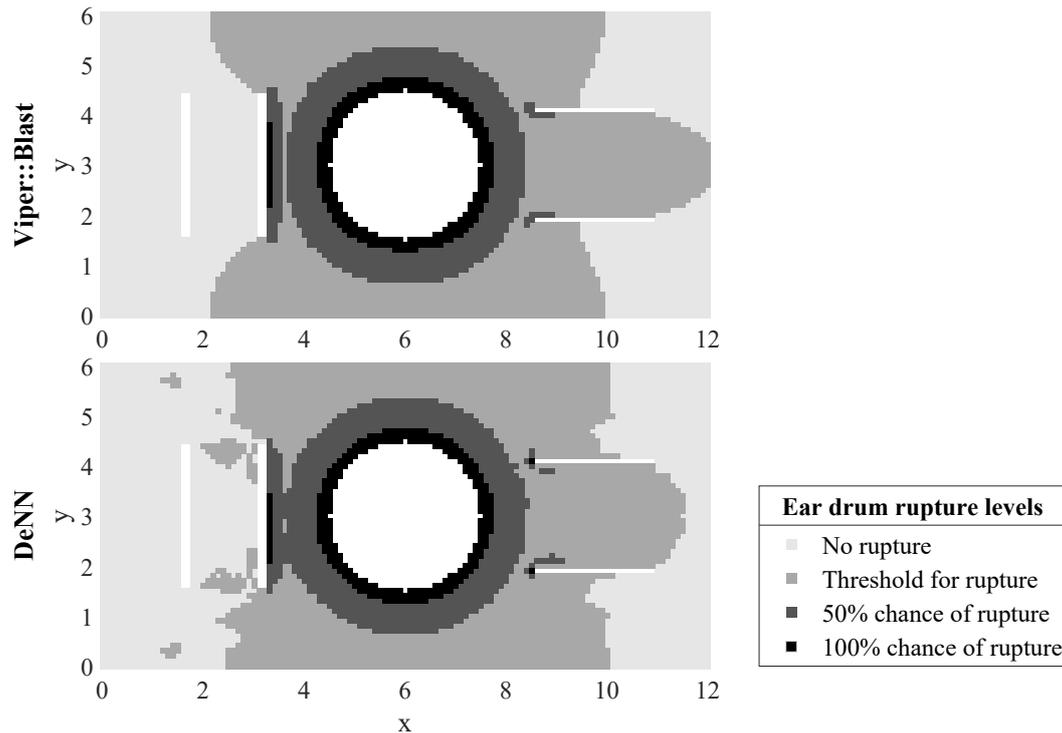


Figure 6.25: Eardrum rupture levels for T2 calculated by the DeNN and Viper. White regions are not predicted, either due to being within a rigid obstacle, or the 1.5 m exclusion zone.

## 6.9 Alternative parameter predictions

This chapter has introduced and developed the DeNN as a tool that generates peak overpressure predictions for domains featuring a number of obstacles, however, retraining the ANNs with different outputs enables them to be used to predict other important variables. This section therefore explores the predictive accuracy of the tuned network architecture when the targets are peak specific impulse, and the blast wave time of arrival.

Statistics associated to each target variable within the training dataset are shown in Table 6.21. Peak specific impulse has a standard deviation around half of the equivalent value for peak overpressure, highlighting how the range of targets is more concentrated around the mean. The mean is also less skewed towards the minimum value and so the DeNN may have difficulty in discerning when an obstacle being translated to its inputs via the directional lasers should alter the predicted value. The distribution of the time of arrival is harder to compare to the other parameters due to the scale change, however, it has a very high correlation with the shortest wave travel distance in the training dataset, effectively creating a linear regression problem where predictions will be largely dependant upon this input value.

Table 6.22 shows the training, validation and testing performance of the DeNN when predicting peak specific impulse. In training and validation the performance is excellent, with average errors less than 10% for all points. This relates to an average MAE that is around double that of the peak overpressure network. However, Table 6.21 showed that there are fewer specific impulse targets with low magnitudes where this MAE would

Table 6.21: Training dataset target variable statistics.

Variable	Units	Min	Max	Mean	Std. Deviation
Peak overpressure	kPa	4.72	672.34	51.63	51.20
Peak specific impulse	kPa.ms	4.60	375.55	76.08	26.86
Time of arrival	ms	1.10	33.17	9.53	5.92

contribute to larger percentage errors (i.e. a target of 10 kPa.ms predicted as 5 kPa.ms relates to an absolute error of 5 kPa.ms and percentage error of 100%, whereas a 50 kPa.ms target predicted as 45 kPa.ms also relates to an absolute error 5 kPa.ms, but a percentage error of  $\sim 11\%$ ). This means that relative to the targets, performance is generally better than with peak overpressure.

Table 6.22: Mean performance metrics from 4-fold cross validation of the DeNN when predicting peak specific impulse.

Development stage	ANN number	$R_t^2$	MAE (kPa.ms)	Average Error (%)
Training	1	0.9926	4.98	6.2
	2	0.9862	4.51	8.4
Validation	1	0.9918	5.22	6.5
	2	0.9840	4.82	8.9
T1	1	0.9604	18.06	15.2
	2	0.9170	9.86	26.7
	Overall	0.9581	15.70	18.5
T2	1	0.9816	8.84	10.5
	2	0.8986	13.70	30.2
	Overall	0.9681	10.44	17.0

Conversely, the obtained testing metrics do not replicate the validation performance for unseen data. This once again suggests that the testing models are not representative of the domains that were randomised to form the training dataset, and so the DeNN is being required to form predictions based on input patterns that it has not seen frequently in its training process. A more comprehensive dataset, featuring a greater number of domains and varied structural arrangements, may help with this issue. However, training with non-randomised models in a probabilistic framework would ensure that the DeNN develops an understanding of the specific wave interaction effects that are present in a set of similar domains. Chapter 7 explores this option as a means of reducing the decline in performance from validation to testing.

Performance when predicting time of arrival of the blast wave at each POI is given by Table 6.23. It is clear that the DeNN is able to predict this variable with very high accuracy as errors are around 3% and correlation coefficients are over 0.999 in all phases of the network's development. As expected, the reduced complexity associated with this parameter, that is closely linked to the shortest wave travel distance input, allows for good generalisation for unseen inputs.

Table 6.23: Mean performance metrics from 4-fold cross validation of the DeNN when predicting time of arrival.

Development stage	ANN number	$R_t^2$	MAE (ms)	Average Error (%)
Training	1	0.9994	0.16	3.2
	2	0.9996	0.24	1.9
Validation	1	0.9994	0.16	3.2
	2	0.9996	0.24	1.9
T1	1	0.9995	0.08	3.2
	2	0.9995	0.18	1.8
	Overall	0.9995	0.11	2.8
T2	1	0.9994	0.09	3.3
	2	0.9993	0.19	2.0
	Overall	0.9993	0.12	2.9

The DeNN was developed with the primary goal of providing peak overpressure predictions in domains featuring various obstacles. Predictions of peak specific impulse and time of arrival may therefore benefit from an alternative featuring engineering process that tailors the inputs and network hyperparameters with greater consideration for the distribution of each variable in a given domain. For example, it is not likely that the wave superposition equation impacts the predictions of time of arrival, however, knowledge of the number of bends in the wave’s path to the POI could improve accuracy.

Nevertheless, this section has shown that the DeNN framework is not exclusively applicable to predictions of peak overpressure. The novel approach to creating a generalised ML tool can be applied to the prediction of other variables, which could in turn, develop into more robust probabilistic assessments of risk and human injury where alternative combinations of parameters are required to determine pulmonary injuries and fatalities.

## 6.10 Summary

In summary, the Direction-encoded Framework is introduced as a novel approach to providing geometrical information to a ML algorithm, such that it’s application to a neural network (referred to as the Direction-encoded Neural Network in this thesis) enables predictions to be produced for domains of any shape and size. Unlike previous applications of ANNs in blast engineering, that resulted in the development of bespoke tools without this level of generalisation, the framework removes the need to encode geometrical information into the network’s architecture when predicting peak overpressure. This is achieved through task focussed feature engineering that provides the ML algorithm with each point’s proximity to surrounding obstacles and the blast wave’s shortest travel distance.

Following a training process using data from 25 randomised domains, it is shown that the DeNN can accurately predict regions of pressure enhancements through reflection and coalescence in addition to shielding and clearing when testing two independent models, with

differing structural arrangements. Overpressure distributions throughout both domains were formed in under 60 seconds, with mean absolute errors less than  $\sim 5$  kPa and correlation coefficients above 0.993. Translating this into eardrum rupture levels resulted in over 93% of points being correctly categorised with the remaining percentage only having one level of error.

The inability of other FREMs to represent complex domain geometries in a general sense limits their application to studies involving only simple geometries and wave interaction effects. Since this is not a limitation of the DeNN, it is now feasible to conduct probabilistic assessments, where many domains, featuring various structural layouts, need to be simulated rapidly. However, it should be noted that at present, charge height and obstacle materials cannot be probabilistically assessed without further training to enhance the current tool's capabilities. With reference to the plot shown in Figure 2.11, the introduction of the Direction-encoded Framework improves model versatility for ML based tools that are used for blast load prediction, as shown by Figure 6.26. Consequently, *Objective 4* of this study has been met.

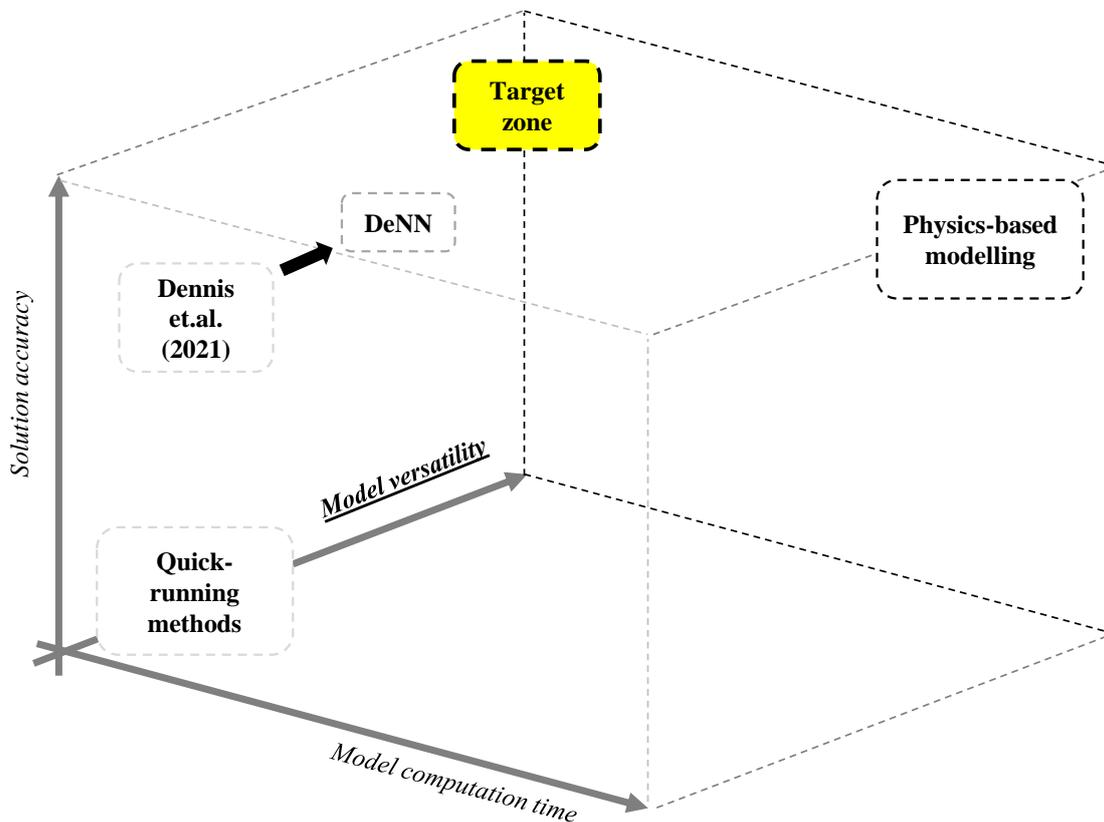


Figure 6.26: Improved versatility of the Direction-encoded Neural Network shown in relation to its computation time and solution accuracy when compared to the Cartesian approach taken by Dennis et al. (2021).

Since all comparison metrics from training and validation outperformed those obtained throughout testing, the independent testing domains are likely to have provided input patterns that were not common in the training dataset, thus requiring extensive interpolation. Consequently, use of a structured training dataset may improve consistency of predictions with unseen inputs. As probabilistic assessments commonly feature similar

domain layouts, data could be taken from the batch to train a new DeNN that could replace the numerical solver part-way through the computation process. Furthermore, the DeNN can be used to generate a prediction for a single point, or a series of points, instead of an entire domain, making it useful for when risk must be assessed at a specific region, such as congregation areas or egress/ingress points.

There are still many opportunities for the predictive performance of the DeNN to be improved, however, this chapter has shown the importance of feature selection in machine learning models by highlighting that prior knowledge of blast engineering can help to form tools that are more suited to the problems faced in the associated field.

The contents of this chapter has been condensed and published in the International Journal of Protective Structures with the title ‘The Direction-encoded Neural Network: A machine learning approach to rapidly predict blast loading in obstructed environments’.

# Chapter 7

## Direction-encoded Neural Network in series (DeNNIS)

### 7.1 Introduction

The Direction-encoded Neural Network can be applied to large batches of unique domains in a number of ways depending on the accuracy and computation time requirements of the associated project. Due to the versatility of the tool, summarised in Figure 7.1, there is no requirement for additional training after the initial process that used 25 randomised domains. However, as mentioned in Chapter 6, it may be possible to improve the predictive accuracy of the DeNN with a second wave of task specific training that fine tunes the network's parameters.

This additional training phase should not be initiated by numerically simulating every domain in the new batch to form the new training dataset, because this would remove the benefit of the DeNN by voiding its versatility advantages over the traditional, Cartesian approaches. Conversely, if no additional training is required, all models of the new batch could be evaluated with the pretrained DeNN, maximising the computational benefit.

A balance between these extremes may present an optimal combination of computation time and predictive accuracy, and so this chapter explores a new approach to developing the DeNN termed the 'Direction-encoded Neural Network In Series' (DeNNIS). By combining the computation time benefits of each novel method introduced in this thesis, the DeNN can be incrementally trained using the dataset that is formed as the batch is evaluated using a numerical solver and the BA. This process continues in series until the DeNN surpasses a desired performance metric, where it then replaces the solver to evaluate the remainder of the batch.

Throughout the following sections the incremental training process is introduced, before various development approaches are compared to determine which analysis option is most suited to the simulation of a new batch of unique domains.

### 7.2 Incremental training framework

Incrementally training the DeNN using data from the BA simulation framework combines the computational benefits of both approaches. This Direction-encoded Neural Network In Series (DeNNIS) procedure, shown in Figure 7.2, removes the need to simulate entire domains using Computational Fluid Dynamics (CFD) if the performance of the ML model surpasses a user-defined target before additional training data is required.

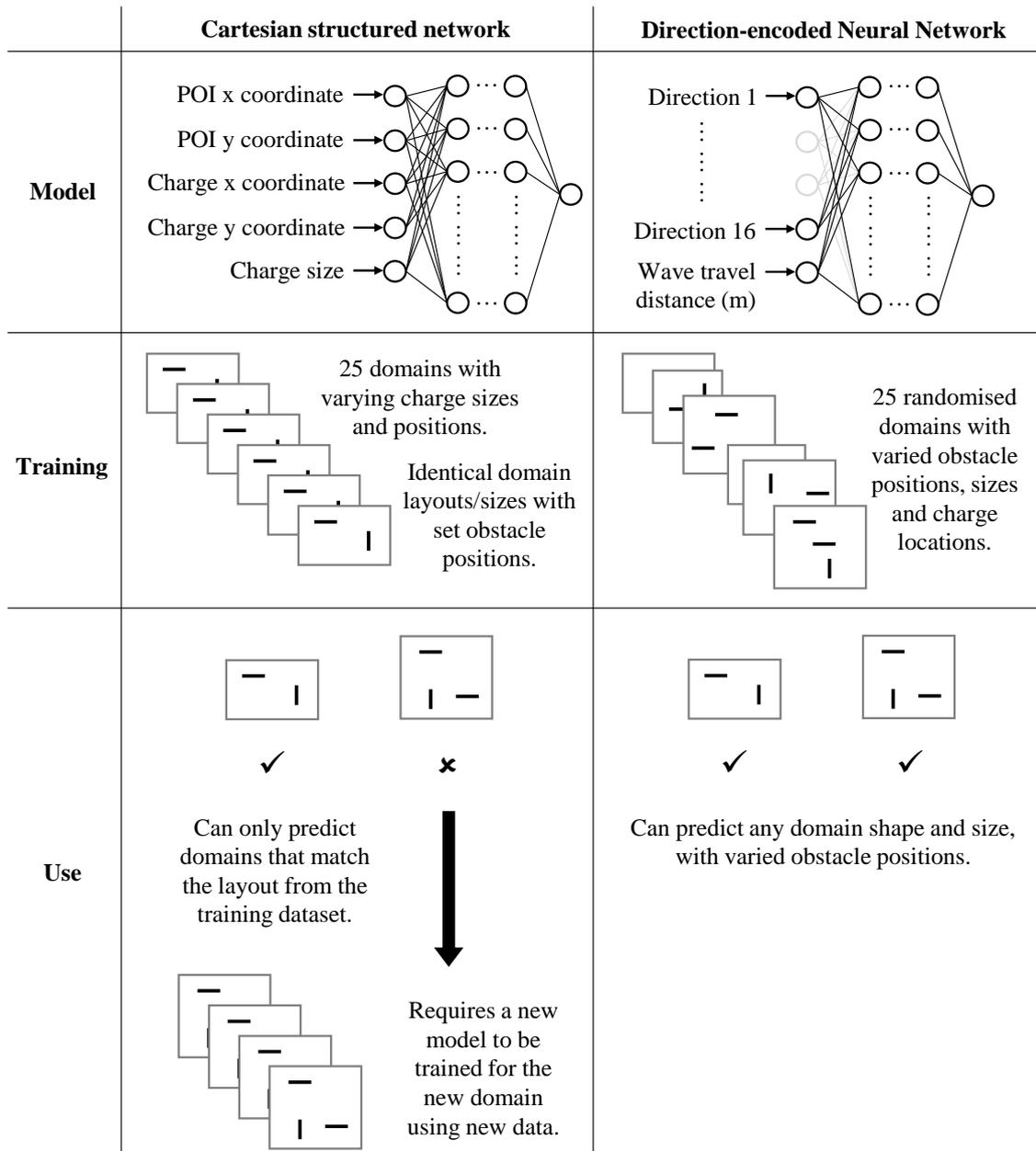


Figure 7.1: Comparison between the development and use of an ANN that uses Cartesian inputs and the Direction-encoded Neural Network, using inputs that are relative to the POI and the charge.

The performance of the ML model is made by evaluating a validation dataset that is formed using values that are independent of the training process. The domains that supply this validation data can be selected in a number of ways, each of which will be introduced and explored in the following sections. Furthermore, in determining if/when the DeNN provides the desired level of predictive accuracy, any performance metrics can be used, including the mean absolute error (Equation 6.2) or average percentage error (Equation 6.3).

Once the required performance is achieved by the DeNN, it replaces the chosen numerical solver to evaluate the remaining models in the batch. Hence, reducing the required computation time. Additionally, both the BA and the DeNN use shortest path analysis following discretisation of each domain in the batch and so combining both approaches with DeNNIS training therefore also reduces the number of calculation steps that are required when compared to running each method independently.

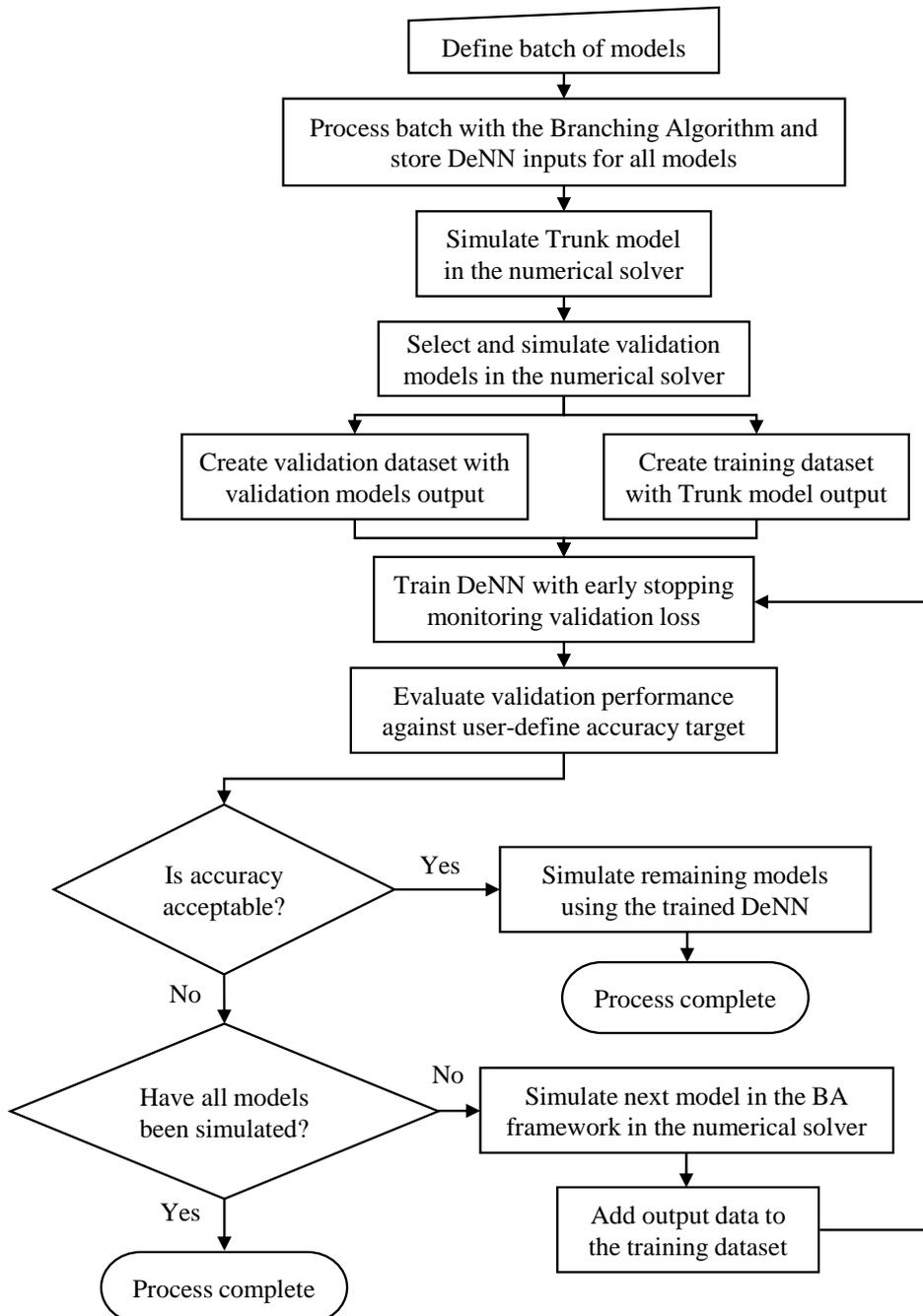


Figure 7.2: DeNNIS procedure allowing for incremental training.

### 7.3 Analysis options

There are various options involved with conducting a DeNNIS analysis that will alter the predictive accuracy of the tool being developed during the incremental training process. Most notably this relates to how the DeNN is initialised before training takes place, how the validation models are selected, and the order that domains are simulated according to the BA framework.

The 12 options that will be compared in this chapter are shown in Table 7.1. The initial state of the DeNN can be ‘untrained’, meaning the weights and biases of ANN-1 and ANN-2 are initialised using the Glorot Normal and zeros functions that were used throughout Chapter 6. Alternatively, the DeNN can be ‘pre-trained’, meaning the parameters are initialised by a separate training process that takes place before the DeNNIS method is applied.

Table 7.1: DeNNIS analysis options.

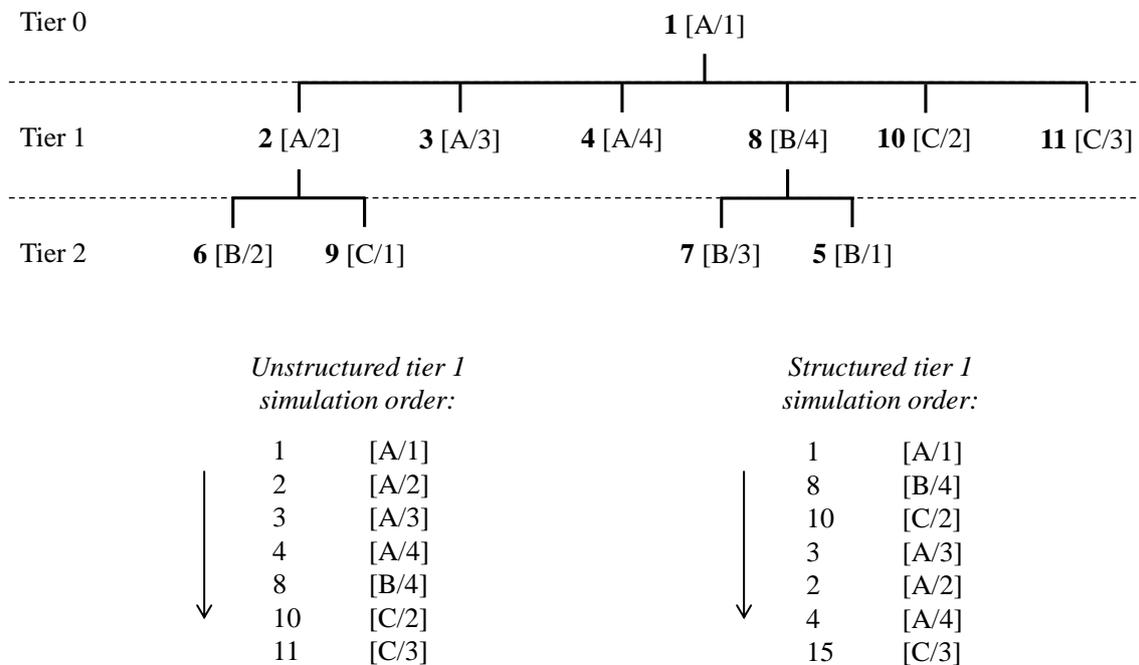
Option	Initial DeNN state	BA training order	Validation selection
1	Untrained	Unstructured	10% from batch
2	Untrained	Unstructured	20% from batch
3	Untrained	Unstructured	One domain per batch geometry
4	Pre-trained	Unstructured	10% from batch
5	Pre-trained	Unstructured	20% from batch
6	Pre-trained	Unstructured	One domain per batch geometry
7	Untrained	Structured	10% from batch
8	Untrained	Structured	20% from batch
9	Untrained	Structured	One domain per batch geometry
10	Pre-trained	Structured	10% from batch
11	Pre-trained	Structured	20% from batch
12	Pre-trained	Structured	One domain per batch geometry

The pretraining approach is commonly seen in the creation of ‘deep fakes’, where a person’s likeness is replaced by another. In this example, the ML tool will be trained on a dataset formed of random people’s faces prior to learning about the specific person being mapped onto the target. This enables it to have an understanding of the features of a face before certain details are introduced for the desired application. In this chapter, pre-training is achieved using the 25 randomised domains discussed in Section 6.3.2, meaning the DeNN will have an understanding of blast wave interactions before the required batch is evaluated. This can also be thought of as a method of transfer learning, where knowledge of previous scenarios are used to benefit the prediction of new, different domains.

Both initialisation methods are trialled alongside various options for how the validation models are randomly selected from the batch. Seeing as training will stop early if there is no improvement in validation loss for 10 steps, where ‘no improvement’ includes variations of  $0.5 \text{ kPa}^2$  or less, the validation models themselves can influence how the DeNN’s predictive performance improves throughout the process by controlling how many training steps are performed.

The final consideration relates to the order of domains being simulated in line with the BA framework. Whilst the BA sorts a batch of domains to effectively reduce computation time, simply evaluating each domain in a sequence overlooks the diversity of the training points that are used to develop the DeNN's accuracy during each iteration.

As shown in Figure 7.3, the approach termed ‘unstructured’, firstly evaluates the trunk model and uses this data to create the training dataset. Next, all models that deviate from the trunk (tier 1) are simulated in numerical order, with non-duplicate points being added to the dataset as needed. This process follows along the tree, passing to each tier to maximise the computational benefit. However, following this process does not guarantee diversity in the dataset since the BA identified domain number 1 to be the trunk with geometry A and charge location 1, but progressing to the first tier of deviated models results in the same geometry contributing to the training dataset multiple times in a row. The performance of the tool may therefore be developed with a limited view of the batch, reducing its ability to generalise for all domain geometries.



Key: **Model number** [Geometry / Charge location]

Figure 7.3: Simulation order options based on the output from the Branching Algorithm considering required diversity in training data points in relation to charge location and domain geometry.

The alternative, ‘structured’, approach requires each tier of the BA tree to be sorted based on the potential diversity that they add to the training dataset. Throughout the current chapter, this process prioritises varied domain geometries, then new charge locations. Any models that do not add to the training dataset diversity are added at the end of the tier in numerical order based on the model number. Figure 7.3 highlights how this process shifts the progression of the simulations to ensure that geometries B and C feature in the DeNN's development at a far earlier increment than with the previous approach.

Conclusions formed from the set of analyses discussed in this section will be highly dependant on the batch of domains being evaluated, however, exploration of each approach aims to identify if a specific method significantly outperforms the others for this application.

### 7.4 Problem scenario

Figure 7.4 presents the batch of domains that will be used to demonstrate the benefits of the DeNNIS method. Four unique geometries and five independent charge locations are included to form a batch of 20 models. All domains are the same size with equivalent charge positions that are at an elevation of 1.5 m.

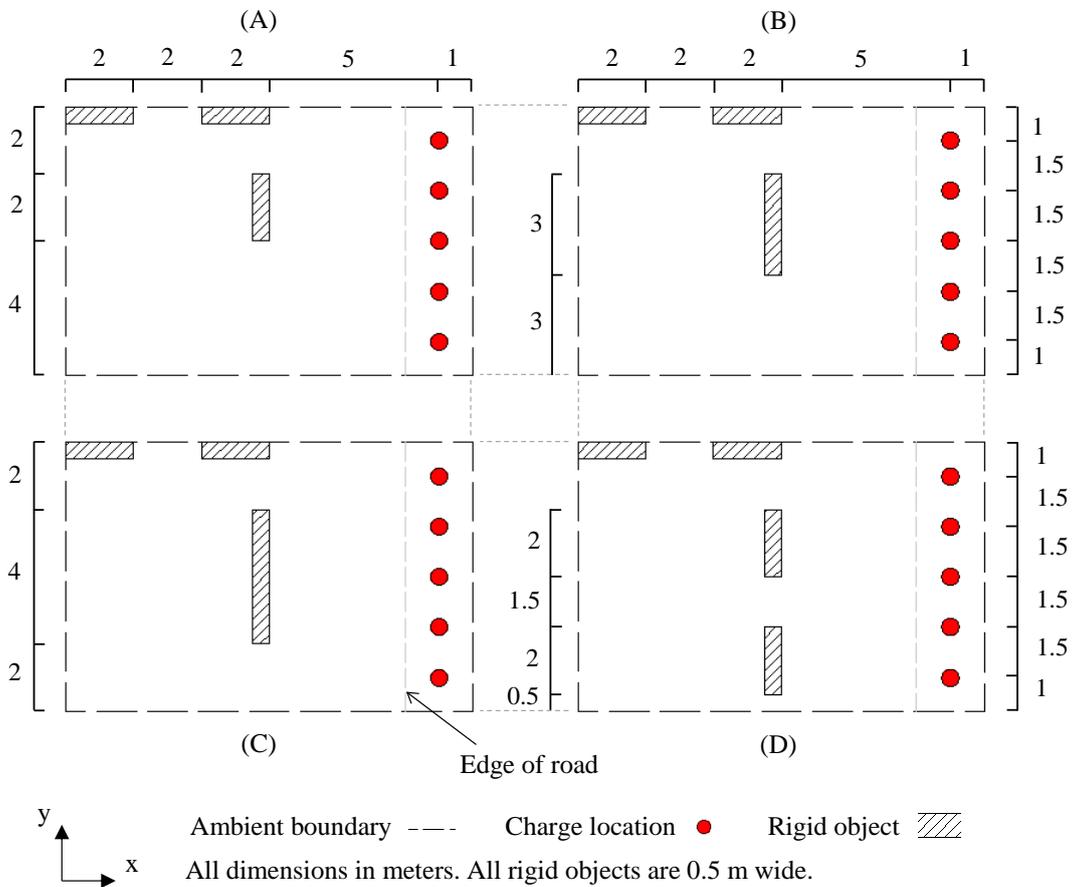


Figure 7.4: Four geometries and five independent charge locations creating 20 models in the batch. Model numbers are as follows: domain A (1–5), domain B (6–10), domain C (11–15), domain D (16–20). Obstacle height of 2 m.

In each geometry, the charge positioned at (1,1) [x,y] corresponds to the first model of that arrangement. This progresses up the y axis to the fifth domain for each geometry corresponding to a charge at (1,7). The batch aims to replicate the detonation of an explosive along a road next to a building entrance being protected by various structural forms.

Figure 7.5 shows the branching tree that is produced when evaluating the batch with the BA. It shows that model 13 is the trunk model that will be simulated and used in the first stage of DeNNIS training. All other models will be processed using Viper if the performance metric is not met by the trained DeNN.

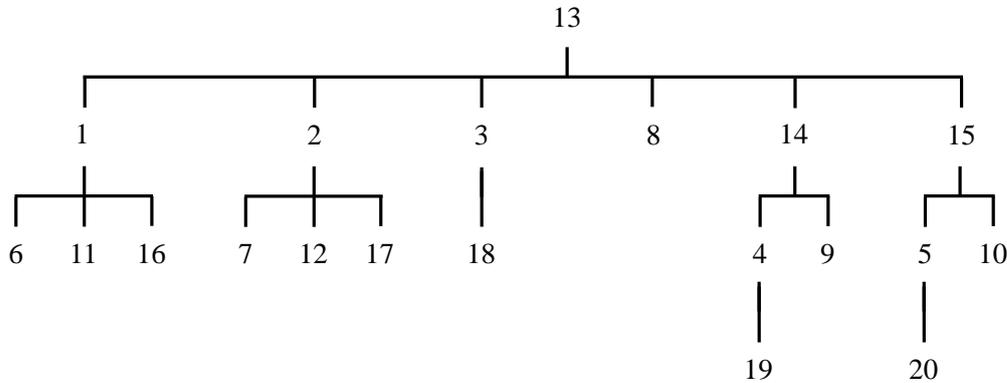


Figure 7.5: Branching Algorithm mapping trees for the batch of domains shown in Figure 7.4.

#### 7.4.1 Viper::Blast modelling

Modelling parameters for the domains that require CFD simulation are given by Table 7.2 in accordance to Section 3.2. In each domain, 1D–3D mapping is implemented when the 1D stage is one cell length away from the closest rigid boundary, this being at 1.499 m stand-off.

Table 7.2: Viper::Blast training model parameters.

Solving method	Ideal gas
Charge size (kg)	1
Charge composition	TNT
Charge density (kg/m <sup>3</sup> )	1600
Charge energy (J/kg)	$4.52 \times 10^6$
Mapping	1D–3D
1D cell size (m)	0.001
1D CFL	0.5
3D cell size (m)	0.02
3D CFL	0.4
Ambient temperature (K)	288
Ambient pressure (Pa)	101325
Termination time (s)	0.05

The DeNN will be trained in a comparable way to the previous chapter, with its predictions being formed on a 2D plane with 1.5 m elevation. It will therefore also be trained with data from a 3D modelling process where the domain height is set to 2 m and all boundaries are ambient apart from the rigid ground plane at  $z_{min}$ .

### 7.4.2 DeNN setup conditions

The DeNN architecture being adopted for this analysis is provided in Table 7.3. This matches the resulting network from Chapter 6 where hyperparameters were optimised for a multiple network application of the DeNN approach that utilises 16 directional inputs, the blast wave’s shortest travel distance to the point of interest (POI), and the superposition equation given by Equation 6.4.

Table 7.3: Direction-encoded Neural Network variables.

Variable	ANN-1 (unobstructed points)	ANN-2 (obstructed points)
Neuron structure	17-550-900-550-800-1	17-800-650-950-600-1
Learning rate	0.0170	0.0033
Dropout rate	0.0290	0.0139
Activation function	ReLU (linear at output)	
Loss function	Mean squared error (MSE)	
Optimiser	AdaGrad	
Batch size	100	
Regularisation	L2	
Weight initialiser	Glorot Normal	
Bias initialiser	Zeros	
Outputs	Peak overpressure at an elevation of 1.5 m (kPa) Blast wave’s shortest travel distance to prediction point, 16 directional inputs with 22.5° angular spacing that rotate so that direction 1 points towards the charge.	
Inputs		

## 7.5 Initialisation trials

### 7.5.1 Ultimate validation performance

To compare the initialisation methods shown in Table 7.1, each network is trained with a performance threshold of 0 kPa.ms MAE. This forces every model in the batch to be used in training the DeNN so that the ultimate performance of the tool can be determined.

Firstly, Figure 7.6 presents a comparison of the validation performance from pre-trained and untrained networks when an unstructured BA training framework is used. For ANN-1 the benefit of pre-training is negligible after the MAE reaches  $\sim 2$  kPa at around 7 incremental training steps. However, prior to this, pre-training provides a clear advantage over an untrained network, particularly after increment 1 when only the trunk model has been numerically simulated and used in training. Here, the average errors for the pre-trained model are consistently below 3 kPa, whereas the untrained networks provide errors up to 4 kPa.

The benefit of pre-training for ANN-2 is less prominent in all cases, particularly when the validation dataset is formed from 10% of the batch. Nevertheless, when one domain per geometry is chosen for the validation dataset, the pre-trained network consistently provides lower errors than an untrained model up to when all 16 training increments have been processed.

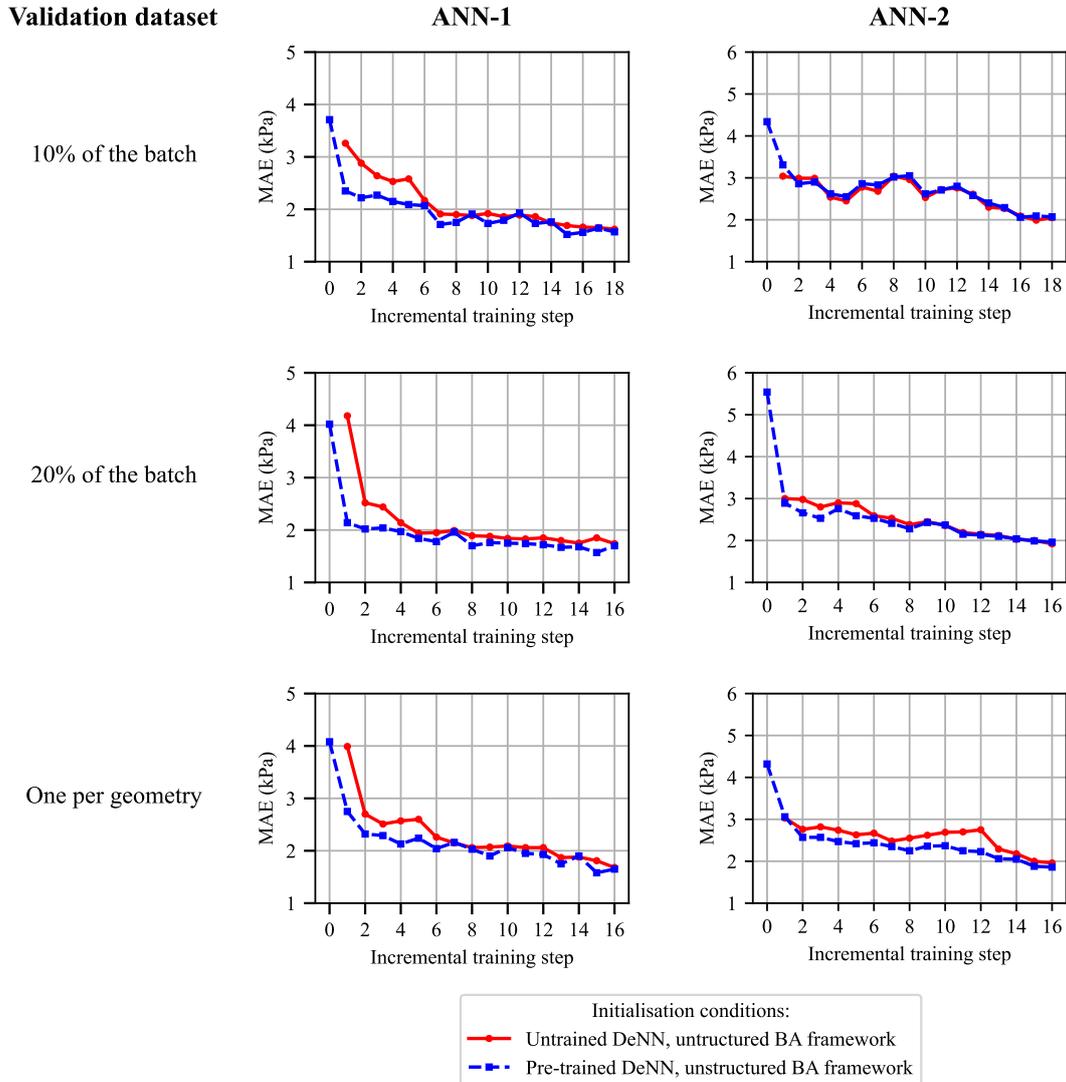


Figure 7.6: DeNNIS method comparison considering MAE of ANN-1 and ANN-2 when evaluating the validation datasets associated with each initialisation option that uses an unstructured Branching Algorithm training framework.

In all cases, the ultimate performance of the trained networks are consistent with an MAE  $\sim 1.8$  kPa and  $\sim 2$  kPa for ANN-1 and ANN-2 respectively. Despite this, use of a pre-trained network is likely to enable the performance to reach this threshold at an earlier training step. Figure 7.7 therefore compares pre-trained networks for all validation dataset options, with the structured BA training framework.

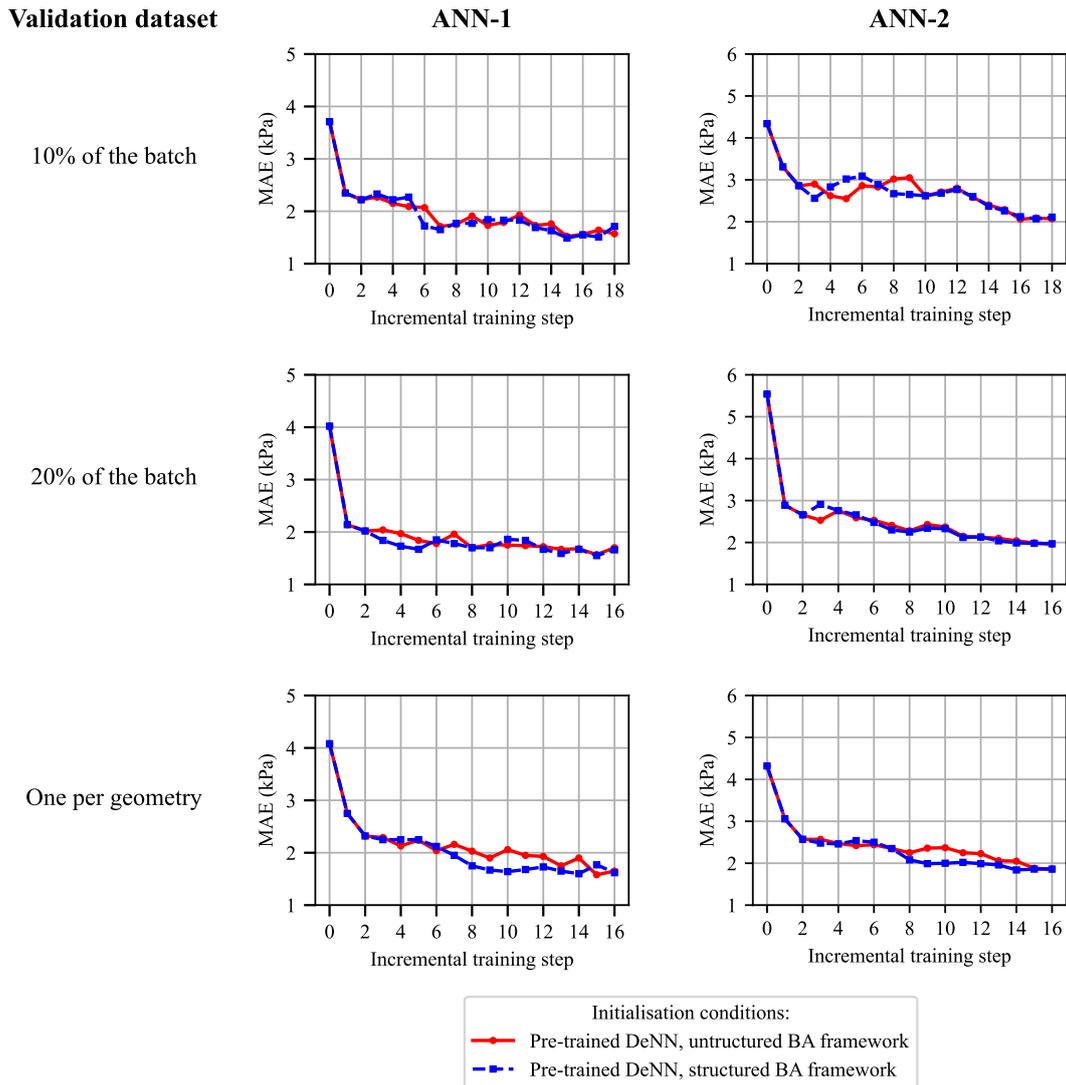


Figure 7.7: DeNNIS method comparison considering MAE of ANN-1 and ANN-2 when evaluating the validation datasets associated with each initialisation option that uses a pre-trained DeNN.

Once again, ultimate performance is consistent across all options. However, use of the structured training framework enables accelerated training at various increments for the 20% and one per geometry methods. For the latter, a notable improvement occurs at steps 7 and 8 for ANN-1 and ANN-2 to provide average errors that are very similar to the ultimate performance at a far earlier stage than when an unstructured approach is used. As mentioned previously, implementing every training step would remove the computation benefit that the DeNN’s novel approach achieves, and so a suitable performance threshold should be defined to limit the amount of training that takes place. Reaching near-optimal performance at an earlier increment is therefore beneficial to the resulting computation time, and accuracy, of the evaluated batch.

It should also be noted that the variations in performance across each option are characteristic of the training process that underpins how multi-layer perceptrons (MLPs) develop their accuracy. These variations will be specific to this training regime and batch of domains, meaning that conclusions formed from the comparison of unstructured or structure BA frameworks are only indicative of what the expected optimum approach would be for most cases.

### 7.5.2 Defined performance target

Ultimate performance can help to inform which analysis option may be optimal for the batch of domains given in Section 7.4, however, in practice the DeNNIS method will be implemented with a performance target that prevents the need for all domains to be used in training. Table 7.4 therefore provides the resulting batch statistics when a performance target of 2.5 kPa MAE is applied independently to ANN-1 and ANN-2.

A comparison of the overall batch accuracy (including CFD simulation output and DeNN outputs as appropriate) and computation time indicates that there is a trade off between these variables. Networks that are trained more effectively, meeting the required performance criteria sooner in the DeNNIS process, contribute to worse overall performance, but are part of a faster analysis process. This is because the CFD solver, Viper, is assumed to have 0 error and it is used less frequently in these cases.

Table 7.4: Average batch performance following DeNNIS training using a performance target of 2.5 kPa MAE for both ANN-1 and ANN-2.

Initial DeNN	BA order	Val. selection	No. DeNN uses	Comp. time (s)	MAE (kPa)	Avr. error (%)
Untrained	Unstructured	10%	4	33521	0.36	1.7
		20%	8	24845	0.81	3.8
		1 per geom.	9	22684	1.02	4.8
Pre-trained	Unstructured	10%	4	33350	0.37	1.8
		20%	9	22179	0.96	4.6
		1 per geom.	12	16668	1.29	6.6
Untrained	Structured	10%	4	33860	0.36	1.7
		20%	9	22452	0.95	4.7
		1 per geom.	8	24667	0.84	4.1
Pre-trained	Structured	10%	4	33537	0.36	1.7
		20%	10	20300	1.07	5.3
		1 per geom.	13	14466	1.45	7.1

As before, pre-trained networks provide a clear advantage over the associated untrained options for each validation approach, provided that computation time remains the key driver for conducting this kind of probabilistic analysis. Nevertheless, with batch errors below 10%, performance is also considered to be good enough for initial assessments of the domains in every instance.

For each validation selection approach, only one example of the random selection is presented. This is important to consider because performance is controlled by the domains that are used in training, and the growth of the training dataset is dependant on which models are removed and used for validation. Variations in performance could therefore occur if different validation models are selected, yet the trends seen here are likely to remain consistent. Subsequently, pre-trained networks and a structured BA training framework will be used throughout the remainder of this chapter due to the low computation times for each of validation selection option.

## 7.6 Prediction variations

Seeing as the DeNN is trained in a process that stops early if predictive performance on a validation dataset surpasses a user-defined value, variation in the accuracy of the model may vary significantly depending on which domains contributed to training and validation. To evaluate this hypothesis, a comparison of the 20% and one per geometry validation options is presented in this section, with the 10% approach being omitted due to overly large computation times shown in Table 7.4.

Use of one domain per geometry in the validation dataset aims to avert the issue of developing a bias tool by ensuring that performance is continually compared to an example of every geometry. As shown in Table 7.5, the consistency in the number of domains that contribute to training and validation should give the network the highest chance of generalising across the entire batch. In this example, the variation in prediction accuracy for each geometry is less significant when a using the one per geometry approach, whereas the 20% option favours domains using geometries A and B. Overall, the reported MAE's are lower than the targetted 2.5 kPa because, as shown in Figure 7.7, ANN-1 achieves this accuracy before ANN-2, but both networks continue to train as new data becomes available. The combined performance, shown in this section, is therefore including ANN-1 that performs with an MAE around 1.8 kPa.

Table 7.5: Comparison of DeNN performance variation (ANN-1 and ANN-2) in the batch considering each geometry for pre-trained networks developed with a performance limit of 2.5 kPa MAE and a structured BA training framework.

Val. selection	Geom.	MAE (kPa)	Avr. error (%)	Num. domains		
				Val.	Train	Pred.
20%	A	1.86	8.23	2	2	1
	B	1.69	7.79	1	1	3
	C	2.26	12.49	0	3	2
	D	2.50	12.34	1	0	4
One per geom.	A	2.11	8.65	1	1	3
	B	2.08	10.09	1	1	3
	C	2.24	12.83	1	1	3
	D	2.44	11.70	1	0	4

Both instances of the DeNNIS training regime result in networks that predict with around 12% error for the more challenging and complex geometries, C and D. However, for geometries A and B, it is likely that the extra training step used by the 20% option contributes to the better performance in every metric that is shown in Table 7.5. Despite this, seeing as the one per geometry approach has similar overall performance, a lack of predictive bias and reduced training time, this method is assumed to be optimal for evaluating this batch of domain using the DeNNIS procedure.

## 7.7 Analysis method comparison

To compare the DeNNIS approach with the other options that a user would have available to them when evaluating a batch of domains, Table 7.6 presents the average batch accuracy, computation time, and number of DeNN uses. The options include using a CFD solver for all domains, incorporating the Branching Algorithm to the CFD process, solely using the DeNN without additional training, or using the DeNN, CFD and the BA in collaboration with each other.

Table 7.6: Performance comparison of various approaches for analysing the batch of domains. DeNNIS method implemented using a pre-trained DeNN, structured training process and one validation domain selected per geometry.

Method	No. DeNN uses	Comp. time (s)	Avr. MAE (kPa)	Avr. error (%)
Viper	0	41824	0	0
BA – Viper	0	34195	0	0
DeNN	20	20	4.41	24.9
DeNNIS	13	14466	1.45	7.1

Clearly, if a process using only CFD were to be used, the BA should be incorporated to save 7629 s (18%) of computation time without reducing the accuracy. Conversely, only using the DeNN, trained with 25 randomised domains, saves practically all of the computation time, however, this results in average errors that are likely to be unacceptable around 25%. The DeNNIS method compromises between these two extremes with average errors around 7%, but a computation time saving of 27358 s (65%) when compared to using Viper for every simulation. This approach is therefore recommended as a means of rapidly evaluating batches of domains. Furthermore, use of an alternative performance target would vary the statistics reported throughout this chapter, thus potentially granting larger computation time savings. However, this would come at the expense of overall batch accuracy.

## 7.8 DeNN performance review

The previous sections have showed that additional, task specific training of the DeNN can result in a tool that is capable of evaluating complex batches of domains with predictive errors less than 10%. This was achieved using a version of the DeNN that was pre-trained using 25 randomised domains that were introduced in Chapter 6. Additional training should also improve the overall performance of the DeNN for models outside of the batch by developing the tools ability to generalise to unseen input patterns. Therefore, this

section compares the performance of the resulting model from the DeNNIS procedure, using a 2.5 kPa MAE performance target, to the original DeNN when predicting domains T1 and T2 from Section 6.3.3.

Table 7.7 provides each performance metric for each ANN that forms the DeNN, in addition to the overall performance for each domain. Notably, the reduction of ANN–2 error from 42.1% to 25.7%, and 57.6% to 36.8%, for T1 and T2 respectively, shows how the initial dataset formed from 25 randomised domains may have restricted the tool's ability to understand the complexities associated with wave coalescence that occurs behind obstacles. This is enforced by the significant improvement in the correlation between predictions and targets for ANN–2 and testing model 2, where an increase of 0.0607 is observed.

Table 7.7: Performance metrics for both testing domains, simulated using the DeNN and DeNNIS methods, compared to equivalent Viper models. Bold statistics indicate best performance for each testing model.

Testing model	Training models	ANN number	$R_t^2$	MAE (kPa)	Average Error (%)
1	Randomised ×25	1	<b>0.9961</b>	<b>4.18</b>	<b>5.4</b>
		2	0.8989	4.48	42.1
		Overall	0.9952	4.26	16.0
	Randomised ×25 + DeNNIS	1	0.9948	4.74	6.4
		2	<b>0.9174</b>	<b>3.64</b>	<b>25.7</b>
		Overall	<b>0.9940</b>	<b>4.42</b>	<b>12.0</b>
2	Randomised ×25	1	<b>0.9971</b>	<b>3.90</b>	<b>7.1</b>
		2	0.8070	6.83	57.6
		Overall	0.9938	4.86	23.6
	Randomised ×25 + DeNNIS	1	0.9963	4.32	7.3
		2	<b>0.8677</b>	<b>5.27</b>	<b>36.3</b>
		Overall	<b>0.9941</b>	<b>4.63</b>	<b>16.8</b>

In contrast, ANN–1 performs slightly worse following additional training, with average errors up to 1% higher. However, this change is considered to be negligible, and generally insignificant, in determining if the tool is suitable for use considering that performance of this network remains within the targeted 10% error threshold. It is possible that this network has become overfit to some training data, however this is unlikely considering the regularisation procedures and early stopping methods that have been employed throughout this thesis.

Overall, the DeNNIS model is shown to produce more accurate predictions for both unseen test domains, with average errors below 17% in both cases. The developed tool should therefore be used as the new ‘pre-trained’ model for future DeNNIS assessments so that even more additional training can continue to improve the predictive accuracy of the tool.

## 7.9 Summary

In summary, this chapter has introduced a new methodology that combines the benefits of two previously published approaches that enable the rapid assessment of various explosive events. It is shown that computational savings of up to 65% are possible, when compared to exclusively using CFD analysis, by incrementally training the Direction-encoded Neural Network (DeNN) in series with a simulation framework produced by the Branching algorithm (BA) for a batch of 20 models.

A comparison of various analysis options showed that use of a pre-trained DeNN, and a BA framework that was structured to ensure a diverse training dataset was developed after each iteration, decreased the amount of training steps that were required to reduce the predictive error of the tool to an acceptable level. Similarly, it was shown that although a validation dataset formed from a random selection of 20% of the domains provided comparable ANN performance, in order to maximise the tools ability to generalise to each unique geometry in the batch, use of one model per geometry was preferred.

Overall, predictive error for the batch of 20 domains reached 7.1% compared to 24.9% if no additional, task specific training was used. Subsequently, as a result of conducting additional training for the DeNN, performance of the tool improved when considering the testing domains that were originally evaluated in Chapter 6. In this case, reducing errors for T1 and T2 from 16% and 24% to 12% and 17% respectively. *Objective 5* of this study is therefore met as this methodology enables the DeNN to be used in probabilistic risk assessments of explosive events, exploring the influence of various uncertainties such as charge location and structural arrangements, with computation times that enable a large number of unique domains to be evaluated.

Preliminary results from the development of the DeNNIS procedure were presented at the 6<sup>th</sup> International Conference on Protective Structures, and provided in the conference proceedings in a paper titled ‘Towards the Development of Machine Learning Tools for Blast Load Prediction’.

# Chapter 8

## Summary and Conclusions

### 8.1 Summary

The need for Fast Running Engineering Models (FREMs) is supported by the requirement for large batches of numerical models to be analysed in probabilistic assessments that account for the uncertainty and variability of an explosion when assessing risk. As part of developing Machine learning tools for blast load prediction in obstructed environments, this thesis aimed to improve the computational efficiency of CFD modelling approaches for batch analysis in addition to creating a new, hybrid method of conducting probabilistic assessments using ML based models. This aim was met in conjunction with the following objectives that identified specific requirements to ensure the contents of this thesis would be novel:

1. Evaluate current literature to identify the limitations and opportunities associated to Machine Learning tools used in Fast Running Engineering Model development.
2. Establish a pipeline for generating experimentally validated data from numerical models.
3. Develop an approach for reducing the computation time required to generate training datasets featuring many models that share comparable setup characteristics.
4. Engineer an input set for Machine Learning models that allows for the tool to be used with movable obstacles, and changing domain sizes.
5. Produce a framework for probabilistically modelling variable explosive scenarios that incorporates the training phase of Machine Learning development with its use, removing the need for batches of tests to be evaluated exclusively with numerical models.

In Chapter 2, the background theory required to understand blast wave mechanics and Machine Learning was presented alongside a comprehensive review of current literature. This enabled the threat posed by the detonation of a high explosive to surrounding infrastructure and human life to be understood, whilst also providing a focussed view of the methods that are used to simulate a given explosive scenario. It was discussed that confined and obstructed environments lead to highly complex wave interactions that are often very computationally expensive to predict. Due to this increased complexity, many of the established FREMs are unsuitable for generating these predictions. Machine Learning tools have been used to accurately model a range of multi-parameter problems for various blast applications, showing promise for future use in this field, yet the lack of tool

versatility also restricts their use to a select number of pre-defined scenarios. The findings of this chapters therefore meet *Objective 1* of this study.

In order to work towards developing Machine Learning tools, Chapter 3 introduced and experimentally validated two numerical solvers, Viper::Blast and LS-DYNA, that are capable of modelling the propagation of a shock wave throughout domains featuring various obstacles. This ensured that *Objective 2* of this study would be met since the pipeline for data generation and use in developing FREMs was established with both solvers providing a means of extracting pressure traces from domains that feature varied charge locations, compositions, and obstacle geometries.

Using the validated solvers discussed above, Chapter 4 introduced a novel approach to conducting batch analyses whereby the repeated simulation steps can be removed through informed data mapping. This ‘Branching Algorithm’ is proved to work effectively for a series of 9 domains, saving  $\sim 50\%$  of the 2D computation time without any loss to the accuracy of the outputs from each domain. Following this, Chapter 5 expanded the methodology to 3D, improving the scope of its potential applications by adding more comprehensive consideration for wave tracking and simulation deviations. Here it was shown that  $\sim 20\%$  computation time could be saved for a batch of 20 containment structures, with 4 charge locations and 5 unique domain geometries. Therefore, since the computation time required to generate training datasets for ML tools can be reduced by using the Branching Algorithm, *Objective 3* of this study was met.

Next, with a focus on developing a model to predict blast loads in obstructed environments, Chapter 6 presented a Direction-encoded Framework for Machine Learning tools. By using a novel set of inputs, each point in a domain can be independently predicted with consideration of its proximity to obstacles and distance from the charge. This contrasts typical models where user-defined coordinates would be used to ultimately restrict the generalisation capabilities of the developed tool. It was shown that the key benefit of the framework, when applied as the Direction-encoded Neural Network (DeNN), is that predictions can be formed in domains of varied sizes with obstacles that do not have a fixed location, hence meeting *Objective 4* of this study.

To emphasise the combined benefit of each approach discussed here, Chapter 7 meets *Objective 5* by presenting an incremental training procedure whereby the DeNN is trained in series with data that is generated from a BA framework. It is shown that for a batch of 20 unique domains, additional training of the resulting DeNN from Chapter 6 leads to computation time savings of 65% when compared to exclusively running CFD analyses. Moreover, average predictions errors of 7.1% and 1.45 kPa provide a means of deriving useful quantitative conclusions about the batch of tests. The approach is therefore well suited for use in probabilistic assessments that aim to explore the effect of varied domain geometries and charge positions, but not the effect of breakable obstacles or the charge height. Furthermore, provided that the chosen numerical solver can be activated through the command line, implementation of this framework can controlled entirely through the associated Python scripts for the BA and DeNN.

The findings of this thesis should be used to inspire a new generation of Machine Learning based tools in blast protection engineering that are versatile, robust and validated for use in probabilistic assessments that consider the inherent variability and uncertainty of explosive events. This thesis has explored methods of improving each key aspect of ML

development, including data generation (The Branching Algorithm), model use (Direction-encoded Neural Network), and training & validation (Direction-encoded Neural Network In Series) to highlight how creative approaches to problems can provide meaningful results and improvements to efficiency that will ultimately help to rapidly assess the infrastructure that protects society.

## 8.2 Conclusions

The key conclusions of from this thesis are listed below:

- Obtaining blast load parameters in complex domains often requires computationally expensive numerical solvers, since experimental trials require expertise and test facilities that are not widely available, and established rapid analysis tools are not able to model the sophisticated physics associated with multiple wave reflections and interaction effects.
- Machine Learning tools provide a means of modelling highly complex, multi-parameter problems, however, most models that are developed for blast applications are bespoke to a very limited number of scenarios that ultimately prevent the tools from being useful outside of the developmental study.
- Viper::Blast and LS-DYNA can be used to accurately simulate shock wave propagation and the detonation of a high explosives provided that mesh size requirements are adhered to.
- The Branching Algorithm should be used when modelling a batch of domains that use the same charge composition and ambient conditions to reduce the required computation time by removing repeat simulation steps.
- A fully connected multi-layer perceptron model can be tuned to accurately provide predictions for peak overpressure in obstructed domains when utilising a novel set of inputs that relate the surroundings of a given point of interest to the network via a range of directional lasers.
- The Direction-encoded Neural Network can be used to rapidly analyse obstructed domains without the need for additional training or further CFD analysis. However, if it is being applied to a new batch of domains, incremental training as part of the DeNNIS procedure should be used to continue to improve the tool's predictive accuracy whilst also reducing the computation time required to evaluate all domains.

## 8.3 Future work opportunities and outlook

As noted previously, this thesis aims to inspire the next generation of Machine Learning tools in blast protection engineering by showing that creative adjustments to established methodologies can provide significant improvements to processes that have remained unchanged for a number of years. Through the process of developing the tools and analysis approaches included in this thesis, a number of opportunities for future work have arisen. For each tool, they are as follows:

### 8.3.1 The Branching Algorithm (BA)

- The inclusion of breakable obstacles and varied material properties when considering the deviation conditions of a domain. Allowing for more comprehensive probabilistic analyses to be conducted for scenarios that incorporate structural response.
- Direct integration of the BA with additional solvers to improve the usability of the method. Currently a Python script can be used to process a batch of models, but manual intervention is required to enact the required mapping procedures.
- Exploration of additional fields of study where the principles of the BA can be adapted and applied.

### 8.3.2 The Direction-encoded Framework and Neural Network

- Tailoring the inputs to the network so that non-rectangular objects and objects that are not orthogonal to the domain boundaries can be included in the obstructed environments. Required so that people, trees, lampposts, vehicles can be better represented when forming predictions throughout an urban environment.
- Expand the input pattern with additional directional lasers that project from the point of interest in 3D to enable predictions that consider the height of the domain and any obstacles, and the topology of the domain. Alternatively, generate 3D predictions by using multiple, stacked DeNNs that are trained for predictions at various heights above the ground.
- Adapt the principle of the framework and apply it to recurrent neural networks so that temporal predictions can be formed, providing a more complete assessment of the blast loads in obstructed environments whilst also enabling human injury assessments to include phase durations and impulse predictions that are required by some criteria.
- Considering the findings of Section 7.8, increase the size of, and diversify, the training dataset to determine if performance continues to improve with additional training.
- Test the principle of transfer learning, discussed by Pannell et al. (2023), to identify if the DeNN in its current form could be used to assist with predictions of alternative charge sizes or compositions in obstructed environments.
- Test alternative ML architectures, such as Transformer and Graph Neural Networks, to ascertain whether a particular ML tool can provide better prediction performance for the application discussed throughout this study when the Direction-encoded framework is applied.
- Sielicki et al. (2020) notes that most casualties from terrorist attacks in Western countries are caused by fragmentation and overpressure, rather than structural failure. The Human Injury Predictor, developed by Pope (2011), therefore accounts for both of these effects, and so future iterations of the DeNN should also aim to integrate fragmentation predictions. Alternatively, new ML tools should be developed to account for fragmentation, and the predictions should be combined with the DeNN to enable robust human injury assessments to be conducted rapidly.

- Validate predictions generated by the DeNN with experimental data to confirm that its training data, generated using Viper::Blast, and its learned representations of complex wave interaction effects are true to the physical problem.
- Explore uses of the DeNN that expand the scope of its application. This could include using its ability to produce rapid predictions as a means of forensically identifying the size and shape of an explosive charge relative to reported injuries and damage assessments throughout a complex domain.

### 8.3.3 Outlook

The findings of this thesis have contributed to the goal of creating versatile and accurate FREMs that can be applied to blast loading problems. Probabilistic analyses, that focus on varying charge location and structural arrangement, can now be conducted for obstructed environments with significant reductions in computational expense by using the DeNN and BA. This allows for an increased number of scenarios to be evaluated when determining the risk posed to a given target, providing a greater appreciation of the uncertainty associated to the event.

With consideration of these findings, the concept of ML for FREMs has been proved, creating a gap for the development of similar tools that can be applied to situations with more specific loading conditions, including underwater or buried charges, and structural response assessments. This process may require additional efforts to improve how experiential trials are conducted in a timely and cost effective manner, so that newly developed models are calibrated to provide an accurate representation of the physical processes being observed. In some instances, this may also highlight the need for a greater understanding of initiation, detonation and propagation processes for various explosive compounds at a thermochemical or molecular level that is yet to be considered at the time of writing this thesis.

Ultimately, the continual development in this field leads towards a future where a single FREM is able to provide a complete solution to any problem through accurate characterisation of an explosion, with probabilistic outputs associated to human injury, structural response and damage mitigation.

# Bibliography

- Alessandrini, V. (2015), *Shared Memory Application Programming: Concepts and Strategies in Multicore Application Programming*, 1st edn, Elsevier Science & Technology, Waltham, MA, USA.
- Alizadeh, M. J., Shahheydari, H., Kavianpour, M. R., Shamloo, H. & Barati, R. (2017), ‘Prediction of longitudinal dispersion coefficient in natural rivers using a cluster-based Bayesian network’, *Environmental Earth Sciences* **76**(2), 1–11.
- Alpaydin, E. (2016), *Machine Learning: The New AI*, MIT Press, Cambridge, Massachusetts.
- Alshammari, O. G., Isaac, O. S., Clarke, S. D. & Rigby, S. E. (2022), ‘Mitigation of blast loading through blast–obstacle interaction’, *International Journal of Protective Structures* .
- Alterman, D., Stewart, M. G. & Netherton, M. D. (2019), ‘Probabilistic assessment of airblast variability and fatality risk estimation for explosive blasts in confined building spaces’, *International Journal of Protective Structures* **10**(3), 306–329.
- Anderson, J. D. (2011), *Fundamentals of Aerodynamics*, 5th edn, McGraw-Hill, New York, NY.
- Angelides, S. C., Burgan, B. A., Kyprianou, C., Rigby, S. E. & Tyas, A. (2022), EMBlast: a software for rapidly and accurately predicting blast loads on structures, in ‘97th Conference of the International Committee for Industrial Construction (CICIND)’, Paphos, Cyprus, pp. 82–88.
- Anthistle, T., Fletcher, D. I. & Tyas, A. (2016), ‘Characterisation of blast loading in complex, confined geometries using quarter symmetry experimental methods’, *Shock Waves* **26**, 749–757.
- Aquelet, N. & Souli, M. (2008), 2D to 3D ALE mapping, in ‘10th International LS-DYNA Users Conference’, Dearborn, MI, USA, 8–10 June, pp. 23–34.
- Bakalis, G., Valipour, M., Bentahar, J., Kadem, L., Teng, H. & Dick, H. (2023), ‘Detonation cell size prediction based on artificial neural networks with chemical kinetics and thermodynamic parameters’, *Fuel Communications* **14**.
- Baker, W. E. (1973), *Explosions in Air*, University of Texas Press, Austin, Texas.
- Barr, A. D., Rigby, S. E., Collins, R., Speight, V. & Christen, T. (2020), ‘Predicting Crater Formation from Failure of Pressurized Water Mains through Analogy with Buried Explosive Events’, *Journal of Pipeline Systems Engineering and Practice* **11**.
- Barton, N. R., Bernier, J. V., Knap, J., Sunwoo, A. J., Cerreta, E. K. & Turner, T. J. (2011), ‘A call to arms for task parallelism in multi-scale materials modeling’, *International Journal for Numerical Methods in Engineering* **86**(6), 744–764.

- Bentley, J. L. (1975), ‘Multidimensional Binary Search Trees Used for Associative Searching’, *Communications of the ACM* **18**(9), 509–517.
- Bogosian, D., Yokota, M. & Rigby, S. (2016), TNT equivalence of C-4 and PE4: a review of traditional sources and recent data, *in* ‘Proceedings of the 24th Military Aspects of Blast and Shock’, MABS, Halifax, Nova Scotia, Canada, pp. 19–23.
- Bortolan Neto, L., Saleh, M., Pickerd, V., Yiannakopoulos, G., Mathys, Z. & Reid, W. (2020), ‘Rapid mechanical evaluation of quadrangular steel plates subjected to localised blast loadings’, *International Journal of Impact Engineering* **137**.
- Boser, B. E., Guyon, I. M. & Vapnik, V. N. (1992), ‘Training algorithm for optimal margin classifiers’, *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory* pp. 144–152.
- Breivik, Ø. & Allen, A. A. (2008), ‘An operational search and rescue model for the Norwegian Sea and the North Sea’, *Journal of Marine Systems* **69**(1-2), 99–113.
- Bruneau, C. H. & Khadra, K. (2016), ‘Highly parallel computing of a multigrid solver for 3D Navier–Stokes equations’, *Journal of Computational Science* **17**, 35–46.
- Burges, C. J. C. (1998), ‘A Tutorial on Support Vector Machines for Pattern Recognition’, *Data Mining and Knowledge Discovery* **2**, 121–167.
- Cha, Y. J., Choi, W., Suh, G., Mahmoudkhani, S. & Büyüköztürk, O. (2018), ‘Autonomous Structural Visual Inspection Using Region-Based Deep Learning for Detecting Multiple Damage Types’, *Computer-Aided Civil and Infrastructure Engineering* **33**(9), 731–747.
- Chen, L., Hassan, H., Tallman, T. N., Huang, S. S. & Smyl, D. (2022), ‘Predicting strain and stress fields in self-sensing nanocomposites using deep learned electrical tomography’, *Smart Materials and Structures* **31**(4).
- Chen, W., Warmink, J. J., van Gent, M. R. A. & Hulscher, S. J. M. H. (2021), ‘Numerical modelling of wave overtopping at dikes using OpenFOAM®’, *Coastal Engineering* .
- Chencho, Li, J., Hao, H., Wang, R. & Li, L. (2021), ‘Development and application of random forest technique for element level structural damage quantification’, *Structural Control and Health Monitoring* **28**(3).
- Chiu, M.-C., Yeh, L.-J. & Lin, Y. C. (2009), ‘The design and application of a robotic vacuum cleaner’, *Journal of Information and Optimization Sciences* **30**(1), 39–62.
- Christensen, P., Fong, J., Shade, J., Wooten, W., Schubert, B., Kensler, A., Friedman, S., Kilpatrick, C., Ramshaw, C., Bannister, M., Rayner, B., Brouillat, J. & Liani, M. (2018), ‘RenderMan: An advanced path-tracing architecture for movie rendering’, *ACM Transactions on Graphics* **37**(3).
- Christensen, P. H., Fong, J., Laur, D. M. & Batali, D. (2006), Ray tracing for the movie ‘cars’, *in* ‘2006 IEEE Symposium on Interactive Ray Tracing’.
- Codina, R. & Ambrosini, D. (2018), ‘Full-scale testing of leakage of blast waves inside a partially vented room exposed to external air blast loading’, *Shock Waves* **28**(2), 227–241.

- Codina, R., Ambrosini, D. & De Borbón, F. (2013), 'Numerical study of confined explosions in urban environments', *International Journal of Protective Structures* **4**(4), 591–617.
- Coppini, G., Jansen, E., Turrisi, G., Creti, S., Shchekinova, E. Y., Pinardi, N., Lecci, R., Carluccio, I., Kumkar, Y. V., D'Anca, A., Mannarini, G., Martinelli, S., Marra, P., Capodiferro, T. & Gismondi, T. (2016), 'A new search-and-rescue service in the Mediterranean Sea: A demonstration of the operational capability and an evaluation of its performance using real case scenarios', *Natural Hazards and Earth System Sciences* **16**(12), 2713–2727.
- Cormie, D., Mays, G. & Smith, P. D. (2009), *Blast effects on buildings, Second edition*, ICE Publishing, London.
- Cortes, C. & Vapnik, V. (1995), 'Support-Vector Networks', *Machine Learning* **20**, 273–297.
- Cranz, C. (1926), *Lehr der Ballistik*, Verlag von Julius Springer, Berlin, Germany.
- Cunha, B., Zine, A.-M., Ichchou, M., Droz, C. & Foulard, S. (2022), 'On Machine Learning-Driven Surrogates for Sound Transmission Loss Simulations', *Journal of Sound and Vibration* .
- Dennis, A. A., Pannell, J. J., Smyl, D. J. & Rigby, S. E. (2021), 'Prediction of blast loading in an internal environment using artificial neural networks', *International Journal of Protective Structures* **12**(3), 287–314.
- Dennis, A. A. & Rigby, S. E. (2023a), Prediction of Blast Loads using Machine Learning Approaches, in 'SECED 2023 Conference, Earthquake Engineering & Dynamics for a Sustainable Future', 14–15th September, Cambridge, UK.
- Dennis, A. A. & Rigby, S. E. (2023b), 'The Direction-encoded Neural Network: A machine learning approach to rapidly predict blast loading in obstructed environments', *International Journal of Protective Structures* .
- Dennis, A. A., Smyl, D. J., Stirling, C. G. & Rigby, S. E. (2023), 'A branching algorithm to reduce computational time of batch models: Application for blast analyses', *International Journal of Protective Structures* **14**(2), 135–167.
- Dennis, A. A., Stirling, C. G. & Rigby, S. E. (2023), Towards the development of Machine Learning tools for blast load prediction, in '6th International Conference on Protective Structures (ICPS6)', 14–17th May, Auburn, AL.
- Denny, J., Langdon, G., Rigby, S., Dickinson, A. & Batchelor, J. (2022), 'A numerical investigation of blast-structure interaction effects on primary blast injury risk and the suitability of existing injury prediction methods', *International Journal of Protective Structures* .
- Denny, J. W., Dickinson, A. S. & Langdon, G. S. (2021a), 'Defining blast loading zones of relevance' for primary blast injury research: A consensus of injury criteria for idealised explosive scenarios', *Medical Engineering & Physics* **93**, 83–92.

- Denny, J. W., Dickinson, A. S. & Langdon, G. S. (2021*b*), ‘Guidelines to inform the generation of clinically relevant and realistic blast loading conditions for primary blast injury research’, *BMJ Military Health*.
- Devlin, J., Chang, M.-W., Lee, K., Google, K. T. & Language, A. I. (2019), BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, in ‘Naacl-Hlt 2019’, Association for Computational Linguistics, Minneapolis, Minnesota, pp. 4171–4186.
- Dong, G. & Liu, H. (2018), *Feature Engineering for Machine Learning and Data Analytics*, 1st edn, CRC Press, Boca Raton, Florida.
- Duchi, J., Hazan, E. & Singer, Y. (2011), ‘Adaptive subgradient methods for online learning and stochastic optimization’, *Journal of Machine Learning Research* **12**, 2121–2159.
- Environmental Systems Research Institute (2020), ‘ArcMap, 2020’.
- Feitosa, D., Ampatzoglou, A., Gkortzis, A., Bibi, S. & Chatzigeorgiou, A. (2020), ‘CODE reuse in practice: Benefiting or harming technical debt’, *Journal of Systems and Software* **167**.
- Flah, M., Nunez, I., Ben Chaabene, W. & Nehdi, M. L. (2021), ‘Machine Learning Algorithms in Civil Structural Health Monitoring: A Systematic Review’, *Archives of Computational Methods in Engineering* **28**(4), 2621–2643.
- Flood, I. & Kartam, N. (1994), ‘Neural Networks in Civil Engineering. I: Principles and Understanding’, *Journal of Computing in Civil Engineering* **8**(2), 131–148.
- Fouchier, C., Laboureur, D., Youinou, L., Lapebie, E. & Buchlin, J. M. (2017), ‘Experimental investigation of blast wave propagation in an urban environment’, *Journal of Loss Prevention in the Process Industries* **49**, 248–265.
- Frank, S., Frank, R. & Needham, C. (2008), Fast running model for urban airblast using engineering principles, in ‘20th Proceedings of Military Aspects of Blast and Shock’, Oslo.
- Fraunhofer EMI (2018), *APOLLO Blastsimulator manual, version: 2018.2*, Fraunhofer Institute for High-Speed Dynamics, Ernst-Mach-Institut, Freiburg, Germany.
- Furlani, T. R. & Lordi, J. A. (1990), ‘Implementation of the direct simulation monte carlo method for an exhaust plume flowfield in a parallel computing environment’, *Computers & Fluids* **18**(2), 217–227.
- Gajewski, T. & Sielicki, P. W. (2020), ‘Experimental study of blast loading behind a building corner’, *Shock Waves* **30**(4), 385–394.
- Gallet, A., Liew, A., Hajirasouliha, I. & Smyl, D. (2023), ‘Influence zones of continuous beam systems’. *Submitted for possible publication in Computers & Structures*.
- Gan, K. L., Brewer, T. R., Pope, D. J. & Rigby, S. E. (2022), ‘Probabilistic analysis of blast–obstacle interaction in a crowded internal environment’, *Probabilistic Engineering Mechanics* **68**.

- Ganaie, M. A., Hu, M., Malik, A. K., Tanveer, M. & Suganthan, P. N. (2022), ‘Ensemble deep learning: A review’, *Engineering Applications of Artificial Intelligence* **115**(May), 105151.
- Gault, K., Sochet, I., Hakenholz, L. & Collignon, A. (2020), ‘Influence of the explosion center on shock wave propagation in a confined room’, *Shock Waves* **30**(5), 473–481.
- Gautier, A., Sochet, I., Lapebie, E. & Boubrit, A. (2020), ‘Shock wave propagation in an obstructed area’, *WIT Transactions on the Built Environment* **198**, 15–27.
- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014), Generative Adversarial Nets, in ‘Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2’, Montreal, Canada, pp. 2672–2680.
- Grisaro, H. Y., Edri, I. E. & Rigby, S. E. (2021), ‘TNT equivalency analysis of specific impulse distribution from close-in detonations’, *International Journal of Protective Structures* **12**(3), 315–330.
- Guo, M. & Hesthaven, J. S. (2019), ‘Data-driven reduced order modeling for time-dependent problems’, *Computer Methods in Applied Mechanics and Engineering* **345**, 75–99.
- Hadgu, A. T., Nigam, A. & Diaz-Aviles, E. (2015), Large-scale learning with AdaGrad on Spark, in ‘2015 IEEE International Conference on Big Data’, Vol. 2, IEEE, pp. 2828–2830.
- Halswijk, W. (2015), BeamBlast : Blast Path-finding algorithms, in ‘16th ISIEMS International Symposium for the Interaction of the Effects of Munitions with Structures’.
- Hanson, W. A. (2020), ‘The CORAL supercomputer systems’, *IBM Journal of Research and Development* **64**(3-4).
- Hijazi, S., Stabile, G., Mola, A. & Rozza, G. (2020), ‘Data-driven POD-Galerkin reduced order model for turbulent flows’, *Journal of Computational Physics* **416**, 109513.
- Hikida, S. & Needham, C. (1981), *Low Altitude Multiple Burst (LAMB) Model: Volume I - Shock Description*.
- Ho, T. K. (1995), ‘Random Decision Forests’, *Proceedings of 3rd International Conference on Document Analysis and Recognition* **1**, 278–282.
- Hopkinson, B. (1915), ‘British Ordnance Board Minutes, 13565’.
- Hornung, A., Phillips, M., Gil Jones, E., Bennewitz, M., Likhachev, M. & Chitta, S. (2012), ‘Navigation in three-dimensional cluttered environments for mobile manipulation’, *Proceedings - IEEE International Conference on Robotics and Automation* pp. 423–429.
- Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. & Burgard, W. (2013), ‘OctoMap: an efficient probabilistic 3D mapping framework based on octrees’, *Autonomous Robots* **34**, 189–206.

- Hua, J., Tembe, W. D. & Dougherty, E. R. (2009), ‘Performance of feature-selection methods in the classification of high-dimension data’, *Pattern Recognition* **42**(3), 409–424.
- Huang, H. N., Chen, H. M., Lin, W. W., Huang, C. J., Chen, Y. C., Wang, Y. H. & Yang, C. T. (2023), ‘Employing feature engineering strategies to improve the performance of machine learning algorithms on echocardiogram dataset’, *Digital Health* **9**.
- Hunt, A. & Thomas, D. (1999), *The Pragmatic Programmer: From Journeyman to Master*, Addison-Wesley.
- Hyde, D. W. (1991), ‘Conventional Weapons Effect Program (ConWep)’.
- Isaac, O. S., Alshammari, O. G., Clarke, S. D. & Rigby, S. E. (2022b), ‘Experimental investigation of blast mitigation of pre-fractal obstacles’, *International Journal of Protective Structures* .
- Isaac, O. S., Alshammari, O. G., Pickering, E. G., Clarke, S. D. & Rigby, S. E. (2022a), ‘Blast wave interaction with structures – An overview’, *International Journal of Protective Structures* .
- Jachym, M., Lavernhe, S., Euzenat, C. & Tournier, C. (2019), ‘Effective NC machining simulation with OptiX ray tracing engine’, *Visual Computer* **35**(2), 281–288.
- Jing, Y., Zhang, L., Hao, W. & Huang, L. (2022), ‘Numerical study of a CNN-based model for regional wave prediction’, *Ocean Engineering* **255**.
- Kang, M. A. & Park, C. H. (2023), ‘Prediction of Peak Pressure by Blast Wave Propagation Between Buildings Using a Conditional 3D Convolutional Neural Network’, *IEEE Access* .
- Kang, M. C., Kim, K. S., Noh, D. K., Han, J. W. & Ko, S. J. (2014), ‘A robust obstacle detection method for robotic vacuum cleaners’, *IEEE Transactions on Consumer Electronics* **60**(4), 587–595.
- Kingery, C. N. & Bulmash, G. (1984), *Airblast Parameters from TNT Spherical Air Burst and Hemispherical Surface Burst*, ARBRL-TR-02555, US Army Ballistic Research Laboratory, Aberdeen Proving Ground, MD, USA.
- Kirchner, M. R., Kirchner, S. R., Dennis, A. A. & Rigby, S. E. (2023), ‘Non-parametric characterization of blast loads’, *International Journal of Protective Structures* .
- Koubaa, A., Boulila, W., Ghouti, L., Alzahem, A. & Latif, S. (2023), ‘Exploring ChatGPT Capabilities and Limitations: A Survey’, *IEEE Access* **11**(September), 118698–118721.
- Kudela, J. & Matousek, R. (2022), ‘Recent advances and applications of surrogate models for finite element method computations: a review’, *Soft Computing* **26**(24), 13709–13733.
- Kumar, R. L., Khan, F., Din, S., Band, S. S., Mosavi, A. & Ibeke, E. (2021), ‘Recurrent Neural Network and Reinforcement Learning Model for COVID-19 Prediction’, *Frontiers in Public Health* **9**.

- Lapoujade, V., Van Dorsselaer, N., Kevorkian, S. & Cheval, K. (2010), ‘A Study of Mapping Technique for Air Blast Modeling’, *11th International LS-DYNA users Conference: Blast/Impact* pp. 23–32.
- Larcher, M., Casadei, F., Solomos, G. & Gebbeken, N. (2011), ‘Can venting areas mitigate the risk due to air blast inside railway carriages?’, *International Journal of Protective Structures* **2**(2), 221–230.
- Larcher, M., Forsberg, R., Björnstig, U., Holgersson, A. & Solomos, G. (2016), ‘Effectiveness of finite-element modelling of damage and injuries for explosions inside trains’, *Journal of Transportation Safety and Security* **8**(1), 83–100.
- Lee, E. L., Hornig, H. C. & Kury, J. W. (1968), *Adiabatic Expansion Of High Explosive Detonation Products*. TID 4500-UCRL 50422., Technical report, University of California, CA, USA.
- Lee, S., Vlahopoulos, N. & Mohammad, S. (2020), ‘Development of a Regression Model for Blast Pressure Prediction in Urban Street Configurations’, *16th International LS-DYNA User Conference* .
- Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. (2018), ‘Hyperband: A novel bandit-based approach to hyperparameter optimization’, *Journal of Machine Learning Research* **18**.
- Li, Q., Wang, Y., Li, L., Hao, H., Wang, R. & Li, J. (2023), ‘Prediction of BLEVE loads on structures using machine learning and CFD’, *Process Safety and Environmental Protection* **171**(February), 914–925.
- Li, Q., Wang, Y., Shao, Y., Li, L. & Hao, H. (2023), ‘A comparative study on the most effective machine learning model for blast loading prediction: From GBDT to Transformer’, *Engineering Structures* **276**(December 2022), 115310.
- Livermore Software Technology Company (2015), ‘LS-DYNA Keyword User’s Manual Volume I’.
- Long, K., Gao, Z., Yuan, Q., Xiang, W. & Hao, W. (2018), ‘Safety evaluation for roadside crashes by vehicle – object collision simulation’, *Advances in Mechanical Engineering* **10**.
- Lu, Y., Shen, J., Wang, C., Lu, H. & Xin, J. (2020), ‘Studying on the design and simulation of collision protection system between vehicle and pedestrian’, *International Journal of Distributed Sensor Networks* **16**.
- Marks, N. A., Stewart, M. G., Netherton, M. D. & Stirling, C. G. (2021), ‘Airblast variability and fatality risks from a VBIED in a complex urban environment’, *Reliability Engineering and System Safety* **209**.
- MathWorks Inc (2021), ‘MATLAB R2021a’.
- Meagher, D. (1982), ‘Geometric modeling using octree encoding’, *Computer Graphics and Image Processing* **19**(2), 129–147.

- Mehreganian, N., Louca, L. A., Langdon, G. S., Curry, R. J. & Abdul-Karim, N. (2018), ‘The response of mild steel and armour steel plates to localised air-blast loading-comparison of numerical modelling techniques’, *International Journal of Impact Engineering* **115**, 81–93.
- Mende, H., Frye, M., Vogel, P. A., Kiroriwal, S., Schmitt, R. H. & Bergs, T. (2023), ‘On the importance of domain expertise in feature engineering for predictive product quality in production’, *Procedia CIRP* **118**, 1096–1101.
- Meng, Q. & Berzins, M. (2013), ‘Scalable large-scale fluid–structure interaction solvers in the Uintah framework via hybrid task-based parallelism algorithms’, *Concurrency Computation Practice and Experience* **26**(7), 1388–1407.
- Menon, S., Mooney, K. G., Stapf, K. G. & Schmidt, D. P. (2015), ‘Parallel adaptive simplicial re-meshing for deforming domain CFD computations’, *Journal of Computational Physics* **298**, 62–78.
- Mera-Gaona, M., Neumann, U., Vargas-Canas, R. & López, D. M. (2021), ‘Evaluating the impact of multivariate imputation by MICE in feature selection’, *PLoS ONE* **16**(7 July), 1–28.
- Mohri, M., Rostamizadeh, A. & Talwalkar, A. (2018), *Foundations of Machine Learning*, 2nd edn, MIT Press.
- Mojtabaei, S. M., Becque, J., Hajirasouliha, I. & Khandan, R. (2023), ‘Predicting the buckling behaviour of thin-walled structural elements using machine learning methods’, *Thin-Walled Structures* **184**(January), 110518.
- Morgan, J. N. & Sonquist, J. A. (1963), ‘Problems in the Analysis of Survey Data, and a Proposal’, *Journal of the American Statistical Association* **58**(302).
- Murmu, S., Maheshwari, P. & Verma, H. K. (2018), ‘Empirical and probabilistic analysis of blast-induced ground vibrations’, *International Journal of Rock Mechanics and Mining Sciences* **103**, 267–274.
- Netherton, M. D. & Stewart, M. G. (2010), ‘Blast load variability and accuracy of blast load prediction models’, *International Journal of Protective Structures* **1**(4), 543–570.
- Netherton, M. D. & Stewart, M. G. (2016), ‘Risk-based blast-load modelling: Techniques, models and benefits’, *International Journal of Protective Structures* **7**(3), 430–451.
- Neto, L. B., Saleh, M., Pickerd, V., Yiannakopoulos, G., Mathys, Z. & Reid, W. (2017), ‘Rapid vulnerability assessment of naval structures subjected to localised blast’, *RINA, Royal Institution of Naval Architects - PACIFIC 2017 International Maritime Conference* (December 2018).
- O’Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L. et al. (2019), ‘Kerastuner’.  
**URL:** <https://github.com/keras-team/keras-tuner>
- Ortega-Arranz, H., Llanos, D. R. & Gonzalez-Escribano, A. (2014), *The Shortest-Path Problem: Analysis and Comparison of Methods*, Morgan and Claypool, San Rafael, California.

- Pannell, J. J., Panoutsos, G., Cooke, S. B., Pope, D. J. & Rigby, S. E. (2021), 'Predicting specific impulse distributions for spherical explosives in the extreme near-field using a Gaussian function', *International Journal of Protective Structures* **12**(4), 437–459.
- Pannell, J. J., Rigby, S. E. & Panoutsos, G. (2022), 'Physics-informed regularisation procedure in neural networks: An application in blast protection engineering', *International Journal of Protective Structures* **13**(3), 555–578.
- Pannell, J. J., Rigby, S. E. & Panoutsos, G. (2023), 'Application of transfer learning for the prediction of blast impulse', *International Journal of Protective Structures* **14**(2), 242–262.
- Park, Y. M. & Kwon, O. J. (2005), 'A parallel unstructured dynamic mesh adaptation algorithm for 3-D unsteady flows', *International Journal for Numerical Methods in Fluids* **48**(6), 671–690.
- Peery, J. S. & Carroll, D. E. (2000), 'Multi-Material ALE methods in unstructured grids', *Computer Methods in Applied Mechanics and Engineering* **187**(3-4), 591–619.
- Pope, D. J. (2011), 'The development of a quick-running prediction tool for the assessment of human injury owing to terrorist attack within crowded metropolitan environments', *Philosophical Transactions of the Royal Society B: Biological Sciences* **366**(1562), 127–143.
- Pringle, C., Bailey, M., Bukhari, S., El-Sayed, A., Hughes, S., Josan, V., Ramirez, R. & Kamaly-Asl, I. (2020), 'Manchester Arena Attack: management of paediatric penetrating brain injuries', *British Journal of Neurosurgery* **35**.
- Qi, S., Zhi, X., Fan, F. & Flay, R. G. (2020), 'Probabilistic blast load model for domes under external surface burst explosions', *Structural Safety* **87**.
- Ray, P. P. (2023), 'ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope', *Internet of Things and Cyber-Physical Systems* **3**(April), 121–154.
- Rebelo, H. B. & Cismasiu, C. (2021), 'Robustness assessment of a deterministically designed sacrificial cladding for structural protection', *Engineering Structures* **240**.
- Remennikov, A., Gan, E. C. J., Tan, S. S., Bharath, B. & Carey, D. (2022), Methodology for predicting explosion risk around underground coal mine openings towards developing exclusion zones, in 'Proceedings of the 2022 Resource Operators Conference', pp. 126–138.
- Remennikov, A. M. & Mendis, P. A. (2006), 'Prediction of airblast loads in complex environments using artificial neural networks', *WIT Transactions on the Built Environment* **87**, 269–278.
- Remennikov, A. M. & Rose, T. A. (2007), 'Predicting the effectiveness of blast wall barriers using neural networks', *International Journal of Impact Engineering* **34**(12), 1907–1923.
- Remennikov, A. & Rose, T. (2005), 'Modelling blast loads on buildings in complex city geometries', *Computers & Structures* **83**(27), 2197–2205.

- Rigby, S. E., Akintaro, O. I., Fuller, B. J., Tyas, A., Curry, R. J., Langdon, G. S. & Pope, D. J. (2019), 'Predicting the response of plates subjected to near-field explosions using an energy equivalent impulse', *International Journal of Impact Engineering* **128**, 24–36.
- Rigby, S. E. & Gitterman, Y. (2016), Secondary shock delay measurements from explosive trials, in 'Proceedings of the 24th Military Aspects of Blast and Shock', Halifax, Nova Scotia, Canada.
- Rigby, S. E., Lodge, T. J., Alotaibi, S., Barr, A. D., Clarke, S. D., Langdon, G. S. & Tyas, A. (2020a), 'Preliminary yield estimation of the 2020 Beirut explosion using video footage from social media', *Shock Waves* **30**(6), 671–675.
- Rigby, S. E. & Sielicki, P. W. (2014), 'An investigation of TNT equivalence of hemispherical PE4 charges', *Engineering Transactions* **62**(4), 423–435.
- Rigby, S. E., Tyas, A., Bennett, T., Warren, J. A. & Fay, S. (2013), 'Clearing effects on plates subjected to blast loads', *Proceedings of the Institution of Civil Engineers: Engineering and Computational Mechanics* **166**(3), 140–148.
- Rigby, S. E., Tyas, A., Fay, S. D., Clarke, S. D. & Warren, J. A. (2014), Validation of Semi-Empirical Blast Pressure Predictions for Far Field Explosions - Is There Inherent Variability in Blast Wave Parameters?, in '6th International Conference on Protection of Structures Against Hazards', Tianjin, China.
- Rigby, S., Fay, S., Tyas, A., Warren, J. & Clarke, S. (2015), 'Angle of incidence effects on far-field positive and negative phase blast parameters', *International Journal of Protective Structures* **6**(1), 23–42.
- Rigby, S., Fuller, B. & Tyas, A. (2018), Validation of Near-Field Blast Loading in LS-DYNA, in '5th International Conference on Protective Structures'.
- Rigby, S., Knighton, R., Clarke, S. D. & Tyas, A. (2020b), 'Reflected Near-field Blast Pressure Measurements Using High Speed Video', *Experimental Mechanics* **60**(7), 875–888.
- Rose, T. A. (2001), An Approach to the Evaluation of Blast Loads on Finites and Semi-Infinite Structures, Phd thesis, Cranfield University.
- Rosenblatt, F. (1958), 'The perceptron: A probabilistic model for information storage and organization in the brain', *Psychological Review* **65**(6), 386–408.
- Roybal, L. G., Jeffers, R. F., McGillivray, K. E., Paul, T. D. & Jacobson, R. (2009), 'Modeling and simulating blast effects on electric substations', *2009 IEEE Conference on Technologies for Homeland Security, HST 2009* pp. 351–357.
- Russell, S. J. & Norvig, P. (2010), *Artificial Intelligence: A Modern Approach*, Vol. 4, 3rd edn, Prentice hall.
- Sauvan, P. E., Sochet, I. & Trélat, S. (2012), 'Analysis of reflected blast wave pressure profiles in a confined room', *Shock Waves* **22**(3), 253–264.
- Schwer, L. & Rigby, S. (2018), Secondary and Height of Burst Shock Reflections : Application of Afterburning, in 'Proceedings of the 25th Military Aspects of Blast and Shock (MABS25)', Military Aspects of Blast and Shock, The Hauge, Netherlands.

- Seisson, G., Lacaze, T. & Rouquand, A. (2020), ‘Uncertainty estimation of external blast effects using the monte-carlo method’, *WIT Transactions on the Built Environment* **198**, 81–92.
- Shehu, E., Lomazzi, L., Giglio, M. & Manes, A. (2023), ‘Computational modeling of confined blast waves with focus on interaction with structures’, *IOP Conference Series: Materials Science and Engineering* **1275**.
- Sielicki, P., Stewart, M., Gajewski, T., Malendowski, M., Peksa, P., Al-Rifaie, H., Studziński, W. & Sumelka, R. (2020), ‘Field Test and Probabilistic Analysis of Irregular Steel Debris Casualty Risks from a Person-Borne Improvised Explosive Device’, *Defence Technology* **17**(6), 1852–1863.
- Singh, D. & Singh, B. (2020), ‘Investigating the impact of data normalization on classification performance’, *Applied Soft Computing* **97**, 105524.
- Sola, J. & Sevilla, J. (1997), ‘Importance of input data normalization for the application of neural networks to complex industrial problems’, *IEEE Transactions on Nuclear Science* **44**(3), 1464–1468.
- Souli, M., Ouahsine, A. & Lewin, L. (2000), ‘ALE formulation for fluid–structure interaction problems’, *Computer Methods in Applied Mechanics and Engineering* **190**(5–7), 659–675.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), ‘Dropout: A Simple Way to Prevent Neural Networks from Overfitting’, *Journal of Machine Learning Research* **15**, 1929–1958.
- Stewart, M. G. & Netherton, M. D. (2015), ‘Reliability-based design load factors for explosive blast loading’, *Journal of Performance of Constructed Facilities* **29**(5).
- Stewart, M. G., Netherton, M. D. & Rosowsky, D. V. (2006), ‘Terrorism risks and blast damage to built infrastructure’, *Natural Hazards Review* **7**(3), 114–122.
- Stirling, C. (2023), ‘Viper::Blast’.
- Tapeh, A. T. G. & Naser, M. Z. (2023), ‘Artificial Intelligence, Machine Learning, and Deep Learning in Structural Engineering: A Scientometrics Review of Trends and Best Practices’, *Archives of Computational Methods in Engineering* **30**(1), 115–159.
- Thai, H. T. (2022), ‘Machine learning for structural engineering: A state-of-the-art review’, *Structures* **38**(January), 448–491.
- Tyas, A., Reay, J. J., Fay, S. D., Clarke, S. D., Rigby, S. E., Warren, J. A. & Pope, D. J. (2016), ‘Experimental studies of the effect of rapid afterburn on shock development of near-field explosions’, *International Journal of Protective Structures* **7**(3), 452–465.
- Ulseth, J., Zhu, Z., Sun, Y. & Pang, S. (2019), ‘Accelerated X-Ray Diffraction (Tensor) Tomography Simulation Using OptiX GPU Ray-Tracing Engine’, *IEEE Transactions on Nuclear Science* **66**(12), 2347–2354.
- US Army Material Command (1974), *Engineering Design Handbook. Explosions in Air. Part One*, Alexandria, VA, USA.

- US DoD (2008), UFC-3-340-02: Structures to resist the effect of accidental explosions, Technical report, Washington, DC.
- Valsamos, G., Larcher, M. & Casadei, F. (2021), ‘Beirut explosion 2020: A case study for a large-scale urban blast simulation’, *Safety Science* **137**.
- van Rossum, G. (1995), Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam.
- Varnavsky, A. N. & Rogulina, A. V. (2020), ‘Simulation of vehicle collisions at an unregulated intersection based on the Monte Carlo method’, *Journal of Physics: Conference Series* **1441**.
- Voulodimos, A., Doulamis, N., Doulamis, A. & Protopapadakis, E. (2018), ‘Deep Learning for Computer Vision: A Brief Review’, *Computational Intelligence and Neuroscience* .
- Wada, Y. & Liou, M. S. (1997), ‘An accurate and robust flux splitting scheme for shock and contact discontinuities’, *SIAM Journal of Scientific Computing* **18**(3), 633–657.
- Wang, Z., Xiao, D., Fang, F., Govindan, R., Pain, C. C. & Guo, Y. (2018), ‘Model identification of reduced order fluid dynamics systems using deep learning’, *International Journal for Numerical Methods in Fluids* **86**(4), 255–268.
- Wei, H., sheng Wang, W. & xuan Kao, X. (2023), ‘A novel approach to ultra-short-term wind power prediction based on feature engineering and informer’, *Energy Reports* **9**(December), 1236–1250.
- Westermann, P. & Evins, R. (2019), ‘Surrogate modelling for sustainable building design – A review’, *Energy and Buildings* **198**, 170–186.
- Wilkinson, C. R. & Anderson, J. G. (2003), *An Introduction to Detonation and Blast for the Non-Specialist*, DSTO Systems Sciences Laboratory, Edinburgh, Australia.
- Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., Haddock, S. H., Huff, K. D., Mitchell, I. M., Plumbley, M. D., Waugh, B., White, E. P. & Wilson, P. (2014), ‘Best Practices for Scientific Computing’, *PLoS Biology* **12**.
- Wu, C., Lukaszewicz, M., Schebella, K. & Antanovskii, L. (2013), ‘Experimental and numerical investigation of confined explosion in a blast chamber’, *Journal of Loss Prevention in the Process Industries* **26**(4), 737–750.
- Wu, J. S. & Tseng, K. C. (2005), ‘Parallel DSMC method using dynamic domain decomposition’, *International Journal for Numerical Methods in Engineering* **63**(1), 37–76.
- Wu, M., Jin, L. & Du, X. (2020), ‘Dynamic responses and reliability analysis of bridge double-column under vehicle collision’, *Engineering Structures* **221**.
- Xiao, D., Fang, F., Buchan, A. G., Pain, C. C., Navon, I. M. & Muggeridge, A. (2015), ‘Non-intrusive reduced order modelling of the Navier-Stokes equations’, *Computer Methods in Applied Mechanics and Engineering* **293**, 522–541.
- Xiao, D., Heaney, C. E., Mottet, L., Fang, F., Lin, W., Navon, I. M., Guo, Y., Matar, O. K., Robins, A. G. & Pain, C. C. (2019), ‘A reduced order model for turbulent flows in the urban environment using machine learning’, *Building and Environment* **148**(October 2018), 323–337.

- Yeo, I.-K. & Johnson, R. A. (2000), 'A new family of power transformations to improve normality or symmetry', **87**(4), 954–959.
- Zaghloul, A., Remennikov, A. & Uy, B. (2021), 'Enhancement of blast wave parameters due to shock focusing from multiple simultaneously detonated charges', *International Journal of Protective Structures* **12**(4), 541–576.
- Zahedi, M. & Golchin, S. (2022), 'Prediction of blast loading on protruded structures using machine learning methods', *International Journal of Protective Structures* .
- Zaleski, D. P. & Prozumant, K. (2018), 'Automated assignment of rotational spectra using artificial neural networks', *Journal of Chemical Physics* **149**(10).
- Zaparoli Cunha, B., Droz, C., Zine, A. M., Foulard, S. & Ichchou, M. (2023), 'A review of machine learning methods applied to structural dynamics and vibroacoustic', *Mechanical Systems and Signal Processing* **200**(April), 110535.

# Appendix A

## BA in 3D: Example read-in file

```
=====
>> Algorithm Settings
=====
12                ...number of models
3                ...number of geometries
1                ...number of probabilistic groups
0.25            ...influence mesh density
1                ...charge locations as deviations (1,yes / 0,no)
0                ...domain scaling (1,yes / 0,no)
m                ...length units (mm/m)
kg              ...mass units (g/kg)
=====
>> File paths (using '/' instead of '\')
=====
C:/Users/temp/Documents/geometry/    ...path to geometry files
C:/Users/temp/Documents/results/     ...path for results
=====
>> Geometries
=====
model_1.stl model_2.stl model_3.stl  ...geometry file name
0 0 0                                ...geometry position 'x'
0 0 0                                ...geometry position 'y'
0 0 0                                ...geometry position 'z'
1 1 1                                ...geometry material group
0 0 0                                ...minimum boundary coordinate 'x'
0 0 0                                ...minimum boundary coordinate 'y'
0 0 0                                ...minimum boundary coordinate 'z'
0 0 0                                ...minimum boundary type 'x' (0,transmit / 1,reflect)
0 0 0                                ...minimum boundary type 'y'
1 1 1                                ...minimum boundary type 'z'
4 4 4                                ...maximum boundary coordinate 'x'
4 4 4                                ...maximum boundary coordinate 'y'
4 4 4                                ...maximum boundary coordinate 'z'
0 0 0                                ...maximum boundary type 'x'
0 0 0                                ...maximum boundary type 'y'
0 0 0                                ...maximum boundary type 'z'
=====
>> Probabilistic settings
=====
TNT                ...charge material
1600               ...charge density (kg/m^3)
4.52e6            ...charge energy (J/kg)
101325            ...ambient pressure (Pa)
288               ...ambient temperature (K)
=====
>> Models
=====
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1    ...probabilistic number
1 2 3 1 2 3 1 2 3 1 2 3            ...geometry number
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2    ...charge coordinate 'x'
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1    ...charge coordinate 'y'
1 1 1 2 2 2 3 3 3 4 4 4            ...charge coordinate 'z'
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3    ...charge size
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 ...termination time (ms)
=====
>> Viper::Blast settings
=====
1                ...solving method
1                ...mapping 1D to 3D (1,yes / 0,no)
0.5              ...CFL
0.01             ...cellsize (m)
=====
>> output
=====
4                ...number of gauges (name, x, y, z)
'1'   0.5   3.5   1.5
'2'   1.5   3.5   1.5
'3'   2.5   3.5   1.5
'4'   3.5   3.5   1.5
=====
```

Figure A.1: Branching Algorithm read-in file example.