NORTHWESTERN UNIVERSITY

Physics-Informed Data-Driven Prediction and Design in Advanced Manufacturing Processes

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Mechanical Engineering

By

Mojtaba Mozaffar

EVANSTON, ILLINOIS

September 2021

ABSTRACT

Physics-Informed Data-Driven Prediction and Design in Advanced Manufacturing Processes

Mojtaba Mozaffar

Manufacturing processes are known for their intricacies in changing material shapes and properties. New generations of manufacturing technologies, known as flexible manufacturing, are moving toward design freedom, which allows producing parts with optimized geometries and high customizations at an affordable cost even for low-volume productions. Two prominent flexible manufacturing processes that are of interest in this dissertation are additive manufacturing and incremental sheet forming. An important limiting factor in advancing current capabilities in such processes is the difficulty to reliably understand and control them due to the complex multi-physics and multi-scale nature of the processes. As the result, current practices are overly conservative, significantly limiting the vast potential of producing parts with customized material properties.

At the same time, we observe a surge in the digitalization of manufacturing processes. Today, manufacturing facilities are more connected to data centers than ever, and various measurement methods are becoming standard components of modern manufacturing pipelines from controlling and monitoring the progress while manufacturing to testing and analysis after the part is built. Therefore, this dissertation is dedicated to developing computational methods to advance modeling and design capabilities with a focus on approaches to optimally use manufacturing data—an underutilized asset of manufacturing systems. My contributions in process characterization and design are organized into five research tasks and briefly discussed below.

Predicting the spatiotemporal behavior of manufacturing processes is challenging due to the long history-dependent correlations and complex unstructured geometric features common in manufacturing. Motivated by this challenge, my first research contribution introduces a data-driven methodology to learn material behaviors on unseen geometries over long simulation periods. My method efficiently combines a recurrent neural network to capture material evolution over time and a graph representation to flexibly extract geometric features. This methodology is demonstrated on thermal prediction of additive manufacturing processes and shows great generalizability across industrial-grade parts.

Plasticity is one of the important pillars of computational mechanics. Conventional plasticity methods heavily rely on restrictive assumptions to reduce the dimensionality of the problem into so-called "effective" parameters. In a first-of-a-kind research, my second contribution proposed a data-driven approach to material constitutive modeling, where the material behavior under complex elasto-plastic loading conditions can be learned from data. My work not only shows that data-driven constitutive modeling is accurate, but also it is computationally efficient and performs well across multiple material systems including composites and metal alloys.

The large design spaces of flexible manufacturing such as the additive manufacturing process present a daunting optimization task, which limits the capabilities of producing highly customized parts. In the third contribution of my dissertation, I proposed a reward-driven solution to the toolpath design problem, where an agent is trained to explore the environment and develop strategies to collect maximum rewards. Four methods (three model-free and one model-based) varying in their exploration and decision-making formulations are developed and tested to design toolpaths for over 400 sections and the results show the effectiveness of this methodology especially in the presence of a dense reward structure.

In my fourth research contribution, I developed a differentiable manufacturing simulator that enables a seamless integration between physics-based and data-driven methods. I demonstrate that the gradients of a physics-based thermal simulation of the additive manufacturing process can be computed using automatic differentiation. Furthermore, this differentiable simulation is combined with neural networks to effectively optimize time-series process parameters and reach ideal thermal responses or melt pool behavior over hundreds of simulation time steps.

The computational expense of physics-based manufacturing models is a limiting factor in the size of the problem that can be reasonably solved especially applications such as iterative design, model predictive control, and uncertainty quantification. In my fifth contribution, I investigated modern heterogeneous computational hardware to accelerate the simulation of additive manufacturing processes. Using the proposed matrix assembly and flux calculation strategies on graphical processing units, a speedup of $100 - 150X$ is achieved compared to an optimized CPU implementation.

ACKNOWLEDGMENTS

This research would not have been possible without the support of many. First and foremost, I want to express my deepest gratitude to my advisors, Prof. Jian Cao and Prof. Kornel Ehmann, who assisted me through the ups and downs of my Ph.D. life, gave me motivation and guidance to find my career path, and helped me grow as a person. I also want to thank Prof. Gregory Wanger who has been incredibly kind and helpful to me throughout the past five years in classes and meetings. I am thankful to Prof. Aggelos Katsaggelos, who introduced me to the world of machine learning and encouraged me to pursue this research.

I am indebted to a long list of colleagues, collaborators, and friends. I will always remember the helps I received from Dr. Newell Moser, Dr. Marco Giovannini, Dr. Huaqing Ren, Mr. Dohyun Leem, and Mr. Shuheng Liao. A big thanks to Dr. Ramin Bostanabad, Dr. Miguel Bessa, Dr. Maysam Gorji, and Dr. Arindam Paul for making amazing collaborations possible.

Last but not least, I am eternally grateful to my family for their unlimited love--Abbas and Narjes for their life-long sacrifices, Maryam and Mohammad for their constant support, and especially Xiaolu for always being there for me.

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AM	Additive Manufacturing
ANN	Artificial Neural Network
BVP	Boundary Value Problem
CPP	Coverage Path Planning
CPS	Cyber-Physical System
DED	Directed Energy Deposition
DFM	Design For Manufacturing
DOE	Design of Experiments
DQN	Deep Q-Network
DSIF	Double-Sided Incremental Forming
FCNN	Fully Connected Neural Network
FEM	Finite Element Method
GDP	Gross Domestic Product
GNN	Graph Neural Networks
GP	Gaussian Process
GRU	Gated Recurrent Unit
ISF	Incremental Sheet Forming
LSTM	Long Short-Term Memory
MCTS	Monte Carlo Tree Search

MDP	Markov Decision Process
ML	Machine Learning
MSE	Mean-Square-Error
NN	Neural Network
PBF	Powder Bed Fusion
PDE	Partial Differential Equations
PPO	Proximal Policy Optimization
PSPP	Process, Structure, Property and Performance
RGNN	Recurrent Graph Neural Network
RL	Reinforcement Learning
RMSE	Root Mean Squared Errors
RNN	Recurrent Neural Network
SAC	Soft Actor Critic
SPIF	Single-Point Incremental Forming
SVM	Support Vector Machines
TPIF	Two-Point Incremental Forming
UCB	Upper Confidence Bound
CPS	Cyber-physical system

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1. Motivation

Manufacturing is a major industrial sector accounting for 10-30% of the gross domestic product (GDP) in leading industrial countries (West et al. 2018). Historical examples show innovations in manufacturing nurture key advances in the automotive, aerospace, electronics, and biomedical industries. However, many manufacturing processes are known for their intricacies in prediction and control due to the extreme loading conditions and complex material flows. These complexities are often exacerbated in modern high-value-added processes, such as Additive Manufacturing (AM) and incremental sheet metal forming, as they are governed by hierarchical multi-scale and multi-physics behaviors.

In recent years, Artificial Intelligence (AI) has become increasingly more capable of automating activities that we associate with human thinking, such as planning, decision making, and problem-solving. A nine-fold increase in the number of publications over the past 20 years (Shoham et al. 2018), 113% increase in start-ups from 2015 to 2018 (Shoham et al. 2018), and an estimated 15.7 trillion-dollar worth of economy in 2030 (PwC 2020) are only a few indications of the existing and future vast impact of AI in both academia and industry. Meanwhile, the recent trend of digitalization in the manufacturing community not only allows more precise control of processes, but also provides cost-efficient access to high-quality large scale data which can be used to achieve more efficient, agile, and innovative manufacturing solutions as a viable alternative to costly and time-consuming experimental approaches (Lee et al. 2015, Hermann et al. 2016).

1.2. Need for Data-Driven Modeling and Design Tools

The uncertainty in the prediction and control of the responses of manufacturing processes is one of the most critical challenges facing state-of-the-art practices. The underlying physics of the material response is challenging to understand and to predict since it involves multi-physics phenomena over a wide range of scales ranging from the micro-scale (such as the influence of a laser beam on powder particles) to the meso-scale (e.g., grain evolution and void generation) to the macro-scale (e.g., the thermal behavior of the material and its influence on an AM build's fatigue life, anisotropic behavior, and strength, to name but a few). Many modern manufacturing processes involve a large set of interconnected process parameters including material properties, setup, environment, tools, and motion parameters, which makes it extremely difficult to establish the influence of process parameters on product properties.

There are two common classes of approaches for characterizing manufacturing processes: (i) computational models, and (ii) experimental studies. Computational mechanics has been a popular method to characterize processes at the macro-scale (Parry et al. 2016, Schoinochoritis et al. 2017), meso-scale (Khairallah et al. 2016, Rai et al. 2016), and multi-scale domains (Wolff et al. 2017, Yan et al. 2018). The relationships between process, structure, property and performance can be individually examined and modeled. This approach, known as the PSPP framework, was initially popularized in the field of material science (Olson 1997) and later adopted to also characterize other manufacturing processes, e.g., connecting process parameters to thermal behavior, porosity, and mechanical behavior of an AM build (Wolff et al. 2017). In a noteworthy work, Yan et al. (Yan et al. 2018) developed a multi-physics model using the PSPP framework to achieve an in-

depth understanding of the AM process. In their work, the Monte Carlo method is used to simulate an electron beam heat source at the micro-scale, a thermal-fluid flow model for simulating powder particle evolution at the meso-scale, and a macro-scale thermal analysis of the process by using the finite element method (FEM). They further simulated the microstructure evolution of the material using the cellular automata method and calculated its mechanical response using crystal plasticity.

A crucial problem with the existing predictive methods for manufacturing processes is their enormous computational cost that might take weeks or months of simulation time (Francois et al. 2017). Often, the computational models are orders of magnitude slower than the experiments, which makes them impossible to use in time-sensitive applications such as real-time control or optimization procedures. Moreover, the physics-based models can have significant discrepancies with experimental data due to simplifying assumptions or incomplete physics.

On the other hand, many experimental studies have been conducted to analyze the influence of process parameters on microstructure and build properties for manufacturing processes (Shamsaei et al. 2015, Wang et al. 2016, Stevens et al. 2017, Fisher et al. 2018, Li et al. 2018). However, experiments are often expensive due to the cost of the required equipment, materials, and manpower. Additionally, the aforementioned experiment-based models use a limited number of experiments and are incapable of taking into consideration the complex inter-connectivity of parameters as well as high-dimensional inputs. Therefore, investigating novel methods to predict and control manufacturing behavior is vital for overcoming existing barriers and satisfying the ever-evolving requirements for modern processes, e.g., AM and incremental forming technologies.

1.2. Research Tasks and Accomplishments

We hypothesize that AI-empowered approaches can address the shortcomings of state-of-the-art methods described in the preceding section. This idea stems from the recent advancements in AI hardware and software capabilities, which offer high predictive power and computational efficiency, and allow leveraging digitalized manufacturing and large-scale data acquisition platforms. To this end, we develop various data-driven architectures and establish new methodologies for two tasks in enhancing (i) process characterization and modeling, and (ii) process design, as they collectively contribute to better manufacturing solutions. As depicted in **Figure 1.1**, my research encompasses several topics at the intersection of computational mechanics and artificial intelligence which I have pursed individually and through several collaborations.

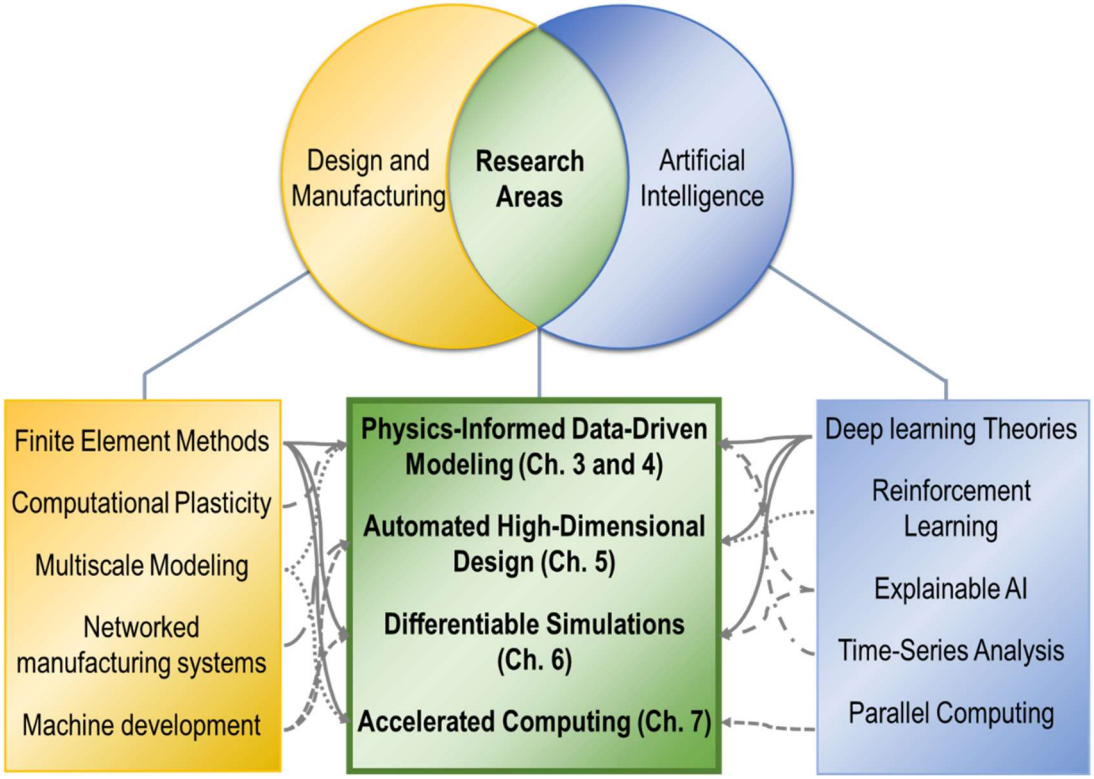


Figure 1.1. An overview of presetned research topics which lie at the intersection of computational mechanics and artificial inteligence.

This dissertation presents five interconnected research tasks as depicted in **Figure 1.2**. We begin with the development of manufacturing process characterization methods with a special attention to improve their ability to generalize across process parameters, geometries, and materials. Later, we present an accelerated physics-based computing package—an essential tool for real-time control and large-scale database construction. Finally, we investigate AI-enabled methods to explore new design solutions for manufacturing processes to process complex unstructured data (e.g., unstructured geometries) as well as concurrent history-dependent parameters (e.g., toolpath and forces). Each task is briefly described hereunder:

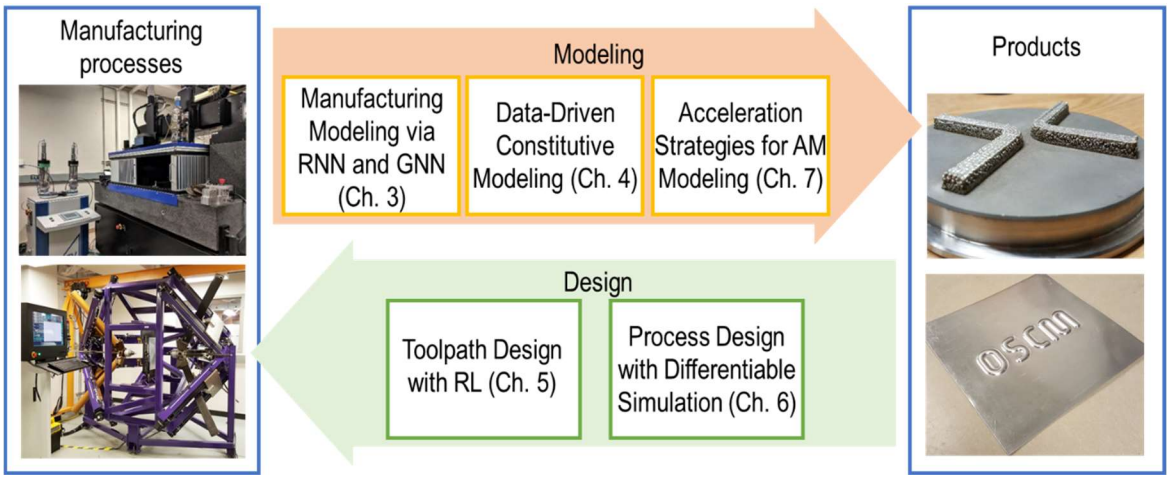


Figure 1.2. Summary of research tasks in two categories of (i) manufactuirng modeling and (ii) process design.

- Research Task 1: Data-Driven Spatiotemporal Manufacturing Process Modeling using Neural Networks. The focus of this task is on data-driven predictive models for manufacturing processes to capture the relationships between process parameters and material behavior under extreme conditions characterized by long history-dependent correlations and complex unstructured geometries. Our framework is demonstrated on the thermal analysis of AM processes. A recurrent neural network (RNN) architecture is proposed to predict the high-dimensional thermal history with variations in build dimensions, toolpath strategy, laser power and scan speed. Our results indicate that the model can accurately predict the thermal history of any given point of the AM build on a test-set database with limited training. The model's ability to accurately predict thermal histories has been demonstrated through an overarching test conducted for long periods. In the second phase of this research task, we improve the capability of our data-driven approach to generalize across challenging industrial-grade geometries by

proposing a novel hybrid graph-based recurrent neural network, which maintains high accuracy on unseen geometries over long simulation periods.

- Research Task 2: Data-Driven Constitutive Modeling for Computational Plasticity. In this task, a first-of-a-kind data-driven constitutive model is proposed to accurately capture path-dependent responses of material systems via deep learning. Two sources of complications in elasto-plastic constitutive modeling are identified as geometry-induced and model-induced complexities and each source is thoroughly studied. A database with representative volume element samples of fiber-reinforced composites is developed which includes variations in microstructure descriptors (e.g., fiber volume fraction, fiber radius, fiber distances) and compound loading conditions. High-fidelity numerical simulations are utilized to analyze the stresses, plastic energy, and total energy of each sample. A novel recurrent neural network structure is proposed to efficiently combine temporal and non-temporal inputs of the constitutive model and accurately predict the elasto-plastic behavior of the composite. Our trained model conveniently achieves an under 0.5% scaled root-mean-square-error for training data as well as loading conditions outside of its training database. Moreover, we demonstrate that the model can implicitly capture yield surface evolution which shows that our approach automatically detects and learns hidden plasticity concepts.

- Research Task 3: Toolpath Design using Deep Reinforcement Learning. Toolpath optimization in manufacturing processes is currently hampered by the high

dimensionality of its design space. In this task, a reinforcement learning platform is proposed that dynamically learns toolpath strategies to build an arbitrary part. To this end, three prominent model-free and one model-based reinforcement learning formulations are investigated to design AM process toolpaths and demonstrated for two cases of dense and sparse reward structures. The results indicate that this learning-based toolpath design approach achieves high scores, especially when a dense reward structure is present.

- Research Task 4: High-Dimensional Manufacturing Process Design via Differentiable Simulations. This research task presents a novel computational paradigm for process design in manufacturing processes that incorporates simulation responses to optimize manufacturing process parameters in high-dimensional temporal and spatial design spaces. We developed a differentiable finite element analysis framework using automatic differentiation which allows accurate optimization of challenging process parameters such as time-series laser power. We demonstrate the capability of our proposed method through three illustrative case studies in AM for: (i) material and process parameter inference using partial observable data, (ii) controlling time-series thermal behavior, and (iii) stabilizing melt pool depth. This first-of-a-kind research task opens new avenues for high-dimensional manufacturing design using solid mechanics simulation tools such as finite element methods.

- Research Task 5: Acceleration Strategies for Finite Element Analysis of AM using Graphical Processing Units. In this task, a novel approach to accelerate the explicit finite element analysis of the transient heat transfer of AM processes is proposed using Graphical Processing Units (GPUs). The challenges associated with this approach are enumerated (e.g., matrix assembly and flux calculation bottlenecks) and multiple strategies to overcome each challenge are discussed. The performance of the proposed algorithms is evaluated on multiple test cases for directed energy deposition and selective laser melting processes. Speed-ups of about $100 - 150X$ compared to an optimized single CPU core implementation for the best strategy were achieved.

1.4. Dissertation Outline

The aforementioned research tasks, collectively establish the scientific and technological foundations for "Physics-Informed Data-Driven Prediction and Design in Advanced Manufacturing Processes". In the following chapter, the technical background on manufacturing processes, and physics-based and data-driven modeling methods are reviewed. Then, each research task is comprehensively discussed in Chapters 3-7. Each research task begins with the motivation behind it, continues with an in-depth discussion on methodology and results, and concludes with a summary of final remarks and future works. Chapter 8 summarizes the novelties and contributions of this research and presents guidelines for future research directions. References and supplemental materials can be found at the end of the dissertation.

CHAPTER 2

Technical Background

In this chapter, we introduce the recurring background concepts of the dissertation with the goal to establish common nomenclature and facilitate the reading experience for all. We start by discussing the two manufacturing processes of interest in the presented research (i.e., AM and forming processes). Later, the fundamentals of physics-based modeling and their established practices are reviewed, and lastly, we briefly introduce state-of-the-art data-driven modeling techniques and relevant formulations to this research.

2.1. Introduction to Flexible Manufacturing Processes

Flexible manufacturing is a modern industrial paradigm in which manufacturing systems can adapt to changes in part design, material, and even the order of operations. This paradigm offers an alternative to traditional high-volume processes (such as casting and stamping) and shows superior cost and energy efficiency for low- and medium-volume products as they require low tooling and setup costs. Flexible manufacturing is an inevitable topic of today's manufacturing as there is an increasing demand for one-of-a-kind parts across many industries (e.g., aerospace and biomedical). Given that, modern processes such as AM and incremental sheet forming (ISF) provide an exciting path toward advancing current manufacturing capabilities; in this dissertation, we developed tools involving both and, hence, provide an introduction to these processes and their variations.

2.1.1. Additive Manufacturing Processes

Additive manufacturing is a relatively new manufacturing process. The early concepts of AM can be traced back to two centuries ago in topography and photo-sculpture (Bourell et al. 2009), but the first AM technology with modern components was proposed by Swainson for plastic prototyping in 1977 (Swainson 1977). Since then, AM technologies have been increasingly offering new capabilities such as fast production, compatibility with complex geometries, support for various classes of materials, and low environmental imprint. These features led to the fast growth of AM in biomedical, aerospace, and automotive fields over the past decades. While developing new AM processes and enhancing existing ones is an active field of academic and industrial research, the ASTM standard recognizes seven families of AM processes including VAT photopolymerization, powder bed fusion, binder jetting, material jetting, sheet lamination, material extrusion, and directed energy deposition, each offering a unique set of features and capabilities (A. S. T. M. 2012).

Metal powder-based AM processes are increasingly employed due to their advantages in producing functional parts with limited manufacturing time and cost. Nowadays, applications of metal powder-based AM processes go beyond just producing prototypes, but are also used for manufacturing products with complex geometries (Yang et al. 2012), varying alloy compositions (Guo et al. 2015, Wenjun et al. 2015), and locally-controlled microstructures (Dehoff et al. 2015, Tan et al. 2015). Our research effort in this dissertation is focused on two metal power-based AM processes, namely, Directed Energy Deposition (DED) and Powder Bed Fusion (PBF). Schematics of these two AM processes are demonstrated in **Figure 2.1**. DED is a class of AM processes that uses focused heat sources, usually an electron or laser beam, to melt the powders and

simultaneously delivers the powder to the focal point of the heat source as the powder delivery nozzle follows the toolpath derived from CAD geometries (Gibson et al. 2010, Gu et al. 2012). PBF is another category of AM processes in which a thin layer of powder is delivered to the base plate using a powder delivery system and then the laser is used to gradually melt and fuse the powder (King et al. 2015).

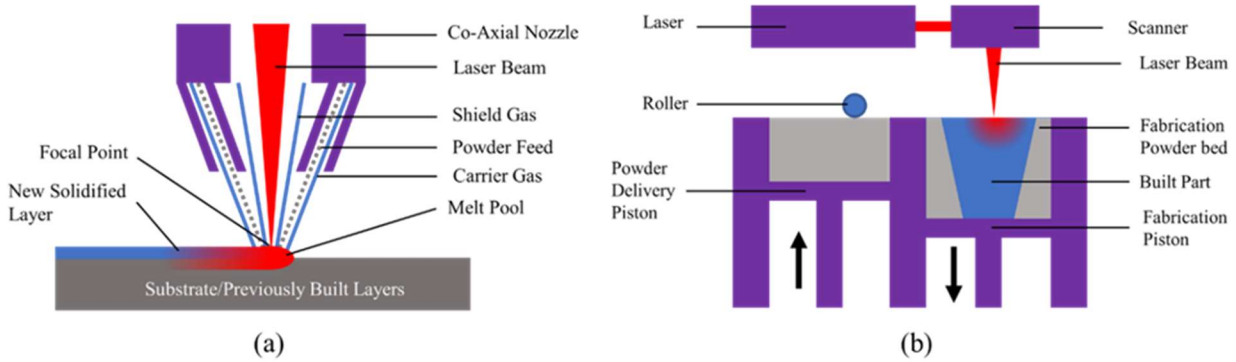


Figure 2.1. Schematic of two AM processes; (a) DED process with coaxial nozzle to deliver powder to the focal point of a laser and (b) PBF process that uses a roller to spread a thin layer of powder before melting the layer (Mozaffar et al. 2019).

From various challenges that are currently facing AM technologies, the lack of established design principles and engineering practices requires immediate attention as it severely restricts the benefits and capabilities one can expect from AM processes. AM design and modeling involve simultaneous physical phenomena happening at different length scales and long time spans—naturally leading to high-dimensional and computationally expensive problems. Current best practices to manufacture a successful part either entails expensive trial and errors or attempts to conservatively opt for safe parameters and design setting, both of which avoid pushing AM processes to their limit and unlock their full potential.

2.1.2. Incremental Sheet Forming Processes

Sheet metal forming processes play an important role in producing durable metallic goods. In contrast to traditional forming processes, such as stamping and deep drawing, where the part is built using costly part-specific tooling, ISF uses generic tooling and incrementally deforms sheet metals until the desired part is achieved (see **Figure 2.2** for an example of ISF configuration). ISF not only saves the time and costs associated with designing, manufacturing, and maintaining tools, but also offers higher formability as it delays local necking and premature fracture in forming parts. Different configurations of ISF are available commercially and within the research community, such as single-point incremental forming (SPIF), two-point incremental forming (TPIF), double-sided incremental forming (DSIF), and accumulative DSIF which differ in terms of the contact and toolpath strategies used. For the sake of brevity, the interested readers are referred to (Moser 2019) for a detailed description and analysis of various ISF processes.

While ISF shows great capabilities in low-volume productions, it also presents unique challenges in modeling and controlling the process compared to traditional approaches. One aspect of these challenges is due to the complex loading histories caused by ISF where the material experiences multiple non-proportional cycles of loading and unloading in different directions. Experimental studies show that these loading conditions cause advanced elasto-plastic behavior in many metallic materials such as the Bauschinger effect, transient hardening, permanent softening, hardening stagnation, and overshooting effect (Barlat et al. 2013, Bruschi et al. 2014).

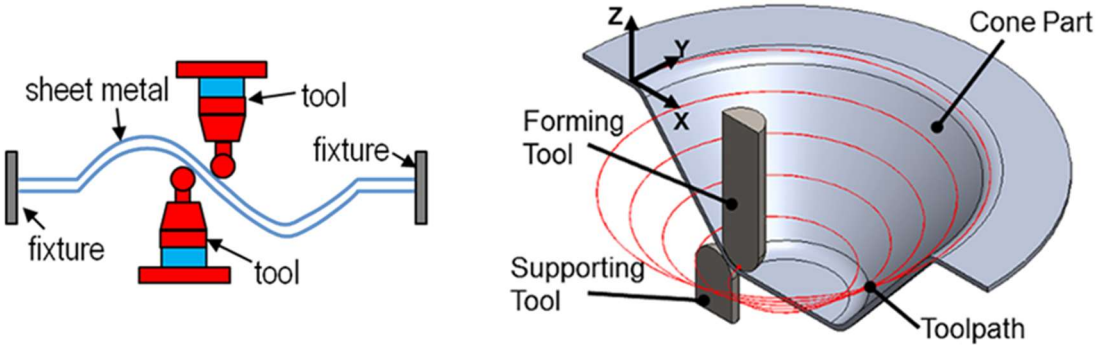


Figure 2.2. Schematic of the double-sided incremental forming process (DSIF). DSIF is a flexible, dieless sheet metal forming process, which incrementally accumulates localized deformations to form the final part. The figure is a courtesy of the AMPL research group.

2.2. An Overview on Physics-Based Modeling Techniques in Manufacturing Processes

Modeling attempts to understand an observed phenomenon and to reproduce the event as close as possible. Therefore, modeling enhances our knowledge of the laws of the world around us, which by its own has been, and still is, an intrinsic value for many great scientists. Besides, modeling allows us to exploit physics laws to design better engineering solutions, making it a core pursuit in engineering practices and recently in many others such as animation production and virtual reality fields. The modern usage of models stems from the massive computational power available today that allows the simulation of extremely large systems. By utilizing realistic simulation tools, one can avoid costly experimental tests. A renowned example of such instances is the car crash test, where the destructive experiment is very expensive and most of such tests are replaced by simulation tools today.

Using the underlying physics to simulate processes allows the capture of complex phenomena through relatively simple known laws of nature. They often lead to robust solutions and generalize well as long as similar physics is applicable. However, analytical physics-based solutions are rare in today's engineering practices, because realistic problems operate on complex domains (e.g.,

unstructured geometries) and boundary conditions. A popular physics-based approach with a wide range of applications in the design and development of engineering solutions is FEM. It is noteworthy that there are several alternatives to FEM for modeling physical phenomena over complex domains such as the finite difference method, finite volume method, and even meshless methods such as the material point method, but here we focus on FEM due to its extensive usage in manufacturing applications. FEM analysis starts with determining the applied physics and their deriving formula. For example, transient heat transfer and Navier-Stokes formulas are partial differential equations (PDE) commonly used for solving thermal and fluid simulations, respectively. The underlying PDE combined with the boundary conditions applied to the problem form a boundary value problem (BVP), which is also known as the strong form of an FEM formulation.

The ultimate goal of FEM is to approximately solve the BVP over any given domain by converting the strong form into a system of equations. In FEM, one performs this task by first calculating the weak form of the BVP, which can be achieved by integrating the strong form with an arbitrary variation, known as the test function. The benefit of the weak form is that it can be integrated over each part of the domain independently, which is a key FEM concept allowing the discretization of the solutions. The FEM domain is broken down into numerous elements and the weak form is applied over each element. As it is difficult to represent and solve for an arbitrary function of the unknown field, $u_{x,y}$, the arbitrary function is redefined with a shape function, a known function of element shape $N_{x,y}$, and field values at nodal points u_i. For instance, the field function of an 8-node hexahedron element would have 8 degrees of freedom as it is defined based on its 8 unknown nodal values. Therefore, by applying the weak form over all elements in the

simulation, a system of equations as a function of nodal values is obtained, which can be solved numerically using various solvers such as the Gauss-Jordan Elimination method. After the nodal values are known, one can calculate smooth fields for derived quantities (e.g., stresses and strains) using shape functions.

When simulating time-dependent physics (i.e., when the effect of acceleration and motion cannot be neglected), we also need a time integration scheme to incrementally simulate consequent time steps. Implicit and explicit solutions are two popular approaches for time integration. In implicit solutions, every time step is solved through a backward Euler integration scheme, which ensures a stable and accurate solution, even with large time steps. However, this solution comes with a significant computational cost of large matrix inversions. Explicit solutions, on the other hand, use a forward Euler integration scheme avoiding iterative solvers; however, this method does not guarantee satisfying the weak form and small time steps are required to ensure the stability of the results. A more in-depth FEM formulation for transient heat transfer is provided in Chapter 7.2.

Without undermining the massive services physics-based modeling techniques (including FEM) have done to advance our lives over the past century, there is room for improvements in some aspects of such models. While each model has its own strength and weaknesses and needs to be evaluated individually, here we shortly discuss the common drawbacks of physics-based modeling. As a first argument, there are many instances that a known meaningful physics is not established, rather, practices rely on phenomenological laws based on limited observations, which makes them prone to inaccuracies. In addition, despite vast improvements in the availability and quality of computational resources, simulating the full physics can be too time-consuming and

infeasible for realistic applications. This is especially a key concern in flexible manufacturing processes, as they require long process time and simultaneously operate on length scales spanning 2-3 different order of magnitude. Additionally, practical use of physics involves various assumptions from their core formulations (even most renowned physics are bounded by the scale they are valid in), to their implementations, and how they represent the reality of the phenomena. Using invalid assumptions or accumulation of assumption inaccuracies can easily jeopardize the quality of a seemingly perfect physical simulation.

2.3. An Overview on Data-Driven Modeling Techniques

In recent years, we see a surge in the usage of data-driven modeling techniques as an alternative to physics-based approaches. Data-driven modeling is particularly a widespread topic in academic studies in fields where the physics is too complex to model (e.g., computer vision and natural language processing) or in dynamic decision-making scenarios (e.g., robotics). Today's success stories of data-driven modeling are fueled by advancement in three pillars: (i) availability of high quality and quantity data as the result of widespread sensors, social media, and networking; (ii) improvements in computational hardware (e.g., CPUs and GPUs) and innovations in new specialized hardware (e.g., TPUs), and (iii) advent of algorithmic and engineering solutions to efficiently process data. In this subsection, we introduce prominent families of data-driven modeling.

2.3.1. Machine Learning

Machine learning (ML) is a collection of statistical concepts used for pattern recognition that allows computers to capture correlations in data and draw inferences (Samuel 1959). There are three main categories of ML solutions: supervised, unsupervised, and reinforcement learning. Supervised learning aims to find the unknown mapping between inputs and outputs given a database of such pairs. In most formulations, first, a model is trained on database instances, and one trained, it can be used to predict unseen cases. Linear regression and logistic regression are two classical supervised models with dense theoretical backbone and widespread industrial adoption as they lead to extraction of explainable features. Support Vector Machines (SVM) aims to find a hyperplane in parameter space that maximizes the decision boundary margin, often leading to robust solutions. Feature engineering is an important aspect of SVM models as they rely on linear feature correlations. Decision trees are another well-known supervised learning tool, where a tree data structure of conditional statements determines the model prediction. While decision trees provide powerful classifiers, they are prone to excessive overfitting and artificially orthogonal decision boundaries. Ensembled ML is an intriguing concept that argues that a combination of multiple models leads to an improved model as long as the sources of error in models are different. Popular examples of ensembled machine learning include random forest (ensembled decision trees trained on subsets of data), AdaBoost, Gradient Boosting, and stacking algorithms (Géron 2019).

Unsupervised learning performs the task of finding patterns within unlabeled data. Many hypothesize that unsupervised learning is the future of learning algorithms as it relaxes the database requirements of supervised learning. Current prevalent applications of unsupervised

learning include clustering, dimensionality reduction, and feature extraction using techniques such as k-means clustering, principal component analysis, and t-SNE. Dimensionality reduction is by far the most utilized application of unsupervised learning in mechanics and allows the extraction of a dense representation of the input, discards irrelevant features and noises, speeds up the training process, and enables subsequent usage of computationally heavy algorithms such as Gaussian processes.

Finally, reinforcement learning (RL) aims to maximize the reward that an agent can collects while interacting with an environment. As the agent collects its own data, in contrast to a stationary database for supervised learning, RL has known to be one of the most challenging aspects of AI technologies. Traditional RL approaches such as Markov Decision Process (MDP), Q-learning, and Monte Carlo Tree Search (MCTS) were originally limited to small state spaces; however, modern integrations of core RL theories with neural networks as a universal function approximation have led to multiple breakthroughs in this field over the past five years (Sutton et al. 2018).

As one can see, there is a range of ML solutions, each suitable for different applications, data types, and sizes. However, common drawbacks of many traditional ML algorithms include the heavy reliance of carefully engineered features and the lack of scalability to large systems. As neural networks offer a great potential to overcome these two drawbacks, the research and many industrial communities have drastically shifted their attention to this class of ML, which is introduced in the following section.

2.3.2. Neural Networks

The concept of neural network (NN) as a model of how brain neurons work was first introduced by the neurophysiologist Warren McCulloch and the mathematician Walter Pitt in 1943 (McCulloch et al. 1943). Over time, NNs proved to offer outstanding flexibility in approximating complex nonlinear functions once they are arranged in multiple connected layers, known as a deep neural network. Geoffrey Hinton et al. demonstrated the first successful implementation of deep neural networks to recognize handwritten digits with an impressive precision of over 98% (Hinton et al. 2006).

Countless types of NN formulations and architectures are proposed for a variety of tasks and innovating new ones is a vibrant field of research to this day. Therefore, in this section, we only introduce the most fundamental formulation of NN (i.e., the fully connected neural network), and the details of each specialized NN used and developed in the dissertation are discussed in each research task. A fully connected neural network (FCNN) transforms multiple input signals (x_j) of each neural to its output signal (n_i):

$$n_i = f\left(b_i + \sum w_{i,j} x_j\right) \tag{2.1}$$

where $w_{i,j}$ refers to the weights for neuron i in connection with input neuron j, b_i is the bias for neuron i and f is a nonlinear activation function. An FCNN is formed by arranging neurons in layers where all neurons in one layer are connected to all neurons in their subsequent layer (see **Figure 2.3**).

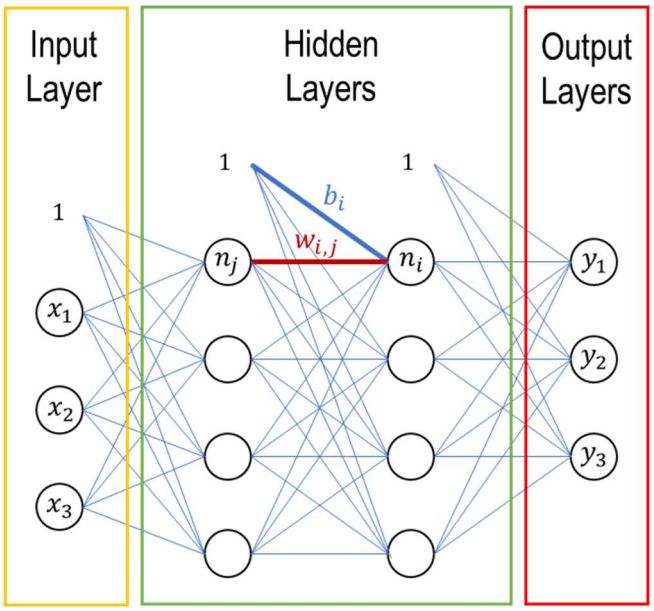


Figure 2.3. Architecture of a fully connected neural network.

Given an input layer X, the FCNN network generates an output of $\hat{Y}$. In a supervised learning fashion, where pairs of correct input-outputs are available we can define a loss based on the difference between network prediction $\hat{Y}$ and database prediction Y (e.g., mean-squared-error) and optimize the weights and biases of the network to minimize the loss. While any optimization method can be potentially used to do such a task, gradient descent is by far the most popular optimization method for neural networks. This is because the gradient of the error with respect to weights and biases can be efficiently computed using backpropagation algorithm, which allows simultaneous optimization of large sets of parameters. Although gradient descent does not provide a theoretical guarantee of global optimality, empirically the iterative process of updating network parameters leads to close-to-optimal solutions given an appropriate initialization.

Despite the vast successes in the development and deployment of NN structure over the past few years, many research questions are yet to be answered, especially with respect to its application

in manufacturing. Designing new NN structures that are compatible with evolving requirements of manufacturing modeling and design tools is a non-trivial task. A fascinating research problem is to find effective approaches to integrate known physics into data-driven modeling. In addition, advancing capabilities on two issues of generalizability and explainability is invaluable. This is especially vital in flexible manufacturing applications where one-of-a-kind designs are common (hence, models should generalize well to new samples) and modeling mistakes can lead to expensive damages (hence, models should be explainable so that one can verify its decisions). Finally, novel integrations of data-driven solutions within manufacturing systems are essential and present many research opportunities—some of which are addressed in the following research tasks.

CHAPTER 3

Data-Driven Spatiotemporal Manufacturing Process Modeling using Neural Networks

3.1. Introduction

Manufacturing processes manipulate material structures to achieve desired functional and geometrical requirements by adding, subtracting, or deforming the material. While some processes such as stamping rapidly transform the material, many other such as machining, AM, and ISF rely on aggregating small local changes to gradually build a part—a common theme in flexible manufacturing processes. In many of such processes, although the changes are applied locally, their effects expand beyond the immediate region of applied energy and globally influence major properties of the part. For example, in DED, a laser beam induces a local melt pool to absorb material particles in a small area; however, the generated heat can re-melt previously deposited layers and alter the residual stress, porosity, and material properties in significantly larger sections of the part. Similarly, in ISF, while the majority of deformation is targeted toward the area under the influence of the tool, local changes propagate stresses throughout the part causing global elastic and plastic deformations. Therefore, the final properties of the part in many flexible manufacturing processes are determined by long sequences of history-dependent influences of the tools as they change the geometric and material structures of the part.

Modeling the spatiotemporal behaviors of manufacturing processes allows better understanding of the processes which can lead to an increase in their efficiency and capabilities and reduce defective parts. Physics-based modeling has been the standard practice for most

manufacturing modeling applications. Although, the strengths and weaknesses of each modeling approach need to be evaluated individually, broadly speaking, high-fidelity physics-based models (e.g., FEM and CFD) come with a cost of extreme computational requirements, low-fidelity and lack accuracy in complex settings (e.g., analytical, and phenomenological models). This is especially true in case of flexible manufacturing processes where simulations are long, and the involved physics operate on scales spanning over 2-3 orders of magnitude.

In this chapter, we introduce data-driven modeling as an alternative to physics-based approaches with the goal to advance the discovery, diagnosis, and optimization of advanced manufacturing processes. It is noteworthy that we consider AM as the main process of interest as it provides many unique modeling challenges. Additionally, AM is one of the most digitalized and sensor intensive manufacturing processes, making it a prime candidate for data-driven modeling. However, the methodology offered here can be easily deployed in many other flexible manufacturing settings where high predictive power is required, and physics-based modeling does not provide a viable solution.

In Chapter 3.2, we show that data-driven methods can accurately capture long history-dependent features in manufacturing simulations. Recurrent neural network (RNN) based architectures are developed to predict the thermal history of any points in a part during AM processes. The effect of long history-dependent thermal states on future thermal responses is captured through an RNN cell of which the input variables include toolpath strategy, deposition time, boundary, laser state, laser intensity, and layer height. Temperature simulation results of different geometries built using varying process parameters are applied to train and test the proposed model, which presents great generalizability across process parameters. Later in Chapter

3.3, we propose a graph-based modeling approach to address one of the key shortcomings of data-driven modeling techniques, i.e., generalizability across unseen complex geometries. Finally, we conclude this chapter by summarizing our findings and potential future research directions in this field in Section 3.4.

3.2. Temporal Data-Driven Modeling of Manufacturing Processes using Recurrent Neural Networks

In many manufacturing processes, the final product is the result of accumulative work on the material over a potentially long process, which leads to challenging history-dependent features, i.e., some properties of the material are determined by history of the process and not solely by the time instant when it directly interacted with a tool. This is common theme that can be observed across a wide range of forming processes (e.g., ISF, rolling, drawing), welding processes, AM, hybrid processes, to name but a few. Here, we introduce our data-driven approach with the key focus on capturing high-dimensional time-series features of such processes.

To showcase our approach, we chose AM as it involves notoriously challenging temporal features. As mentioned in Chapter 2.1.1, despite AM's immense potential, its widespread adoption is hampered by current challenges in prediction and control such as quality variability and process inefficiencies. Experimental studies have been conducted to analyze the influence of process parameters on microstructure and build properties in (Shamsaei et al. 2015, Wang et al. 2016, Stevens et al. 2017, Fisher et al. 2018, Li et al. 2018). However, these models use a limited number of experiments and are incapable of taking into consideration the complex inter-connectivity of AM process parameters. Physics-based models were proposed to capture thermo-mechanical,

thermo-fluidic, and/or microstructural evolution of the process in (Chiumenti et al. 2017, Wang et al. 2017, Bostanabad et al. 2018). However, these models are not applicable in many cases because of their enormous computational cost that might take weeks or months of processing time, even on supercomputers (Francois et al. 2017), and their discrepancy with experimental results due to the simplifying assumptions made or incomplete physics.

Current trends in manufacturing, such as Industry 4.0 (Zühlke 2013) and cyber-physical systems (Lee et al. 2015), increase the visibility and accessibility to information, which leads to extensive research in data-driven models in the manufacturing community (Lee et al. 2013, Bostanabad et al. 2016, Bessa et al. 2017, O'Donovan et al. 2018). The application of machine learning in a polymer powder bed fusion process was discussed in (Baturynska et al. 2018). A data-driven model for characterizing geometrical dimensions of trace products using dense neural networks was proposed by (Caiazzo et al. 2018). In (Kamath et al. 2017), the authors compared multiple regression models for developing a surrogate model for AM simulation. The self-organizing map technique was used for quantifying the geometric accuracy of the Fused Filament Fabrication process in (Khanzadeh et al. 2018). Wang et al. (Wang et al. 2020) developed a machine learning approach to predict tool wear in machining processes. The neural networks are used to predict the behavior of the materials in sheet forming processes (Gorji et al. 2019).

Due to the limited number of samples and the omission of crucial time-series features of the process such as the toolpath, which directly affects thermal history and hierarchical microstructure in AM, most data-driven models fail to provide a profound understanding of this process. To address this gap, we propose a recurrent neural network structure for predicting the thermal history of any given point in an AM build in a many-to-many configuration. The proposed approach is

well-suited for AM processes since it can accurately calculate the high-dimensional thermal history of the builds in a computationally efficient manner.

3.2.1. Introduction to Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a special type of artificial neural network (ANN), designed with the idea that the outcome of each neuron is dependent on its input (like other types of ANN) and a history variable from past operations, which enables this structure to work with sequential data. In particular, the Long Short-Term Memory (LSTM) (Hochreiter et al. 1997) and the Gated Recurrent Unit (GRU) (Cho et al. 2014) are two successful formulations of RNN structures for training long sequences of data. The underlying concept of RNN structures, i.e., the use of information from previous steps combined with the current state of the system, is in line with differential equations. Thus, physics that can be formulated with ordinary or partial differential equations, such as the finite element method (FEM), are good applications for RNN models. RNNs allow for the temporal dependency in the input data to be learned without the need to specify a fixed set of lagged observations. Traditional time series analysis such as auto-correlation requires the identification of seasonality and stationarity in a time series, which may change based on laser speed, size of the build, etc., and need to be explicitly adjusted for each simulation. Further, neural networks are robust to noise in input data and the output variables and can learn despite missing values. These properties of RNNs led to the hypothesis that the RNN structure can predict the temperature history in AM regardless of its highly nonlinear nature.

For RNN formulation, the Gated Recurrent Unit (GRU) (Cho et al. 2014) is adopted in this work as follows:

$$r_t = sig(W_r.[h_{t-1}, c_t] + b_r) \quad (3.1)$$

$$z_t = sig(W_z.[h_{t-1}, c_t] + b_z) \quad (3.2)$$

$$\hat{h}_t = tanh(W.[r_t \times h_{t-1}, c_t] + b) \quad (3.3)$$

$$O_t = h_t = (1 - z_t) \times h_{t-1} + z_t \times \hat{h}_t \quad (3.4)$$

Where the GRU cell takes a candidate update (c_t) and the hidden state h_{t-1} as the input, and generates an output at the current time, O_t. r_t and z_t represent the reset and update gates correspondingly where each uses separate sets of weights (W), biases (b), and a sigmoid activation function. The output (O_t) is used as the hidden state for the next time step (h_t) and the hidden state is initialized with an initial field h_0.

3.2.2. Proposed Model Architecture

A stacked RNN structure with GRU formulation is considered in this research as depicted in **Figure 3.1**. Each GRU cell receives input features (x_t) for that time step and a hidden state from the previous time step (h_{t-1}) and outputs a new hidden state (h_t). The number of units in each GRU cell represents the dimension of the hidden state, which is connected to other cells using weights and biases. Using multiple layers of the GRU structure enables the model to comprehend deeply hidden correlations in the data. Fully connected layers are considered to combine the outputs of the GRU units into a single time-series temperature output.

The stacked GRU model is developed using the Keras deep learning library (Chollet 2015) with the Adam (Kingma et al. 2014) optimization procedure and a mean-square-error (MSE) cost function between the model prediction and the database thermal history.

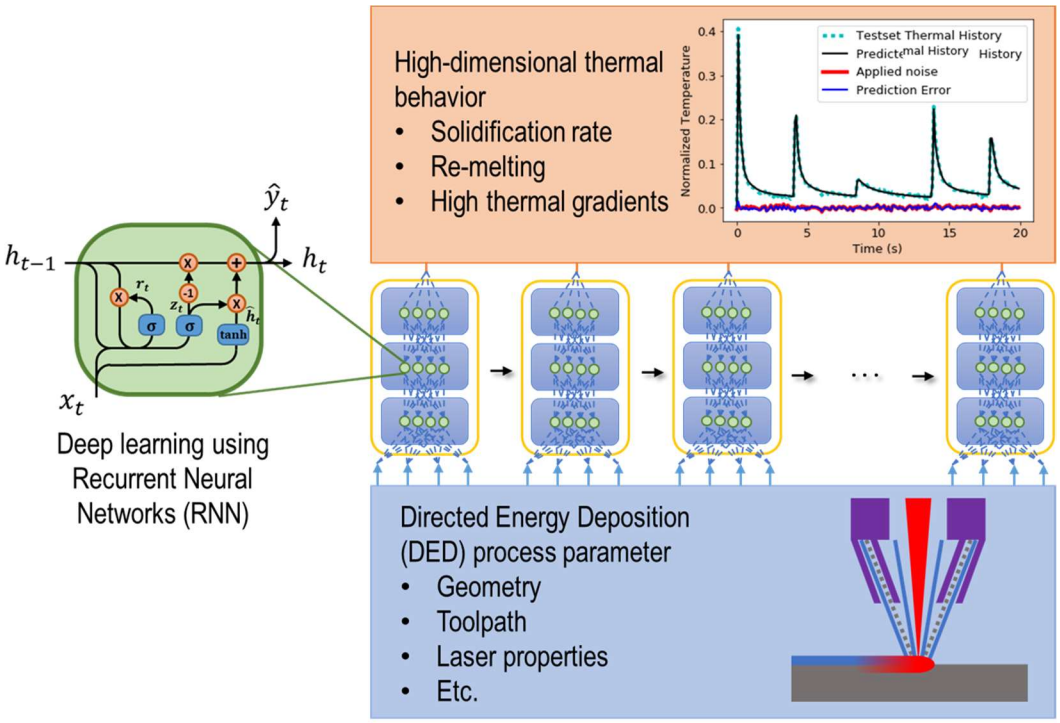


Figure 3.1. Schematic of the many-to-many stacked RNN structure with GRU formulation in relation with process inputs and thermal outputs; Green circles represent GRU units, blue rectangles represent GRU cells, yellow boundaries represent stacked GRU wrappers, and blue dashed lines within the GRU wrappers represent trainable parameters. The schematic and formulation of GRU units are provided on the left based on the formulation given in (Cho et al. 2014, Olah 2015).

3.2.3. Database Development and Characteristics

The database considered for training the model is built using an in-house finite element code, GAMMA, for transient thermal analysis using an explicit solver (Smith et al. 2016). This choice enables us to have access to high volumes of input data required for training the model to learn

about the phenomena occurring not only on the exterior of the AM build but also within its interior. A wide range of input parameters such as laser power (500-1,000 W), scan speed (5-30 mm/s), toolpath strategy (e.g., zigzag, unidirectional or circular motion), geometric size (5-40 mm), and shapes (e.g., cubic, cylindrical, thin wall) are used to generate over 250,000 training points for DED simulation of stainless steel 316L objects.

The input features designed for each training point include toolpath feature (based on the distance between the coaxial powder nozzle outlet and the birthed element), the time of deposition, closest distance to the boundary of the build, layer height, laser intensity, and laser state (on or off). Input features are extracted as time series data from FEM output files and stored in a three-dimensional tensor compatible with the RNN structure. To investigate the capability of the RNN structure to operate in noisy environments, an artificial noise with a standard deviation of 30 K is added to the thermal history in the database. All input features and thermal histories are normalized using a linear mapping from the minimum and maximum of each feature to the range of 0 to 1 in order to speed-up the optimization procedure. The model is trained over 80% of the database, while 20% of the database is left out for testing.

3.2.4. Results and Discussion

The RNN model is trained for different configurations such as number of RNN layers (1-5 layers), GRU units (100-500 units), and fully connected layers (1-3 units). Even for configurations with a small number of layers and units the model reaches 1e-4 MSE after 100 epochs of training. An epoch is a complete pass of training through the database in batch mode. For a configuration of 3 stacked GRU layers with 500 units and 1 fully connected layer with 100 neurons, the model

reaches 3.210e-5 MSE on training data and 3.84e-5 on testing data with 100 epochs of training. The training process took approximately 40 hours on a Nvidia Quadro P5000. The prediction for two random points of a test-set over 20 s of the process is demonstrated in **Figure 3.2**. The smoothness of the predicted thermal histories and the similarity between the patterns of the applied noise and the prediction error (discrepancy between the noisy database and predicted thermal history) proves that the RNN structure can effectively avoid noise in the database and capture most of the critical features of the thermal history including sharp changes and flat regions that happen due to the phase transitions in the material.

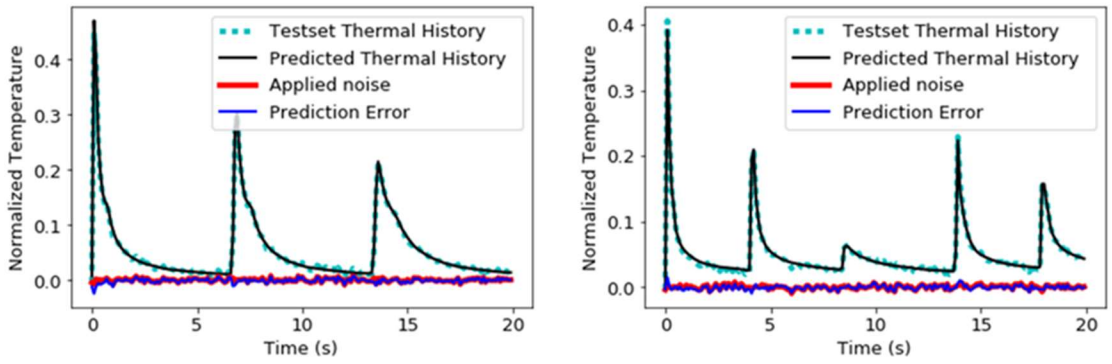


Figure 3.2. Evaluation of the stacked RNN model on the test dataset for two random points over 20 s of the DED process; Comparison of the model prediction (black line) and the test-set value (cyan dashed line) for the thermal history of a point in a thin-wall build with uni-directional toolpath (left) and a cylindrical build circular toolpath (right).

One of the key features of RNN structures for this application is that it can predict any arbitrary number of time increments (0.1 s in this case). To assess the generality of the model, a trained model is used for predicting a longer time span than it has been trained for. Particularly, two stacked RNN models trained for 20 s and 50 s of the process are used for predicting 100 s of the thermal history. The model trained on 20 s (**Figure 3.3** top) performs well for the training period

and continues to have a reasonable prediction for more than three times of the time span it has been trained on, to have an overall MSE of 7.05e-5 for 100 s of the process. The model trained on 50 s (**Figure 3.3** bottom) decreases the MSE to 3.17e-5 for 100 s of the prediction. This significant drop in the long-term error of the model comes with the cost of almost two times the required training time and resources. It is noteworthy that the sharp gradients, melting and re-melting of the material, which are the main targets of this work, happen in a few layers after material deposition and after that the temperature changes smoothly. Therefore, the stacked RNN is an effective model for predicting the most critical features of the thermal history, while the long-term predictions can be easily improved if necessary.

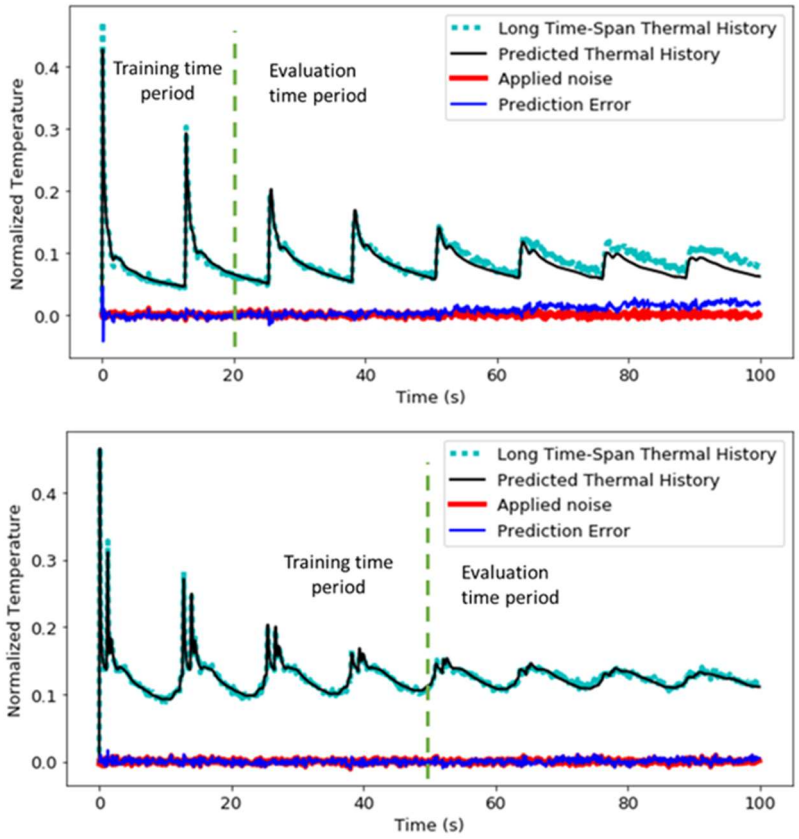


Figure 3.3. Evaluation of the stacked RNN model on the test dataset for 100 s, while trained on 20 s (top) and 50 s (bottom) of the process; Comparison of the model prediction (black line) and the test-set value (cyan dashed line) for the long-time span thermal history of a point in a cubic build with zigzag toolpath.

The capability of the trained model is further investigated for predicting the behavior of a dissimilar geometry type from the training database. A new database with different geometric features is developed, and the trained model is used to predict the thermal history of three points of the build. As depicted in **Figure 3.4**, the model can accurately predict the behavior of points 1 and 3. However, there is significant error in the model prediction for point 2, which is because the geometric feature and the state of the boundaries close to this point is unprecedented to the model. The results indicate that even with two hand-picked geometric features introduced in Section 3.2.3 (i.e., closest distance to a boundary of the build and layer height features) the model can perform

reasonably well at material points of untrained builds with similar geometric features as the training database. However, to represent complex geometries, a more flexible geometric feature extraction methodology will be needed.

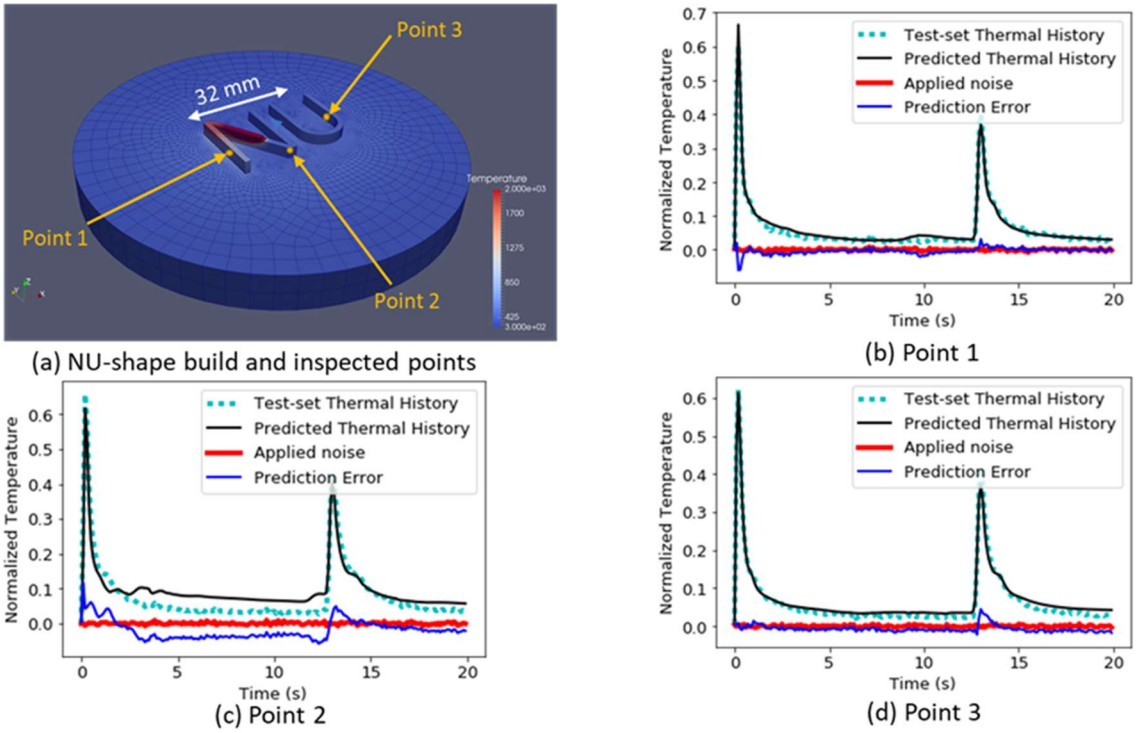


Figure 3.4. Evaluating the trained model on a dissimilar geometry; The NU-shape build geometry and the inspected point locations (a), comparative figures for the points 1, 2 and 3 between model prediction and the test-set (b), (c), and (d), respectively. The toolpath of this build goes from the buttom left to the upper right side of the letter N and then moves from the upper left to the upper right side of the letter U.

3.3. Geometry-Agnostic Data-Driven Manufacturing Modeling using Graph Neural Networks

Generating accurate shapes is one of the most important aspects of a successful manufacturing process. Geometry is not just the output of a manufacturing process, but also a key factor in determining the appropriate materials and manufacturing processes. In fact, an entire sub-field of manufacturing, i.e., design for manufacturing (DFM), is dedicated to establishing best practices for designing geometric features that lead to favorable manufacturing outcomes. Therefore, considering that geometric features have significant effect on part properties, it is crucial for simulation tools to accurately model this relationship. Like the previous section, we demonstrate our methodology on AM processes, however, noting that the same approach can be utilized across many manufacturing processes.

High-fidelity computational models can offer accurate representation and modeling of complex geometries; however, they require massive computational resources and time making them infeasible in time-sensitive applications. Therefore, many research attempts investigated alternative approaches. Empirical solutions based on experimental data are proposed to model manufacturing processes for real-time prediction and control systems (Song et al. 2012, Jin et al. 2016). Yet, they offer low accuracy as they over-rely on limited experimental data and fail to capture interconnected dependencies in process parameters. Alternatively, many analytical and semi-analytical solutions are proposed to offer a trade-off between computation cost and accuracy of high-fidelity models. While early versions of such analytical solutions were only applicable to single-track scenarios, recent publications extend the capability of such models to multi-track and multi-layer cases (Huang et al. 2019, Ning et al. 2020, Sheng et al. 2020). Despite the recent

progress in the field, analytical solutions deviate from realistic responses even in moderately complex geometries as they include numerous simplifying assumptions.

In the previous section, we showed that data-driven methods can predict long history dependent features of AM. Additionally, we demonstrated that our RNN architecture can predict the behavior of samples with similar classes of geometries to ones in training database with a scaled mean-squared-error of $3e^{-5}$, however, the error can significantly increase in cases for unseen classes of geometries. A surrogate model was proposed by Roy et al. (Roy et al. 2020) to achieve real-time thermal history prediction. A set of input features (e.g., distance from moving heat distance and cooling surface) are carefully selected from the geometry representation using GCode to reduce the computational demands. Their method can be generalized to different sizes of the same geometry, various process parameters and different materials achieving a predictions accuracy of 95%. Two machine learning-based models using extreme gradient boosting (XGBoost) and long short-term memory (LSTM) are proposed by Zhang et al. (Zhang et al. 2020) to predict thermal history using six input variables, including laser power, scan speed, layer index, time index, average height, and width. Models are trained and validated using real-time experimental measurements taken by IR thermal cameras under setups with varying process parameters. The authors reported a best runtime of 0.34 s for XGBoost, which is small enough for real-time prediction of data captured by IR cameras. In the study by Haghighi et al. (Haghighi et al. 2020), physics-based and data-driven models are combined to characterize filament bonding and porosity distribution in extrusion-based additive manufacturing for PLA material, which is a thermoplastic polyester. An analytical heat transfer model was applied for thermal profile characterization and an artificial neural network was adopted for filament deformation characterization. Their hybrid

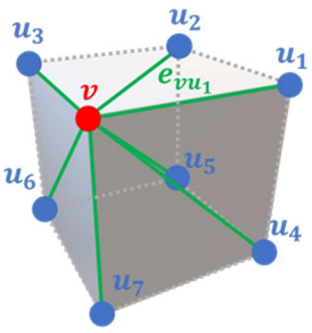
model achieved an average accuracy of 95% and 94% in modeling inter-layer and intra-layer bonding, respectively. In summary, however, the aforementioned data-driven approaches demonstrate their results only for simple geometries, such as thin walls and cubic structures.

As one can see, despite major achievements in data-driven modeling, state-of-the-art approaches fall short on a key issue—generalization across unseen geometries, which is crucial for AM modeling as AM is mostly used for producing one-of-a-kind and unstructured geometries. Given these limitations, the objective of this work is to introduce a novel physics-aware data-driven predictive model that can benefit from the high predictive power and computational efficiency of AI-empowered models while drastically improving their ability to generalize across challenging geometries. Our approach captures the intricacies of the physics through a graph representation, which provides a flexible representation of complex unstructured geometries. Additionally, the approach follows the local contributions of each node to other nodes within an element (in our case, a hexahedron connecting 8 nodes) and provides fundamentally similar calculation pathways to primary physics-based approaches, i.e., FEM.

3.3.1. Introduction to Graph Neural Network

Graph neural networks (GNN) are deep learning architectures that capture dependencies in unstructured graphs via message passing between neighboring nodes. Due to their flexibility, GNNs have rapidly gained popularity in social sciences (Wang et al. 2018), chemistry (Fout et al. 2017), and image processing (Wang et al. 2018). The two major categories of GNN includes spectral-based and spatial-based methods. Spectral-based methods (e.g., GCN (Kipf et al. 2016), AGCN (Li et al. 2018), CHEBNET (Defferrard et al. 2016)) define a convolution operation based

on the Laplacian eigen-basis, which makes this class most applicable to problems with a static graph representation (Liu et al. 2020). In contrast, spatial-based GNNs define the convolution operation on the graph by aggregating the information from neighboring nodes and edges. Spatial-based GNN formulations consist of three steps: (i) message creation and propagation at the starting nodes, (ii) message aggregation at the target nodes, and (iii) calculation update based on the aggregated message. Here, two nodes are neighbors if they belong to the same element. A schematic of our definition of neighboring nodes and the three message-passing steps are demonstrated in **Figure 3.5**.



$$(i) \quad Message_{vu_1} = f(h_u, e_{vu_1})$$

$$(ii) \quad Message_{v} = AGG(Message_{vu_i})$$

$$(iii) \quad h_v^{new} = f'(Message_{v}, h_v)$$

Figure 3.5. Schematic of a target node and its neighboring nodes within an element. Message passing includes three fundamental steps: (i) message construction, (ii) message aggregations, and (iii) and target update.

we hypothesize that the spatiotemporal dependencies in AM processes which are traditionally modeled via physics-based simulations, such as FEM, can be alternatively captured using graph neural networks. This hypothesis stems from the resemblance between finite element matrix assembly operations (which combine the local contributions of element interconnectivity) and

graph neural network message-passing formulations. To test this hypothesis, we developed a database of simulation-based thermal responses while building a variety of industrial-grade AM parts. We investigate two GNN architectures to reproduce AM responses. The schematics of the two architectures are depicted in **Figure 3.6** and their details are elaborated upon in what follows.

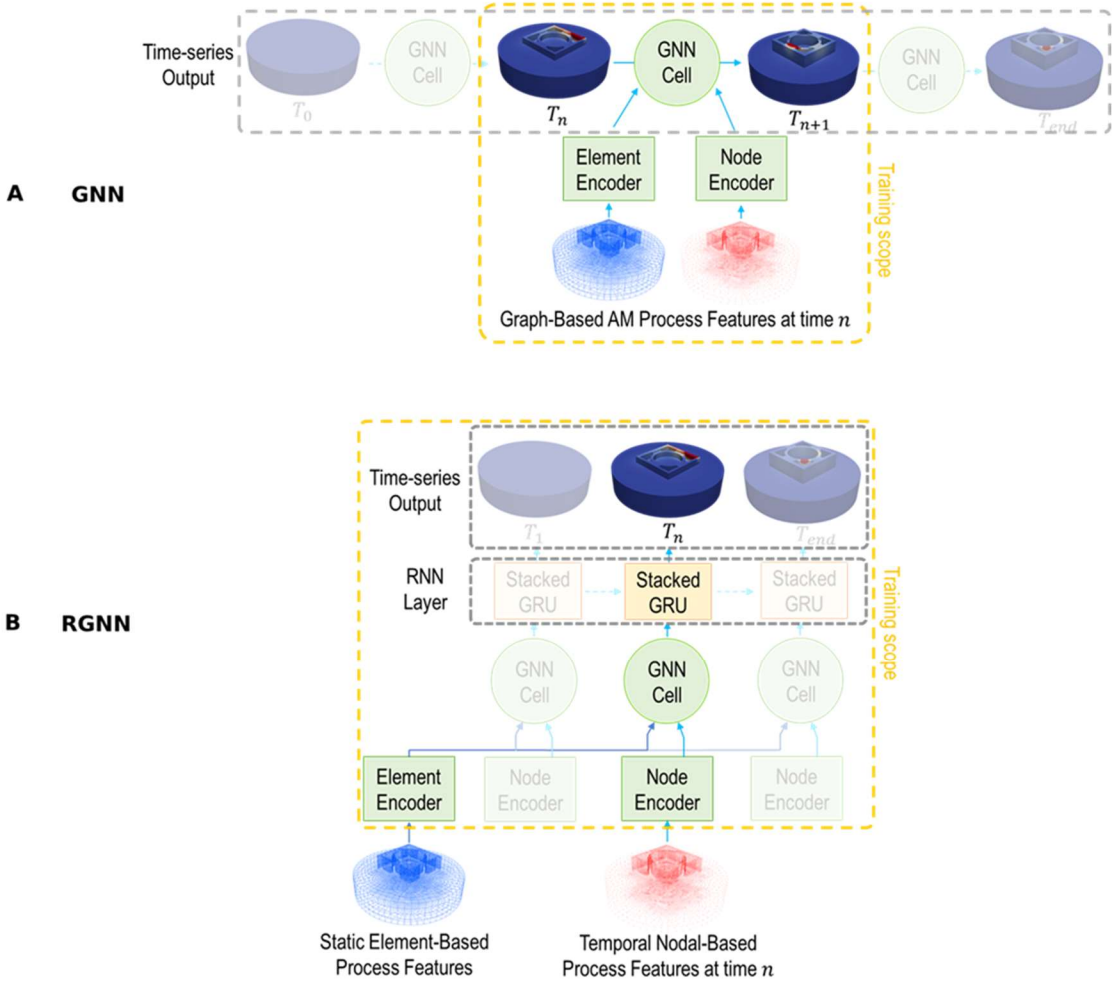


Figure 3.6. Schematics of the two architectures for spatiotemporal prediction of AM thermal responses: **(A)** The GNN architecture predicts the single-time step update in each training instance given the node and element features at the time-step; **(B)** The RGNN architecture predicts and trains multi-time step interactions where at each time step the network receives a temporal nodal-based encoded representation, a non-temporal element-based representation, and the hidden state of the previous stacked GRU cell and outputs the thermal distribution over the geometry. Both architectures can be recursively evaluated to produce thermal outputs of arbitrary length.

3.3.2. Proposed Network Architectures

GNN Architecture

As the first option, we devised a GNN-based architecture in which the network receives the thermal responses in the previous time step as well as process parameters at that time including nodal and element-based features (detailed in Section 3.3.3) and outputs the thermal responses in the current time step, as depicted in **Figure 3.6A**. In essence, the network is responsible for calculating thermal fluxes at each time step to properly update the thermal fields. In this network, each training or evaluation step predicts one forward time increment, however, by recursively providing the network output as the next time increment's input we can produce time-series responses of arbitrary length. For the GNN cell formulations in **Figure 3.6A**, we found that DeeperGCN (Li et al. 2020) empirically leads to better results compared to existing alternatives, though it is noteworthy that in our experience the difference in performance has been insignificant. In the DeeperGCN formulation, h_v^l (node features for the lth layer) will be passed to a layer-normalization layer, an activation layer with a rectified linear unit, and a drop-out layer for feature preprocessing. Subsequently, a spatial convolution is performed with the preprocessed features to update the features for the next layer. The spatial convolution operation is defined as (Li et al. 2020):

$$h_v^{(l+1)} = MLP\left(h_v^{(l)} + AGG\left(\left\{ReLU\left(h_u^{(l)} + e_{vu}^{(l)}\right) + \varepsilon : u \in \mathcal{N}(v)\right\}\right)\right) + h_v^{(l)} \tag{3.5}$$

where $\mathcal{N}(v)$ is the set of neighboring nodes of v, $h_u^{(l)}$ are the neighbor node feature and $e_{vu}^{(l)}$ are the features of edges connecting v and u. $ReLU$ is the rectified linear unit activation function and

AGG is the aggregation function, which is the SoftMax function in the current work. MLP is a multi-layer perception and ε is a small positive constant to assure numerical stability. As shown in Eq. (3.5), the encoded neighboring node features $h_u^{(l)}$ and edge features $e_{vu}^{(l)}$ are first added and fed to an activation function to construct the message from each neighboring node. The message is aggregated and then combined with nodal features in an MLP to calculate an update. Finally, the updates are added to the previous node to construct the output of the GNN layer. The last step provides the residual connection which not only facilitates the network training process, but also improves the interpretability of the models as it resembles thermal fluxes.

RGNN Architecture

As an alternative architecture, to potentially better model long temporal dependencies of the thermal system, we developed a Recurrent Graph Neural Network (RGNN) architecture as depicted in **Figure 3.6B**. In this architecture, the network takes the time-series nodal features and edge features as the input and directly generates time-series thermal responses over an arbitrary number of time steps. At each time step, a GNN cell is used to capture local interactions between nodes and elements using their corresponding features. By concatenating the candidate updates generated by the GNN cells with shared parameters, a time-series candidate update (c_t) is assembled, which feeds into a stacked RNN layer. For the RNN formulation, the Gated Recurrent Unit (GRU) (Cho et al. 2014) is adopted in this work (see Eq. 3.1-3.4). In our implementation, the GRU cell takes a candidate update (c_t) and the hidden state s_{t-1} as the input, and outputs the

temperature field at the current time, T_t. The output (T_t) is used as the hidden state for the next time step (s_t) and the hidden state is initialized with the initial temperature field T_0.

3.3.3. Geometric Database Development and Characterization

We developed a database based on high-fidelity finite element simulations, which has allowed us to have access to the thermal histories of all geometric points. An explicit in-house AM simulation package, GAMMA (Smith et al. 2016, Mozaffar et al. 2019), is used to solve the transient heat transfer equations while we gradually activate elements as the toolpath passes over an predefined mesh. Heat conduction, convection, radiation, and external heat flux as the result of the laser beam are modeled in our simulations, while stainless steel 316L is used as the material. Key process and material properties are reported in **Table 3.1**.

Table 3.1. Process and material properties for the generated database.

Material Properties (SS316L)		Process and Environmental Properties	
Density	8,000 Kg/m^3	Ambient Temperature	300 K
Heat Capacity	0.5 J/g·K	Laser Power	1 KW
Latent Heat	272.5 J·g	Laser Diameter	2 mm
Conductivity Coefficient	21.4 W/m K	Report time step	0.1 s
Solidus/Liquids	1,648.15/ 1673.15 K	Scan speed	10 mm/s

To ensure that we train and test the proposed models on diverse geometries, we selected 45 different industrial-grade geometries from the ABC database (Koch et al. 2019), where 40 of them are used for training and 5 geometries are randomly separated for testing. Four samples of the

geometries are shown in **Figure 3.7**, while the complete set of geometries is provided in **Appendix A - Figure A.1**. The selected geometries vary in their size, number of layers, shapes, and geometric features (e.g., wall thickness) to capture a wide range of AM builds. CAD geometries are scaled and placed on a substrate of 20 mm height and 100 mm diameter to make them suitable for sample DED manufacturing. All the geometries are meshed using ABAQUS with 8-node hexahedron elements, where each element has an approximate edge size of 5 mm for the substrate and 1 mm for the part, resulting in about 10k - 50k elements in the meshes. To incentivize the network to generalize across different toolpaths, a Python script is developed to automatically generate DED layer-by-layer toolpaths for arbitrary geometries where the contour patterns are randomly selected from 9 toolpath strategies varying in their motion directions, patterns (zigzag versus spiral) and starting positions.

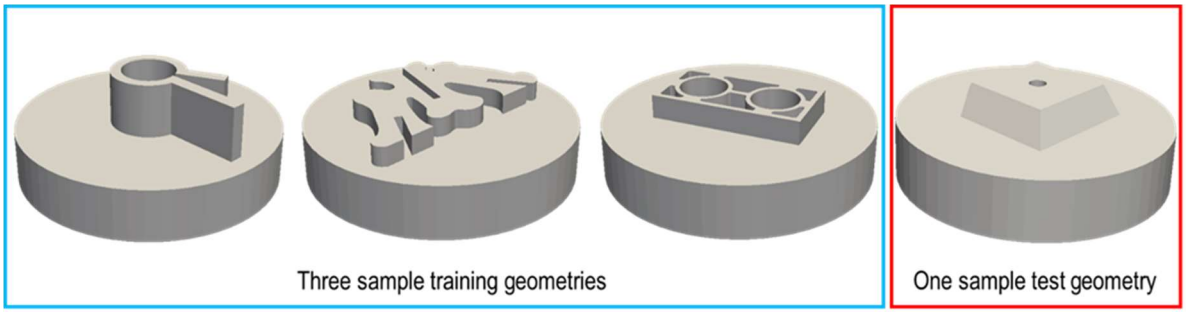


Figure 3.7. Sample AM builds adapted from the ABC online repository (Koch et al. 2019) for industrial-grade geometries. Geometries are oriented and placed on substrate plates to construct the AM simulation database. Three geometries within the blue border are in our training dataset while the geometry within the red border is used as one of the test samples. All geometries are provided in the **Appendix A - Figure A.1**.

We create the database using graph representations, where the graph typology is constructed based on the meshed geometry, i.e., every node of the mesh is defined as a node of the graph, and the edge is defined with a connectivity matrix that indicates which nodes are within a common

element. Each node of the graph is embedded with three features: (i) birth flag (indicating whether a node is active at that time step), (ii) layer height, and (iii) laser distance feature defined as the inverse of the distance between each node and the laser beam at each time step. In addition to matrix connectivity, we consider an element-based distance feature indicating the distances between any two nodes of an element. As element connectivity and feature are constant in time, we implement them as static inputs to the model across all time steps, whereas nodal features are provided as time-series inputs (see **Figure 3.6B**). The two networks require different sampling methods. For the GNN architecture, we randomly sample pairs of inputs (process features and thermal responses at time n) and outputs (thermal responses at time $n+1$) at different times of the simulations. In contrast, the RGNN model receives time series inputs and outputs of 50 consequent time steps (equivalent to 50 mm of deposition), where the starting times are sampled from the length of the simulations. To provide a fair comparison between the performances of the two networks, they train over the same amount of data collected from the same simulations, separated into training and test sets. All inputs and outputs of the models are normalized to values between $0-1$ to assist the optimization process.

3.3.4. Results and Discussion

We implemented the two architectures using the Pytorch deep learning library (Paszke et al. 2019) and Pytroch Geometric package (Fey et al. 2019) in Python. The models are trained on their corresponding training sets, while the test sets are only deployed for evaluation without contributing to backpropagation. Each training epoch consists of a complete pass on training samples in which we update model parameters and simultaneously calculate a mean squared error

(MSE) of the predictions versus database ideal responses. After each epoch, all test samples are evaluated and their MSE errors are stored. We stop each training process when no improvements in the MSE of the test samples are seen. The training process takes approximately 2 and 4 weeks on an Nvidia RTX 8000 GPU for the GNN and RGNN, respectively.

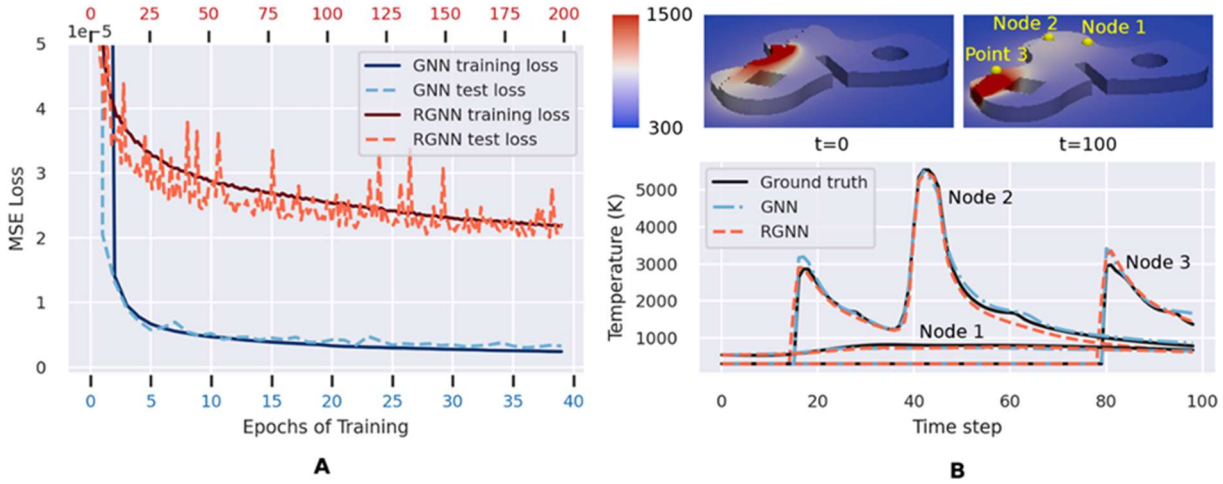


Figure 3.8. Training and evaluation results for the proposed GNN and RGNN formulations: **(A)** The evolution of the train and test losses over training epochs is normalized per node per time step; **(B)** An example simulation and the predicted thermal history at three points with the location of points depicted on the top right and the comparison of histories between GNN, RGNN and the ground truth on the lower right. Note that $t = 0$ refers to the starting time of the 100 time-step test sample and not the entire build.

Our results, depicted in **Figure 3.8**, indicate that the GNN architecture rapidly captures important correlations in data and reaches a satisfactory error of $2.36e^{-6}$ MSE on the training set and $3.24e^{-6}$ MSE on the test set in just 40 training epochs. The GRNN architecture requires $5X$ more epochs to stabilize on the test set MSE with an error of $2.19e^{-5}$ MSE on the training set and $2.20e^{-5}$ MSE on the test set. However, note that the RGNN model attempts to predict 50 steps

into the future—a drastically more difficult task compared to the single time step prediction. Therefore, a higher error is reasonable and expected for the RGNN model. We further demonstrate the output of the developed data-driven models for a sample case in the test set (see **Figure 3.8B**). The output of each trained model is compared to the ground truth simulation results for three randomly selected points on the geometry surface for 100 time-steps. This is possible because both GNN and RGNN models, although being trained on a fixed number of time steps (1 and 50 respectively), can be recursively evaluated for any arbitrary length of time. The GNN model results in $6.07e^{-5}$ MSE and RGNN model in $3.28e^{-5}$ MSE averaged over all nodes of the simulation. Qualitatively, the results show a good agreement between both models and the ground truth, however, the GNN model captures the thermal stagnation around solidification better while the RGNN model predicts the pick temperatures more accurately.

To further investigate the stability and capability of the model for long simulations, we evaluate the models on 45 samples (40 for training and 5 for test sets) over 1,000 time steps, which is $1{,}000X$ and $20X$ the training span of the GNN and RGNN models, respectively (see **Figure 3.9**). Often, such intense extrapolations fail in machine learning; however, we see that both models are capable of reasonable predictions over long periods of time and their errors, although raising over time, remain stable. In **Figure 3.9A-3.9C**, we demonstrate a sample case study for both GNN and RGNN models over 1,000 time steps including their thermal contours as well as root mean squared errors (RMSE) for all material points in each data-driven simulation. While starting similarly, the RGNN model significantly outperforms the GNN model after about 100 time steps, showing a superior capability to capture long interactions. A similar conclusion can be drawn by observing the RMSE evolution over all training and test samples as shown in **Figure 3.9D** where the GNN

model results in a RMSE of 0.0313 for the training set and 0.0383 for the test set, while the RGNN model shows a negligible error propagation with a RMSE of 0.0161 on the training set and 0.0152 on the test set. Impressively, both models result in close performance between the training sets and test sets (the RGNN model even shows a slightly better accuracy on the test set), which shows that they can generalize well across completely unseen geometries for long simulations. Additionally, the RGNN model also shows a better performance in long-term predictions, effectively demonstrating a saturation of error propagation.

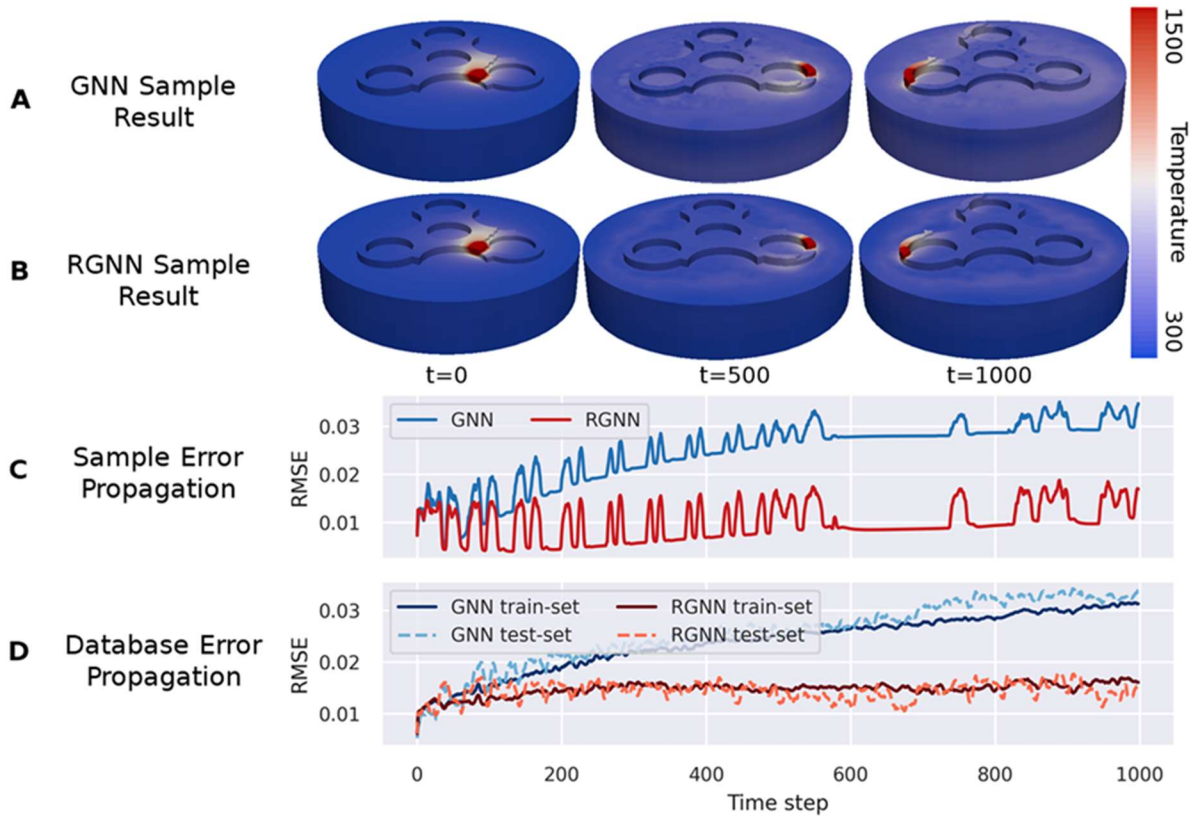


Figure 3.9. Evaluation of the trained models capability to produce long-term simulations. The evolution of the thermal field on a sample simulation is depicted for the GNN and RGNN models (**A** and **B**). The error propagation of the sample simulation and all database simulations for both models are presented (**C** and **D**).

3.4. Conclusions and Future Works

Recent advancements in high-throughput computing combined with the popularity of data-sharing protocols and cyber-physical systems create a unique opportunity to develop data-driven models for heterogeneous materials and challenging manufacturing processes. In this research task, we presented a stacked RNN structure with a GRU formulation of time-series modeling of manufacturing processes and demonstrated that it accurately predicts thermal histories in AM builds. Our results show that the model reaches an MSE of 2.97e-5 on a test dataset after a 100 epoch training. Additionally, two overarching tests for predicting the thermal history over a longer time span and non-trained geometries were also examined, showing the potential of RNN models to predict complex behaviors in AM processes. The accuracy of the model can be further improved by increasing training epochs.

A key gap in the capability of state-of-the-art data-driven models related to their poor generalizability across geometries was also addressed. We demonstrated that our proposed RGNN architecture can effectively capture local intricacies of the process through a graph representation and long-term temporal correlations via a recurrent network structure to achieve unprecedented generalizability over unseen geometries and maintain it through 1,000 time steps, which is over $20X$ of its training span. Our codes and data are made available for the research community at https://github.com/AMPL-NU/Graph_AM_Modeling, to further explore its general applicability in other manufacturing processes.

Further improvement can be achieved by expanding on both key elements of the data-driven models: database and network. As the model heavily depends on the size and quality of the database, an improvement avenue is to broaden the database to different process parameters,

materials, and geometries. Similarly, one can improve the network by deploying larger networks (as we do not observe overfitting in our models) and training over longer time step periods to further reduce the errors. While the presented work trains on simulation data, this framework can also be directly deployed on experimental data as high-quality shared repositories for manufacturing processes are growing. We believe that our approach opens the path for new generations of physics-informed data-driven modeling in manufacturing processes with the flexibility to go beyond thermal responses for predicting more challenging aspects of a wide range of manufacturing processes.

CHAPTER 4

Data-Driven Constitutive Modeling for Computational Plasticity

4.1. Introduction to Computational Plasticity

Understanding the deformation of materials under different loadings is a core concept in design and manufacturing. Since accurate measurements of the stresses is infeasible in most applications, numerical simulations, such as FEM, are commonly used to quantify the material states over complex geometries. However, solving the FEM formulation requires establishing the relationship between the stress and strain components for each material point. This relationship is untrivial and, as one can imagine, it varies widely between metals, rubber, and composites, to name but a few. In essence, a material's constitutive model, based on the previous state of the material including the applied stresses ($\boldsymbol{\sigma}_t$) and other internal variables, known as state variables ($\boldsymbol{q}_t$), attempts to update the two variables (i.e., stress and state variables) for the current time step given the current deformation ($\Delta\boldsymbol{\varepsilon}_{t+1}$), as depicted in **Figure 4.1**. Without loss of generality, we focus this discussion on elasto-plastic behavior as they occur in most metals.

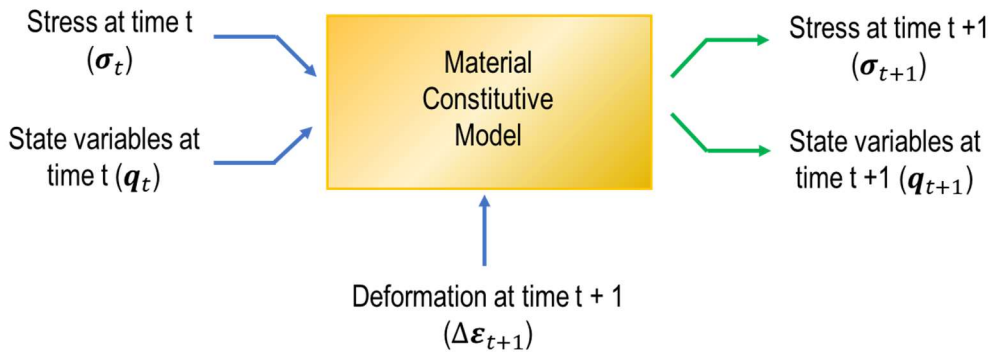


Figure 4.1. Calculation scheme for a material's constitutive model where the model receives the previous stress and state variables as well as the current deformation and outputs the updated stress and state variables at each time step.

$$\Delta\boldsymbol{\varepsilon} = \Delta\boldsymbol{\varepsilon}^e + \Delta\boldsymbol{\varepsilon}^p \tag{4.1}$$

$$\Delta\boldsymbol{\sigma} = \mathbb{C} : \Delta\boldsymbol{\varepsilon}^e \tag{4.2}$$

$$\Delta\boldsymbol{\varepsilon}^p = \Delta\lambda\boldsymbol{r}\,,\ \Delta\boldsymbol{q} = \Delta\lambda\boldsymbol{h} \tag{4.3}$$

$$\Delta\lambda\ \Phi(\boldsymbol{\sigma},\boldsymbol{q}) = 0 \tag{4.4}$$

In classical plasticity, the key to compute the material response is to correctly decompose the deformation into elastic and plastic components (Eq. 4.1), which is unknown at each time step. To solve for these components, a number of additional equations are required to account for the number of unknowns in the system. Some of the relationships stem from the definition of the variables; for example, the elastic strain increments are linearly correlated with stress increments through the known matrix of elastic stiffness ($\mathbb{C}$) according to the definition of the elastic strain (Eq. 4.2). Other equations are established based on theoretical assumptions (e.g., plastic flow rule

determining the evolution of the plastic strain, Eq. 4.3, and the definition of effective stress and strain) or phenomenological laws developed mainly based on experimental observations (e.g., yield criteria, $\Phi(\boldsymbol{\sigma}, \boldsymbol{q})$, determining when deformation is plastic and hardening laws defining the evolution of the yield surface). A summary of these equations that need to be solved simultaneously is provided by Eqs. 4.1 - 4.4, where $\Delta\lambda$ denotes the plastic multiplier increment, $\boldsymbol{r}$ is the normal direction to the yield surface at yield position, and $\boldsymbol{h}$ defines the evolution of the state variables, which is often determined by the hardening law.

To guarantee the consistency condition (Eq. 4.4) an iterative Newton-Raphson method is commonly used in practice, where one first assumes that all deformation increments are elastic and later solves for the current $\Delta\lambda$ iteratively until a predefined numerical accuracy is achieved.

The outstanding progress of the computational plasticity field to establish the stress-strain relationship stems from reducing 3-dimensional plastic behavior into so-called “effective” phenomenological laws that can be calibrated using a minimal number of experiments. This process introduces various assumptions including yield criteria formulation, flow rule, and effective stress-strain definitions. In addition, solving the plasticity system of equations becomes more computationally demanding and unstable as one tries to predict more complicated phenomena such as the Bauschinger effect (Armstrong et al. 1966, Leem et al. 2019), ratcheting (Chaboche 1991, Ohno et al. 1993), anisotropy (Hill 1990, Barlat et al. 1991), viscoplasticity (Chaboche 1989), permanent softening (Geng et al. 2002, Lee et al. 2007), and distortional hardening (François 2001, Feigenbaum et al. 2007, Barlat et al. 2011).

In this work, we aim to introduce a first-of-a-kind data-driven constitutive modeling approach and show that it can directly provide a closed-loop mapping between meterial deformation and stresses in their original 3-dimensional spaces. Our appraoch resolves the need for numerous theoritical assumptions introducing a more flexible and general constitutive model for unconventioal mateirals. Furthermore, as a data-driven model does not require iterative solutions, it opens a new avenue for drastic improvments in the efficiency of computational plasticity methods.

In this chapter, we first introduce the main steps in our approach to data-driven constitutive modeling in Section Chapter 4.2. Two case studies are elaborated upon in Section 4.3 and a detailed network analysis is presented in Section 4.4. Finally, the conclusions and future research directions are enumerated in Section 4.5.

4.2. Theoretical Approach to Data-Driven Constitutive Modeling

We follow a three-step framework to create data-driven constitutive models. First, samples of input space are extracted using a design of experiments technique. As the model solely relies on the data to interpret the constitutive relationships, it is crucial that the samples sufficiently represent the distribution of input conditions in which the data-driven model is expected to perform. Second, we create a database of material responses corresponding to the input samples. While in principle, any reliable source would suffice, we pursue computational analysis to create a database of ground-truth input-output pairs. Finally, we propose a novel machine learning approach to learn the material plasticity as it trains over thousands of material response samples captured in the database.

4.2.1 Design of Experiments in Input Space

To capture the behavior of a general constitutive model, we consider three key types of variables in our design input space: (i) microstructure descriptors (e.g., volume fraction, inclusion geometries), (ii) material properties for each microstructural phase (e.g., elastic moduli, yield behavior), and (iii) loading conditions (e.g., deformation range and patterns). For the temporally fix features (e.g., microstructure and material descriptors), a space-filling design of experiment technique is deployed to maximally encompass information on the entire valid range of variables. Here, we build an RVE database using a variant of the descriptor-based approach (Xu et al. 2014) as it enables establishing physically interpretable links between microstructural descriptors and material properties. First, we identify key microstructural descriptors that characterize the RVE and conduct design of experiments (DOE) with Sobol sequence technique. Then, we reconstruct the RVEs corresponding to the DOE points. Finally, we post-process the generated RVEs to extract more microstructural descriptors that are not used in stage one. In the case of fibrous composites where the fibers of an RVE are equally-sized and randomly dispersed within the matrix (an example is given in Section 4.3.2), we choose the following three descriptors to characterize the morphology: fiber volume (or area) fraction (v), fiber radius (r), and minimum allowable center-to-center distance between any two fibers (c). The first two descriptors are known to affect the material properties in fiber composites (Bessa et al. 2017). The third descriptor is used to set a minimum distance between any two fibers to avoid overlaps, facilitate FEA, and partially control the spatial distribution of fibers within the matrix. Given the 3D input space of $[v, r, c]$, we generate

a DOE of size 5,000 where the range of each parameter is selected sufficiently large (see **Table 4.1**) to reconstruct a wide range of RVEs.

Table 4.1. Parameter ranges for RVE reconstruction and load-path design (Mozaffar et al. 2019).

	v (%)	r (μm)	c (μm)	$E = [e_{11}, e_{22}, e_{12}]$
Min	5	3	8	$[-0.02, -0.02, -0.02$
Max	40	10	20	$[0.02, 0.02, 0.02]$

To reconstruct the RVE corresponding to the i^{th} DOE point (i.e., given $[v_i, r_i, c_i]$), we first randomly place $n_i = \frac{v_i L^2}{100 * \pi * r_i^2}$ fibers of radius r_i in a square RVE of side length $L = 200\ \mu m$. Then, we iteratively perturb the fiber locations until their spatial distribution satisfies c_i. It is noted that some combinations of $[v, r, c]$ might correspond to infeasible RVEs or our iterative perturbation might stop before c_i is satisfied. The 3D input space of $[v, r, c]$ along with the feasible DOE points are visualized in **Figure 4.2** where it can be observed that some regions of the $[v, r, c]$ space do not correspond to realizable RVEs. **Figure 4.3** also shows four sample RVEs for easier interpretation of microstructural differences. Note that the triplets $[v, r, c]$ cannot uniquely characterize a microstructure with randomly dispersed equally-sized fibers. Hence, we post-process the reconstructed RVEs to extract four more morphological features that quantify the spatial distribution of fibers. These features are the minimum, maximum, mean, and standard

deviation of nearest neighbor distances across the fibers: $\boldsymbol{nn} = [nn_1, nn_2, nn_3, nn_4]$. That is, for the i^{th} RVE, we calculate the nearest neighbor of all the fibers (center-to-center distance) and then calculate the abovementioned statistics. These seven non-temporal features (along with the deformation path) are employed in our deep learning task as inputs.

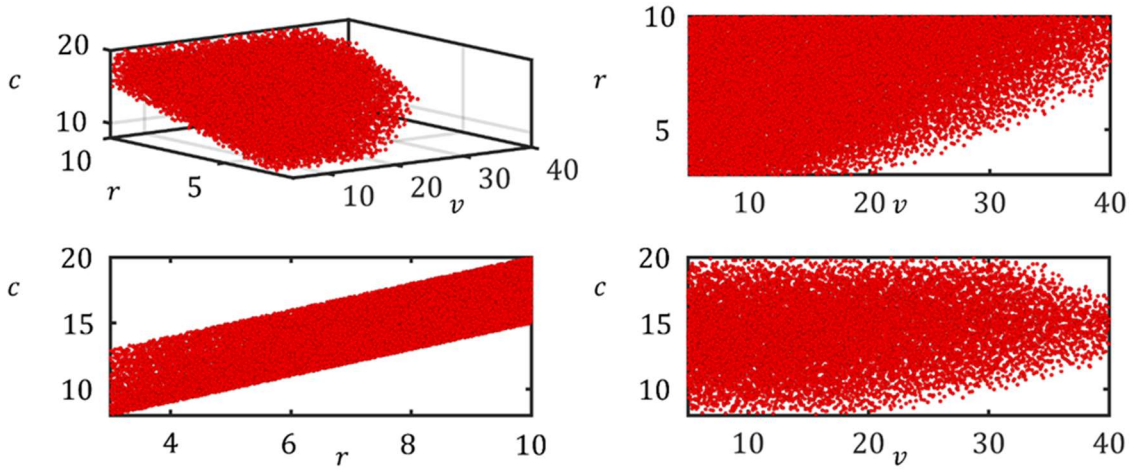


Figure 4.2. Design of experiments with 5,000 points in the 3D space of $[v, r, c]$. v is in percent while r and c are in μm (Mozaffar et al. 2019).

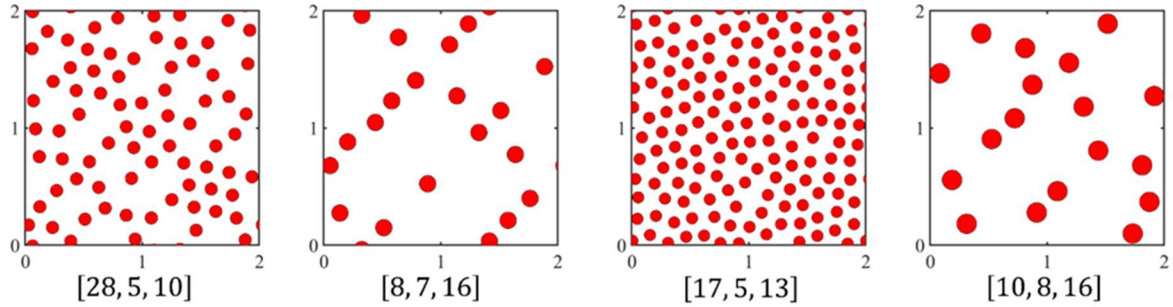


Figure 4.3. Four sample RVEs. Side lengths are all 200 μm and the triplet below each RVE corresponds to $[v, r, c]$. v is in percent while r and c are in μm (Mozaffar et al. 2019).

Sampling temporally varying features such as deformation path is more intricate as it involves generating a sequence of points (rather than a single point) for each sample path, see **Figure 4.4A**. To address this issue, we assume that any dynamic feature evolves to its end-state in n_{ts} time steps

of size Δt. From these time steps, we then choose n_{cp} equally spaced ones as control points and assign them random deformations which are uniformly drawn from the feature's range. Finally, we realize a path of dynamic deformations by connecting the feature values of these control points via an interpolator. We have considered two different interpolators: Gaussian process (GP) and polynomial regression, see **Figure 4.4B** and **C**. For multi-dimensional features such as strain, we generate paths along each dimension independently.

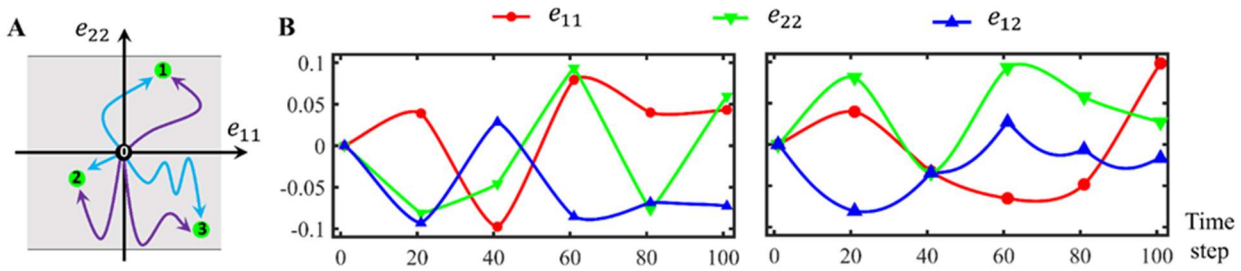


Figure 4.4. Sampling the temporally varying loads: **(A)** Three end-states are marked in the strain space spanned by e_{11} and e_{22} ($e_{12} = 0$ for clarity). For each end-state, two deformation paths that connect it to the origin are illustrated. The grey area indicates the range of each strain component. **(B)** Two examples indicating the temporal evolution of the three strain components that, collectively, determine the deformation path to an end-state. The markers on each path indicate the control points used in interpolation. Here, $n_{ts} = 100$, $n_{cp} = 6$, and the interpolator is a zero-mean GP with power exponential kernel. Paths in **(B)** are not related to **(A)** (Mozaffar et al. 2019).

4.2.2 Database Assembly

We simulate the behavior of the constructed RVEs under complex loading conditions using high-fidelity finite element analysis (FEA). A MATLAB code developed by Bessa et al. (Bessa et al. 2017) creates the scripts that interact with the finite element software ABAQUS® to preprocess, execute and postprocess all the simulations automatically for the two-dimensional RVEs using an

implicit static solver. The analysis begins with generating the material and boundary condition files. The MATLAB script parses the geometry descriptors for each RVE in the database and generates python scripts which creates the geometry, meshes it with a predetermined mesh size ($1.5\,\mu m$), and assigns the materials to corresponding sections. We apply periodic boundary conditions to the edges of RVEs as it better estimates the overall performance of the composite. Temporal strain components $e_{11}(t)$, $e_{22}(t)$, and $e_{12}(t)$ are defined as displacement boundary conditions on the RVEs. Next, The MATLAB script generates the simulation files and executes ABAQUS® using them. Finally, the outputs of each simulation is postprocessed to extract three engineering stresses $\sigma_{11}(t)$, $\sigma_{22}(t)$ and $\sigma_{12}(t)$ as well as the plastic energy $U_p(t)$. Note that the computational analysis is completely automated to produce large databases without manual input.

4.2.3 Machine Learning Approach

Once the input space is sampled and the corresponding output database is created, an RNN is fitted to learn the plasticity constitutive law by relating stresses and plastic energy to microstructure descriptors and loading conditions. RNNs describe plasticity as a map including time evolution of the different variables:

$$\bar{\sigma}_t = f(\bar{e}_{1:t}, m, p, t) \tag{4.5}$$

where $\boldsymbol{m}$ and $\boldsymbol{p}$ describe the microstructure descriptors and the properties of the phases, $\bar{\boldsymbol{e}}_{1:t}$ is the history of spatially averaged strains applied to the RVEs from the first to current (t) deformation increment, and $\bar{\boldsymbol{\sigma}}_t$ is the spatially averaged stress at the current deformation increment.

RNNs are an extension of neural networks designed to handle sequential data, i.e., they can learn events happening along different time sequences that can be captured with a different number of snapshots. RNNs use history-dependent hidden states $s_t(x_t, s_{t-1})$ to compute the outputs $o_t(x_t, s_t)$ which enables them to carry information of previous inputs for future predictions, where x_t is the input feature at increment t. This unique feature of RNNs combined with the flexibility of their model architecture has proven to be greatly beneficial on tasks such as machine translation, natural language processing and voice recognition, among others (Mozaffar et al. 2018, Young et al. 2018). Early formulations of RNNs suffer from a phenomenon known as vanishing/exploding gradients, first noticed in (Hochreiter 1991), which hinders the backpropagation-based training process of the networks for long sequences. Long Short-Term Memory (LSTM) (Hochreiter et al. 1997) was proposed to avoid vanishing gradients by using multiple data gate mechanisms which control the flow of storing or forgetting information in hidden states and outputs. Gated Recurrent Unit (GRU) (Cho et al. 2014) uses a similar concept as LSTM while using a simplified formulation. Although LSTMs and GRUs have shown to have close performance in many cases (Chung et al. 2014), GRUs formulation is less prone to overfitting and allows faster training due to the smaller number of trainable parameters. Stacking multiple RNN units enables the model to predict higher-level nonlinearities and interactions between features. A major challenge in predicting plasticity constitutive laws for material systems is that the model should effectively correlate temporal loading inputs with non-temporal RVE design features (e.g., volume fraction, fiber radius, or elastic moduli). Three variations of RNN architecture are considered to address this challenge as depicted in **Figure 4.5A** to **C**.

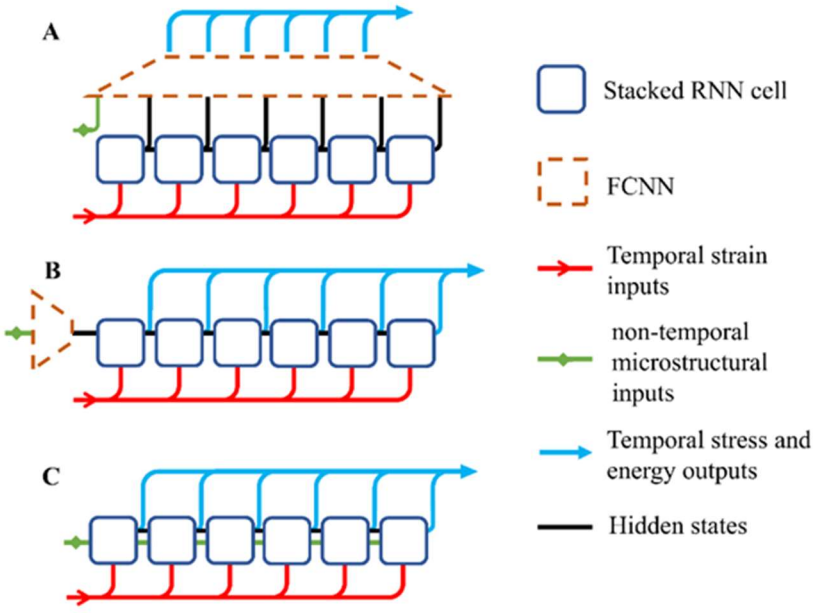


Figure 4.5. Variation of RNN architecture to encapsulate temporal and non-temporal inputs; **(A)** post-mixing non-temporal data through a dense network, **(B)** configuring non-temporal data as initial hidden state value through a dense network, or **(C)** establishing a secondary non-temporal hidden state in GRU formulation (Mozaffar et al. 2019).

First, non-temporal features can be merged with temporal RNN outputs using fully connected neural network (FCNN) layers to form a hybrid deep learning architecture (**Figure 4.5A**). While this approach is plausible for applications with fixed output length at the final time-step, its structure does not provide a natural fit for constitutive law discovery of material systems as it restrains the temporal prediction of the model to a fixed length and offers limited correlation between temporal and non-temporal features.

As a second approach, the non-temporal features can be integrated into the RNN formulation as the initial value for hidden states (**Figure 4.5B**). As the dimensionality of non-temporal inputs and hidden states are often different, a dense network can be used to perform this mapping. Although this approach has shown promising results for image processing tasks (Karpathy et al. 2015, Vinyals et al. 2015), it is not the most effective architecture for constitute laws because all

information in the hidden states are subject to change as they pass through GRU cells. That is, non-temporal inputs can get corrupted with other hidden features which makes it excessively difficult for GRU cells to access them at downstream time steps.

We propose a GRU formulation in which a secondary hidden state is used to carry non-temporal inputs through the GRU cells which allows nonlinear correlations between temporal and non-temporal input features while providing each GRU cell direct access to temporal inputs, non-temporal inputs, and history-dependent hidden states (**Figure 4.5C**). The altered formulation of a GRU unit (Cho et al. 2014) used in this work is as follows:

$$r_t = sig\big(W_r.\big[h_{t-1}, x_t, h_f\big] + b_r\big) \tag{2}$$

$$z_t = sig\big(W_z.\big[h_{t-1}, x_t, h_f\big] + b_z\big) \tag{3}$$

$$\hat{h}_t = \tanh\left(W.\big[r_t \times h_{t-1}, x_t, h_f\big] + b\right) \tag{4}$$

$$o_t = h_t = (1 - z_t) \times h_{t-1} + z_t \times \hat{h}_t \tag{5}$$

where sig is the sigmoid function. The reset gate (r_t) determines the combination of inputs (x_t), previous hidden states (h_{t-1}), and the secondary hidden state (h_f) to build a candidate hidden state ($\hat{h}_t$), and the update gate (z_t) controls the influence of candidate hidden state to the unit output (o_t), which is also used as the new hidden state (h_t). Using this variation of GRU formulation non-temporal features are protected from corruption by r_t, the reset gate. Although this approach adds additional weights and biases to the GRU, increasing model complexity and training time, it enables GRU cells to capture intricate interactions between input features throughout the entire temporal states.

The proposed neural network models based on the architectures demonstrated in **Figure 4.5** is developed using the Keras library (Chollet 2015). The model includes RNN cells to detect history-dependent features and combines with one or more time-distributed dense layers (Keras) to transform high-dimensional outputs of RNN cells into the desired 4 outputs (three engineering stresses and plastic energy) of the RVEs over time. A cost function of the mean absolute error between the output values in the developed database and the predictions are defined and the training process is performed using Adam optimization method (Kingma et al. 2014). The inputs (i.e., microstructure descriptors and deformation paths) and outputs (i.e., stresses and energies) of the database are normalized to a range between 0 and 1 to expedite the training process by reducing narrow valleys in trainable parameter space.

We use the scaled mean absolute error (SMAE) metric to evaluate the results of the model. This enables us to have a fair error measure with the same dimensions as the original outputs which is independent of the data magnitude as it is scaled over the range of data. Although the plastic energy should not decrease over time according to the second law of thermodynamics, it is noteworthy that small decreases in the plastic energy can be seen even in the FEA results. However, to quantify if the plastic energy is predicted with an acceptable range of error, we define a second metric as the accumulated plastic energy deviation from monotonic increase averaged over test set samples, which is named as scaled mean plastic energy decrease (SMPED).

The developed model is trained on 80% of the database while the rest is used as test set for validation. The train set SMAE decreases consistently as we increase the number of RNN layers, RNN units, or time-distributed layers due to the computational complexity the models. However, using excessively complex configurations causes the model to overfit the train set samples. Also,

our results indicate that adding extra time-distributed layers adversely affect the test set SMPED, which is another form of undesirable overfitting.

4.3. Plasticity Modeling Results

Our analysis shows the architecture depicted in **Figure 4.5C** leads to the best results (details presented in Section 4.4.1). Based on this architecture, we consider two examples to illustrate the capabilities of sequence learning in finding plasticity constitutive laws. The first example focuses on a single RVE with a curved inclusion which imposes significant distortional hardening. The second example pertains to a class of RVEs with distributed circular inclusions. This later example demonstrates that our approach allows to systematically formulate microstructural information in plasticity constitutive laws while classic constitutive modeling lacks such generality.

4.3.1 Case I: RVE with Curved Inclusion

Even simple heterogeneous materials can undergo complex history-dependent plastic deformation. An illustrative example is devised by considering a periodic microstructure of a material composed of distorted elliptical fillers as shown in **Figure 4.6A**. Without loss of generality, consider the matrix material to be an aluminum alloy (AA6061) described by a von Mises isotropic hardening model, and the rubber fillers described by an Arruda–Boyce hyperelastic material model (Arruda et al. 1993). The material properties of the models are provided in **Table 4.2**.

Table 4.2. Matrix and fiber material properties for case 1(Mozaffar et al. 2019).

Matrix density ($\boldsymbol{\rho_m}$)	$\mathbf{2.7\ g/cm3}$
Matrix Young's Modulus ($\boldsymbol{E_m}$)	68.9 GPa
Matrix Poisson's ratio ($\boldsymbol{\nu_m}$)	0.33
Matrix Voce isotropic hardening $\boldsymbol{\sigma_{iso,m} = B - (B - A)\exp(-n\epsilon)}$	A = 74.4 MPa B = 144.98 MPa n = 7.25
Fiber density ($\boldsymbol{\rho_f}$)	1.0 g/cm3
Fiber Arruda shear coefficient ($\boldsymbol{\mu_f}$)	166 MPa
Fiber Arruda locking stretch ($\boldsymbol{\lambda_f}$)	2.8

The combination of a ductile matrix and a hyperelastic filler with non-symmetric geometry results in a compound elasto-plastic behavior where the matrix deforms plastically while the fillers can store a significant amount of elastic energy. Although the constituents are isotropic, the filler geometry induces significant anisotropic behavior and distorts the yield surface obtained for the macroscopic heterogeneous material as it undergoes different deformation paths. The macroscopic constitutive behavior of the heterogeneous material results from relating the applied average strain components $e_{11}(t)$, $e_{22}(t)$, and $e_{12}(t)$ at time step *t* to average stresses $\sigma_{11}(t)$, $\sigma_{22}(t)$, $\sigma_{12}(t)$ and plastic energy $U_p(t)$. The average stresses are computed via homogenization of the RVE, while the average strains are converted into a periodic boundary value problem (Bessa et al. 2017). Note that each stress state of the RVE depends on the deformation path towards getting there.

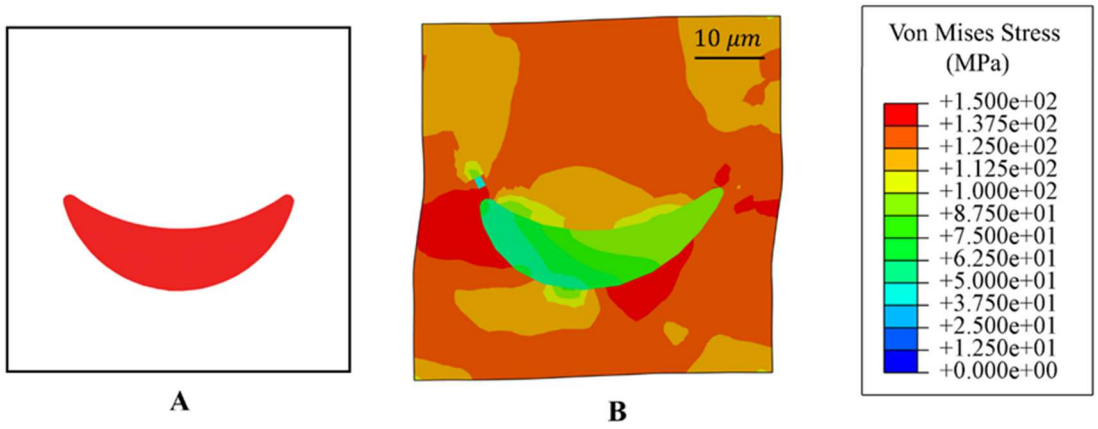


Figure 4.6. (A) Undeformed configuration of RVE with curved ellipse and **(B)** von Mises stress contour of the deformed periodic RVE in MPa for illustrative case 1(Mozaffar et al. 2019).

Once the RVE in **Figure 4.6A** is simulated via FEA under 15,000 different deformation paths, a database with the average stresses and plastic energy for each deformation path is generated (100 deformation states per deformation path). Using this dataset (created in 2 weeks using 48 cores of a high performance computing cluster), we train an RNN with an architecture illustrated in **Figure 4.5C** whose parameters are trained on 80% of the database. We assess the predictive power and data sufficiency by the unseen 20% portion of data. We use a scaled mean absolute error (SMAE) metric to evaluate the results of the model and a second metric for the plastic energy, called scaled mean plastic energy decrease (SMPED), to quantify a possible decrease in plastic energy as this should not happen since plasticity is an irreversible form of deformation (second law of thermodynamics). The designed model consists of 3 stacked layers of 500 RNN units followed by a single time-distributed dense layer, which corresponds to around 3 million trainable parameters. Finally, a leaky rectified linear unit (Xu et al. 2015) activation functions is used to impose nonlinearity into the neural network model. We trained the model for 200 epochs which resulted

in 0.00281 training set SMAE, 0.00355 test set SMAE, 0.000181 training set SMPED and 0.000186 test set SMPED.

Figure 4.7 presents results for two different validation deformation paths: one deformation path of the test set that was not used in training (**Figure 4.7A** to **C**); and a linear unidirectional loading and unloading deformation path for validation purposes that is not present in either of those sets (**Figure 4.7D** to **F**). As observed in **Figure 4.7B** and **Figure 4.7C**, the RNN is predictive along deformation paths of the test set (unseen data) for both quantities of interest. In addition, we also created validation cases of deformation paths that were not present in either the training or test sets. **Figure 4.7D** shows a linear unidirectional strain path which stretches the RVE in the e_{11} direction to 0.1 engineering strain and then in the opposite direction to -0.1 engineering strain while e_{22} and e_{12} are kept at zero. Note that the training set does not include any linear strain path, rather it is constructed via a Gaussian process which results in fluctuating paths. **Figure 4.7E** and **F** demonstrate that our RNN model is also able to predict these average stress states and plastic energy. We note that the model can also effectively predict the Poisson's ratio effect between σ_{11} and σ_{22}. The Supporting Information includes details on the RNN architecture analysis, as well as convergence studies and error metrics used to assess the predictions. In addition, accuracy of the predictions can be easily improved by increasing the size of the training dataset, but these results indicate that the model can generalize well to loading conditions outside this dataset.

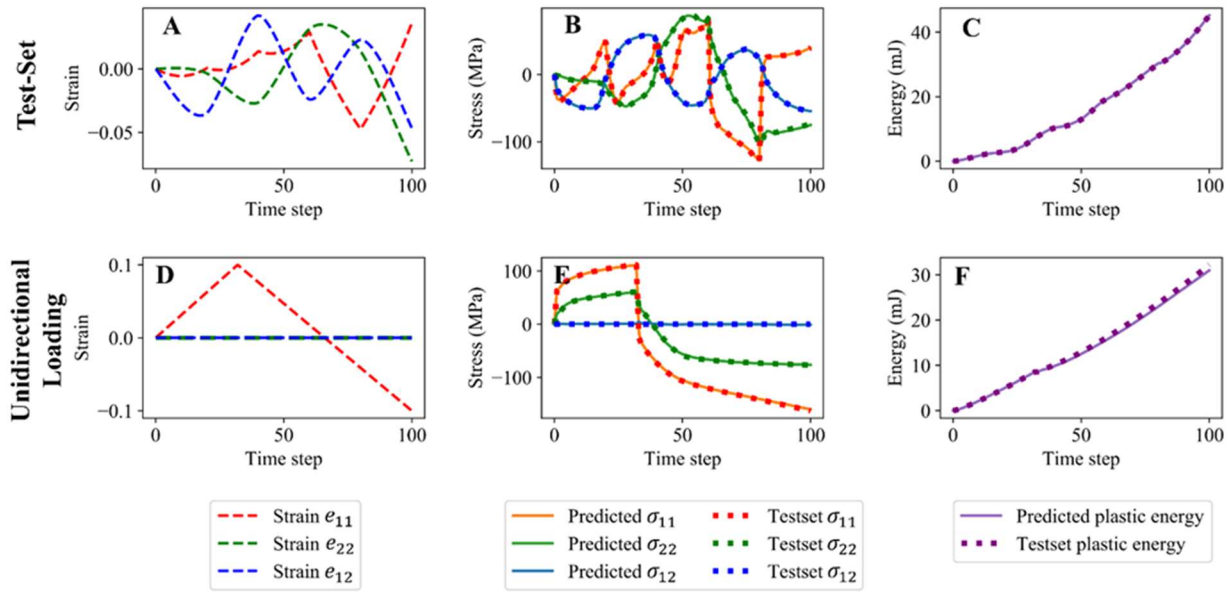


Figure 4.7. Evaluation results for the trained model in case 1. The top row demonstrates **(A)** the applied average strains, **(B)** the predicted and database average stresses and **(C)** the predicted and database plastic energies for a test set sample (unseen in training process). The bottom row depicts the **(D)** average strains, **(E)** average stresses, and **(F)** plastic energies for the unidirectional loading test (Mozaffar et al. 2019).

We further explore the predictive capabilities of our model by evaluating the yield surface evolution as the RVE experiences different loading conditions. **Figure 4.8** shows the yield surface at the onset of plasticity in purple, and the yield surface obtained at the end of three deformation paths. We define plastic deformation to start when the plastic energy increases by the threshold of 1 mJ. Alternatively, the average equivalent plastic strain of the matrix could be used. We construct the yield surface by loading the RVE from its current stress state to 40 deformation paths in different directions (details are provided in Appendix B). The principal stresses of the RVE (i.e., eigenvalues of the stress tensor) are calculated when the RVE is plastically deformed above the mentioned threshold and the stress state is stored in order to reconstruct the yield surface. Details of yield surface construction are provided in Supporting Information. Each of the four yield surfaces shown in **Figure 4.8** includes the result obtained directly from FEA in dotted lines, the

prediction from RNN in solid lines and the applied deformation before yield surface construction with respective colors. The yield surface at the onset of plasticity (purple) resembles the elliptical shape of von Mises yield surface which is in-line with the behavior of the matrix. However, as seen at the end of the three deformation paths, the yield surface is distorted, shrinks/expands and rotates for different deformation histories. Remarkably, the RNN can track the complete yielding behavior accurately, including the anisotropic and distortional yield behavior. Therefore, using sequence learning for finding plasticity laws of general RVEs is demonstrated to be possible, laying the foundations for a new modeling route for plasticity that learns the compound correlation of yield surface and hardening laws without any explicit guide or definition of classical plasticity terms such as effective plastic strain and effective stress.

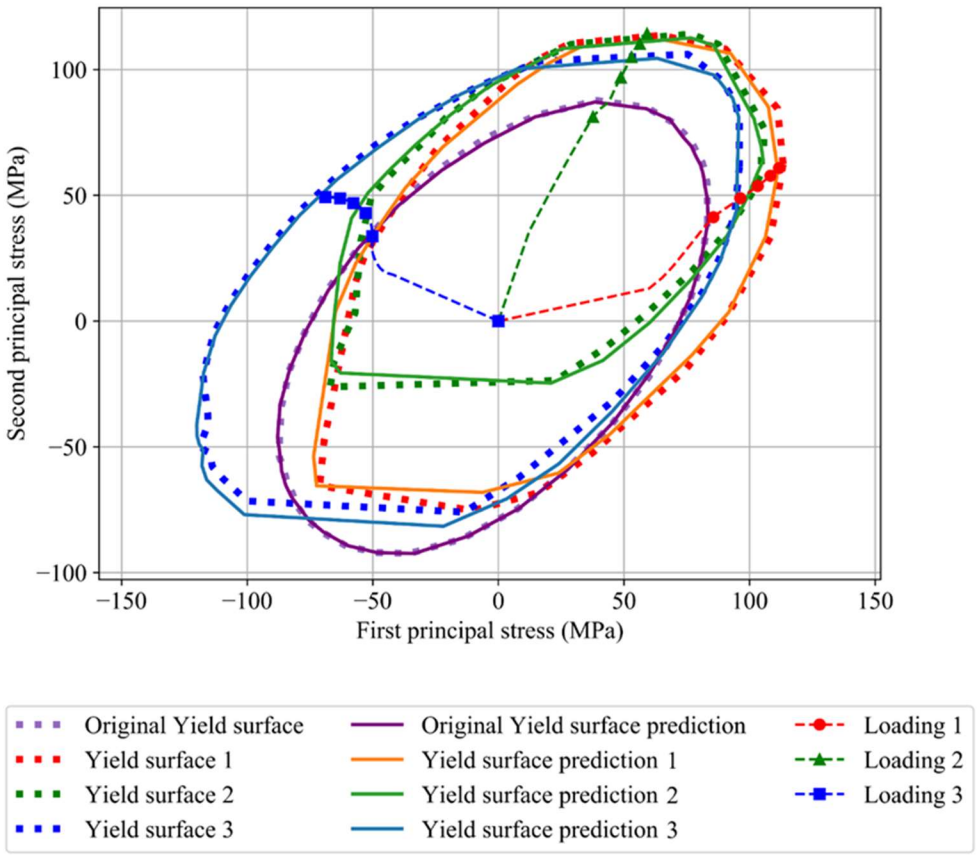


Figure 4.8. Yield surface evolution under different deformation conditions and paths. FEA-based and RNN predicted yield surfaces are demonstrated in dotted lines and solid lines, respectively, at the end of three different deformation paths as compared to the original yield surface (purple) (Mozaffar et al. 2019).

4.3.2 Case II: RVE with Distributed Circular Inclusions

To explore the flexibility of the proposed framework, we study a second case which differs from the previous case in two major aspects. First, we consider periodic microstructures with distributions of circular fibers, where each sample in the database varies in terms of their fiber volume (area) fraction, fiber radius, and distance. We consider epoxy with combined isotropic and kinematic hardening and carbon with elastic behavior as the matrix and fiber material models, respectively. Details of material properties are provided in **Table 4.3**. Second, we considered

polynomial deformation paths with maximum strain of 2% in this case. These two design choices give us a compound elasto-plastic behavior for the RVE as the matrix deforms plastically while fibers can store a significant amount of elastic energy, whereas the behavior of the first case was mostly dominated by plastic deformation.

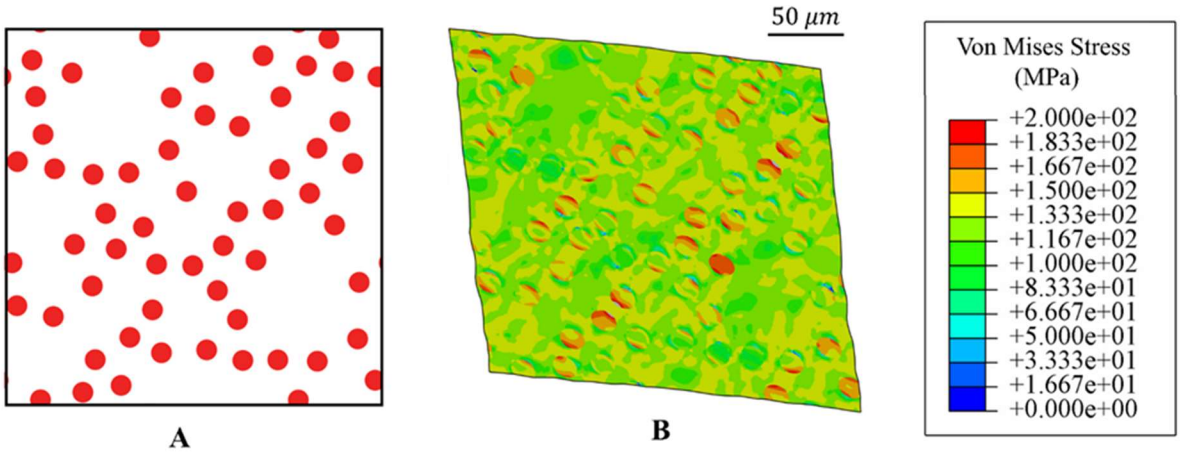


Figure 4.9. (A) Undeformed configuration and **(B)** von Mises stress contour of a deformed sample of periodic RVE with distributed circular fillers in MPa for illustrative case 2 (Mozaffar et al. 2019).

Table 4.3. Matrix and fiber material properties for case 2 (Mozaffar et al. 2019).

Matrix Young's Modulus ($\boldsymbol{E_m}$)	**4.07 GPa**
Matrix Poisson's ratio ($\boldsymbol{\nu_m}$)	0.34
Matrix Voce isotropic hardening $\boldsymbol{\sigma_{iso,m} = B - (B - A)\exp(-n\epsilon)}$	A = 16.44 MPa B = 77.5 MPa n = 746.2
Matrix kinematic back-stresses ($\boldsymbol{n_{kin,m}}$)	2

Fiber density ($\boldsymbol{\rho_f}$)	1.8 g/cm3
Fiber Young's Modulus ($\boldsymbol{E_f}$)	15 GPa
Fiber Poisson's ratio ($\boldsymbol{\nu_f}$)	0.2

The inputs to the RNN model are the non-temporal microstructure descriptors as well as the temporal deformation paths and the outputs are temporal stresses and plastic energy over 100 increments for each RVE. Similar to the previous case, two validation tests are presented and neither of which is used in the training process. A database with 5000 samples is used in this case, 80% of which used for training. A model with architecture shown in **Figure 4.5C** and similar configuration as the previous case is trained for 500 epochs resulting in 0.00242 and 0.00257 SMAE on training set and test set correspondingly, while the model error for the SMPED metric is 0.00104 on both datasets. **Figure 4.10A** to **C** demonstrate the comparison of our model prediction with the ground truth FEA for a polynomial deformation path outside of the training samples. The results indicate that the model can accurately predict both stress and energy responses of the RVE. The results of linear unidirectional loading test (depicted in **Figure 4.10D** to **F**) show that the model is accurately predictive for stress and most regions of plastic energy. The small noise in the plastic energy prediction is caused by the sharp change in deformation path, which was not observed when the deformation paths were sampled with GP regression due to its smoothness.

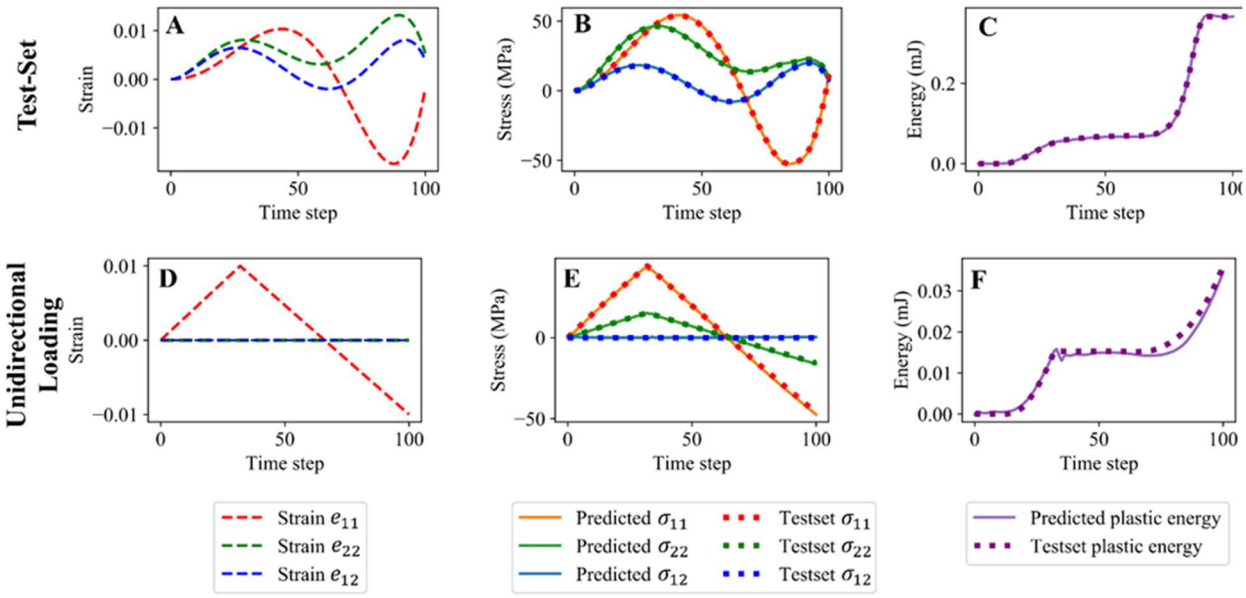


Figure 4.10. Evaluation results for the trained model in case 2. The top row demonstrates the **(A)** the applied average strains, **(B)** the predicted and database average stresses and **(C)** the predicted and database plastic energies for a test set sample (unseen in training process). The bottom row depicts the **(D)** average strains, **(E)** average stresses, and **(F)** plastic energies for the unidirectional loading test (Mozaffar et al. 2019).

4.4. Network Analysis

4.4.1. RNN architecture analysis

The three RNN architectures introduced in **Figure 4.5** to combine temporal and non-temporal features are extensively tested and their training result are presented in **Figure 4.11** and **Table 4.4** after 500 epochs of training. The hybrid architecture, which combines temporal GRU outputs with non-temporal FCNN features (**Figure 4.5A**) cannot achieve accurate prediction on training set and suffers from extreme overfitting. While the architecture with hidden state initialization (**Figure 4.5B**) performs moderately, the proposed architecture with a secondary hidden state (**Figure 4.5C**) achieves significantly better accuracy consistently across different epochs and metrics.

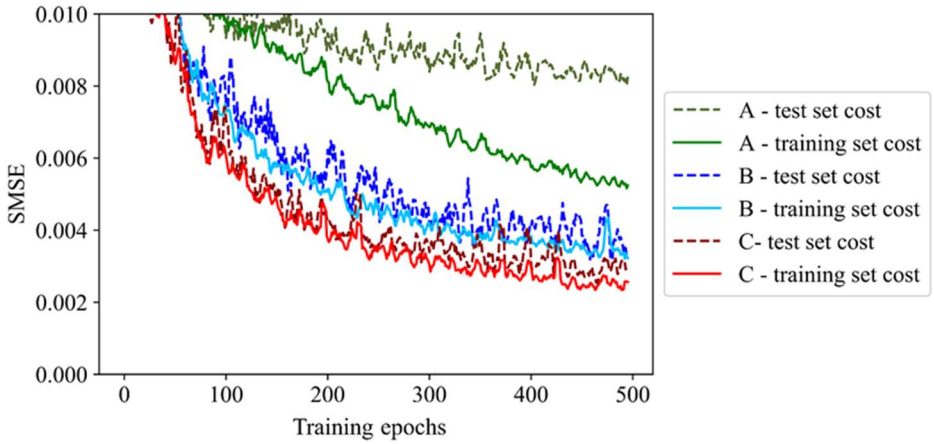


Figure 4.11. Cost function evolution as a function of training epochs for three different RNN architectures (Mozaffar et al. 2019).

Table 4.4. Metrics comparison between trained RNN architectures after 500 epochs of training (Mozaffar et al. 2019).

RNN architecture	Train set SMAE	Test set SMAE	Train set SMPED	Test set SMPED
Configuration A – hybrid mix	0.00557	0.00888	0.00233	0.00243
Configuration B – hidden state initialization	0.00345	0.00367	0.00130	0.00144
Configuration C – secondary hidden state	0.00242	0.00256	0.00104	0.00104

4.4.2. RNN hyperparameter tests

The hyperparameters and configurations of the presented RNN models are studied and optimized in this work. This analysis includes but is not limited to activation functions, optimization algorithms, cost functions, dropout layers, normalization process, and addition of time-series dense layers. **Figure 4.12A** depicts the results achieved by varying number of neurons in each GRU cells. It can be seen that 100 neurons cannot provide enough computational complexity to the model. While the model with 1000 neurons result in lower SAME on training set compared to the model with 500 neurons, the models perform closely on the test set. Considering that the model with 1000 neurons require more computational resources and training time and overfits on the training set, RNNs with 500 neurons are used in this work. Similarly, **Figure 4.12B** suggests that a model with 3 layers of stacked GRU layers achieves the best result with required least computational resources compared to models with 1 or 5 layers.

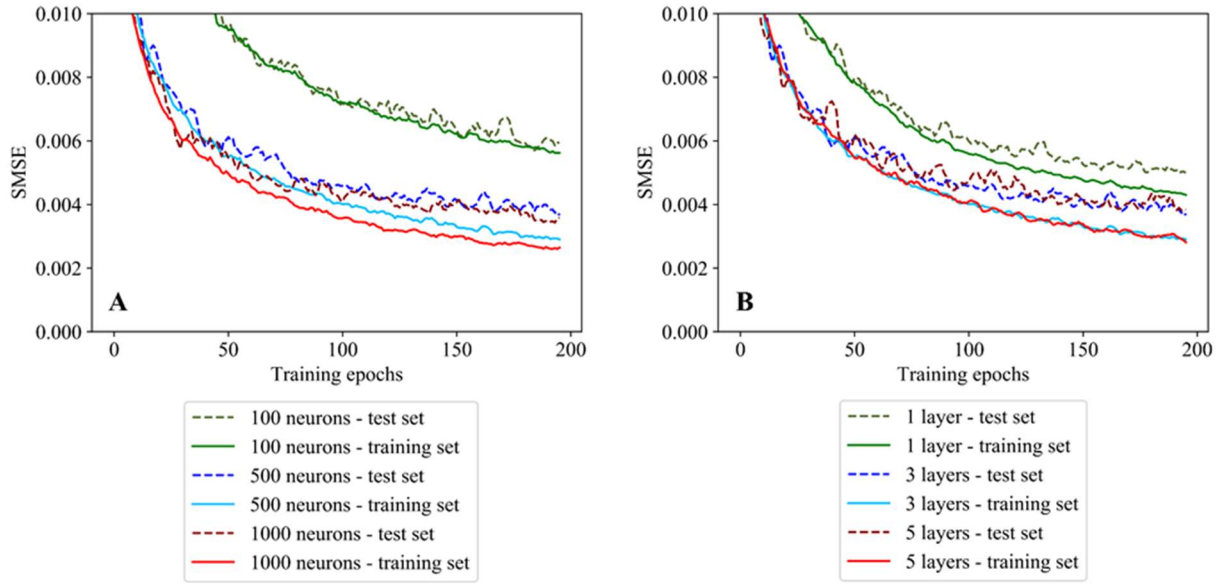


Figure 4.12. Hyperparameter analysis of the RNN model over 200 epochs of training for **(A)** number of neurons in GRU cells and **(B)** number of stacked GRU layers (Mozaffar et al. 2019).

4.4.3 Performance of proposed RNN architecture

We analyze the performance of the model with different sizes of training set to study the required database for achieving certain error metrics, which is demonstrated in **Figure 4.13**. As we increase the size of the training set, the model with 3 layers of 500 neurons performs better in both training set and test set; however, larger databases lead to an expected improvement of performance. Ultimately, the require size of database is dictated by the complexity of the behavior of the RVE and the required accuracy. In this work, we demonstrate that one can achieve predictive deep learning models for advanced plasticity behavior with databases that are computationally (or experimentally) built in a feasible time frame.

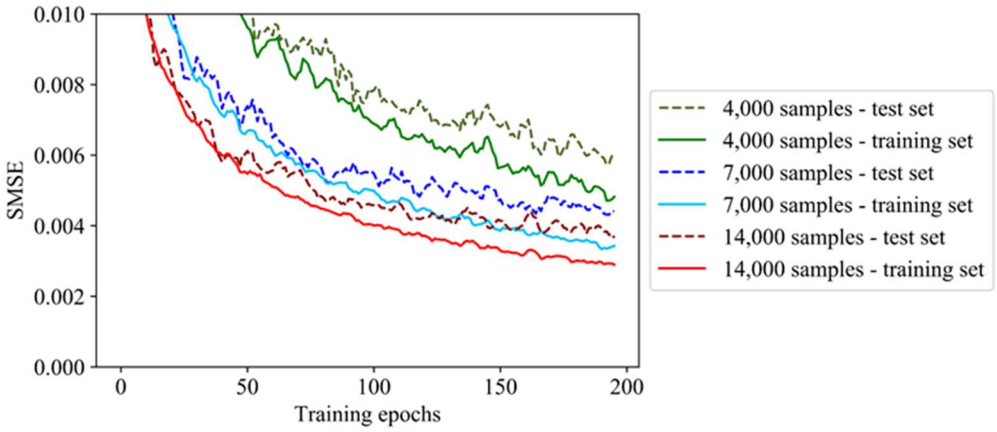


Figure 4.13. Convergence test for the RNN over 200 epochs of training (Mozaffar et al. 2019).

Note that once trained, our data-driven constitutive model performs far faster than the finite element method. As an example, the developed data-driven model predicts the behavior of one RVE in the second case study in 0.108 seconds on a Nvidia Titan black GPU while it takes 7.48 minutes on four cores of Intel Xeon CPU E5-2687 for the finite element method to complete the simulation. While the exact number highly depends on the hardware and simulated physics, it can be confidently stated that the data-driven approach offers orders of magnitude faster evaluation. This has important implications on multi-scale simulations where the constitutive laws at each point of the macro-scale material can be given by RNN models, instead of expensive RVE analyses. Furthermore, we note that the two approaches scale differently, given the type of hardware they require and application. For instance, calculating the response of 100 different RVE cases via the data-driven approach using the same hardware takes only 0.547 seconds, which is due to the batch processing capability of GPUs. Finite element methods, on the other hand, scale by distributing sub-domains over multiple CPUs to obtain performance gains through parallel

computing. These gains often saturate due to the communication overhead between processing units.

4.5. Conclusions and Future Works

In this work, we show a first-of-a-kind data-driven approach to constitutive modeling that enables capturing the complex behavior of general materials including elasto-plastic deformation, energy absorption, and yield surface evolution. Our results indicate that the trained model can comfortably reach under 0.5% SMAE error, while being fast to evaluate (a fraction of a second) because there is no need for iterative solution schemes such as the Newton-Raphson, typical in classical plasticity. While we showcased this idea in heterogenous material settings, the same concept applies for homogenous materials as well. As in (Gorji et al. 2020), we used a similar RNN-based constitutive model to accurately capture the challenging behavior of metal alloys under forming processes such as the Bauschinger effect and hardening stagnation.

A natural next step for this research path is to implement data-driven models inside FEM packages and investigate their potential in different applications and optimize their performance in different hardware settings (e.g., parallel execution on CPU, GPU, and TPUs). Since the RNN formulation is end-to-end differentiable, it enables efficient computation of consistent tangents (the partial derivative of stresses with respect to strains). Therefore, implicit FEM solutions, which require a consistent tangent, seems most compatible with our proposed data-driven model as it simultaneously solves the constitutive relationship for all elements in FEM simulations and guarantees convergence. For explicit solutions, however, our preliminary experimentations shows that incremental stress propagation between elements can generate severe instability for the

simulation. The implementation of data-driven constitutive model for explicit solutions in ABAQUS is demonstrated in **Figure 4.14**. These results do not show instability for single element cases. The fully connected neural network stays stable for multi-element tests despite having larger prediction error; however, the recurrent network grows unstable. I hypothesize this issue can be mitigated by using better temporal discretization schemes or imposing restrictions on the recurrent state characteristics. To best of our knowledge, this is an unaddressed gap in current state of data-driven constitutive modeling and can be a key topic in the future research.

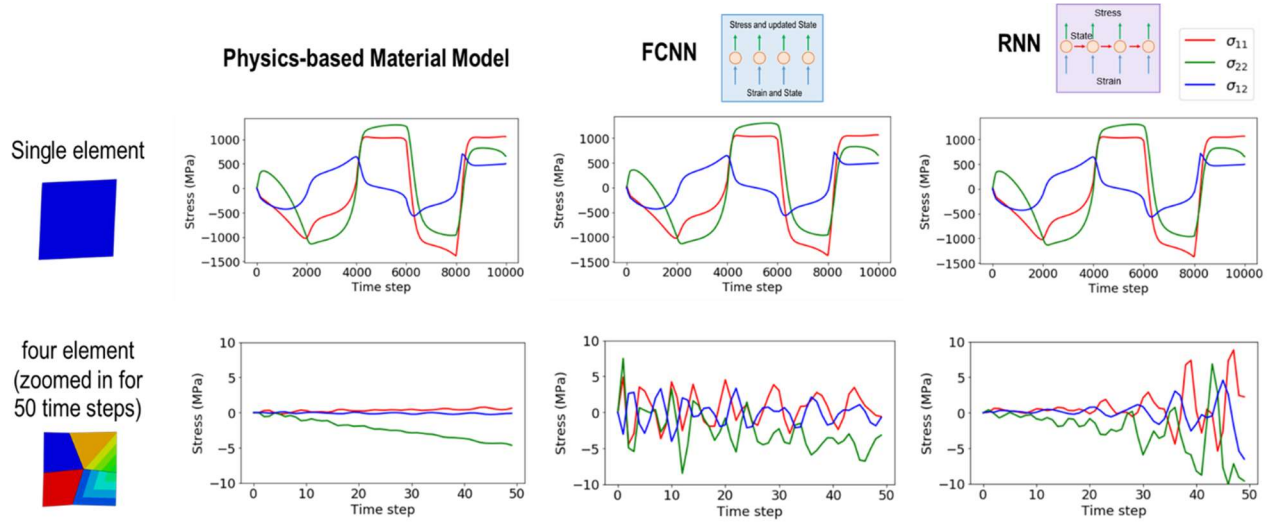


Figure 4.14. Demonstration of instability in RNN-FEM implmentation for multi-element simulaitons in ABAQUS.

CHAPTER 5

Toolpath Design for Additive Manufacturing using Deep Reinforcement Learning

5.1. Introduction

The performance of metal-based AM is currently hampered by the lack of robust design and prediction tools. Industrial AM practices often need series of trials and errors to produce parts to ensure that the geometric and mechanical requirements are satisfied. This is because AM involves multiple physics spanning over length scales that are orders of magnitude different. Therefore, AM modeling involves computationally expensive multi-scale methods with significant uncertainties in the process, which subsequently makes optimization-based design methods infeasible in the space. While the influence of some process parameters such as laser power, powder parameters and scan speed on the microstructure and final properties of the AM build are extensively studied in the literature, the influence of toolpath strategies and, more importantly, approaches for toolpath design yet to be thoroughly investigated. Designing toolpaths is particularly a challenging task as the large number of possibilities and the high-dimensional nature of the problem exacerbates optimization conditions.

In this chapter, we aim to address the toolpath design task by introducing a reinforcement learning (RL) platform that dynamically learns toolpath strategies to build any geometry with optimized performance. To this end, we investigate prominent model-free and model-based reinforcement learning methods to design AM toolpaths in two design cases.

In what follows, we discuss the importance of toolpath design methods and current practices in Section 5.1.1. We establish the fundamental formulation and definition of RL in Section 5.1.2. Later in Section 5.2 we provide details of our envisioned RL framework for toolpath design, introducing the environment and analysis cases in Section 5.2.1 and 5.2.2 respectively. We discuss our methodology and results for model-free methods in Section 5.3. Next, the implementation details and results of a model-based approach is presented in Section 5.4. Finally, we conclude this chapter by discussing the impact of our findings and future directions in Section 5.5.

5.1.1 Introduction to Toolpath Design in Additive Manufacturing

The process of toolpath generation usually starts from a CAD geometry. A slicing algorithm produces parallel sections of the CAD at different heights corresponding to deposition layer heights. Later, the area encapsulated by the boundaries of each section is filled with 2D patterns, often in the form of boundary contour passes or raster patterns. While planar coverage path planning (CPP) can be applied to toolpath design (Weiss-Cohen et al. 2008, Zuo et al. 2010, Zhou et al. 2012, Chaari et al. 2014, Zhou et al. 2014, Pratama et al. 2015), current literature suggests that these methods do not scale to large input spaces common in AM processes due to NP-complete complexity of the solution. Furthermore, cost structure design is an untrivial aspect of CPP solutions which greatly limits the potential of this methodology to purposefully design the mechanical behavior of the parts.

There are several publications that show that the choice of the toolpath significantly influences many aspects of metal-based AM. Steuben et al. (Steuben et al. 2016) considered three different toolpath patterns for building a part using a fused deposition modeling (FDM) process and

demonstrated that the pattern has a significant effect on the ultimate stress and elastic modulus of the build. Akram et al. (Akram et al. 2018) formulated a microstructure model using a Cellular Automata (CA) and demonstrated a strong correlation between the toolpath pattern (i.e., unidirectional and bi-directional) and the grain orientations. In (Bhardwaj et al. 2018), the authors considered bi-directional and cross-directional toolpath strategies to manufacture cubic parts with a Direct Metal Laser Sintering (DMLS) process and studied the surface finish, residual stress and mechanical properties of the parts. Their study indicates that the parts built with cross-directional strategy display better mechanical properties, which is due to their desirable structure of columnar cells. Similarly, the experimental study, done by Sebelle et al. (Sabelle et al. 2018), concluded that even the angle of a parallel toolpath greatly influences properties such as porosity, ultimate tensile and thickness in the Selective Laser Sintering (SLS) process.

From the above-mentioned research, it can be evidently seen that the choice of the toolpath is important as it opens a new avenue to customize material behavior. However, existing research does not offer a robust solution for the analysis of this influence nor tools to prudently design toolpaths. In this work, we present a novel way to represent the toolpath and learn design strategies that lead to optimal performance.

5.1.2 Introduction to Reinforcement Learning

Toolpath design is proposed to be modeled as a RL problem in which an agent learns to design optimal toolpaths as it dynamically interacts and collects data from an AM environment. In the RL schema, the agent is responsible for determining the actions at each time step (a_t), which influences the environment causing it to move from its current state (s_t) to a new state (s_{t+1}) and

generates a reward feedback (r_t) for the agent. Here, "state" refers to the representation of the environment that is visible to the agent. The agent learns to maximize the long-term rewards that it receives in its lifespan by attempting more of the strategies that lead to the most favorable rewards.

Most modern RL algorithms can be categorized into three main classes:

1. Policy optimization methods parameterize the policy, $\pi_\theta(a|s)$, and optimize θ to maximize the expected reward. Policy gradient methods estimate the gradient using the policy gradient theorem (Sutton et al. 2018) while evolutionary methods, such as cross-entropy methods (Salimans et al. 2017), perform the optimization without gradient estimation. Policy optimization methods suffer from poor sample efficiency, requiring millions of samples. Furthermore, for most policy optimization algorithms, all samples should be generated using the agent's policy at each training step, which exacerbates the sample efficiency of these methods as historic data cannot be used in the training process.

2. Value function optimization methods (also called Q-learning) do not optimize the policy directly. Rather, they aim to find the optimal action-value function $Q^*(s,a)$, as defined in Eq. 5.1, to represent the maximum discounted reward the agent can collect from any state. Following the actions that lead to maximum optimal action-values, $a^* = argmax_a Q^*(s,a)$, guides the agent to maximize its reward.

$$Q^*(s,a) = max_\pi \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1}) | \pi, s_0 = s\,, a_0 = a\right] \tag{5.1}$$

In Eq. 5.1, π represents the policy, H represents the environment horizon and γ is the discount factor—a positive value smaller than one. Discounting the future rewards encourages the agent to collect immediate rewards, bounds the accumulated rewards, and reduces the variance of the expectation since we often are progressively more uncertain about future rewards. The action-value function can be obtained using the Bellman equation (Eq. 5.2) with guaranteed convergence in tabular cases. By exploiting the self-consistency of the problem structure through the Bellman equation, action-value function optimization methods can learn the optimal action-value function and implicitly determine the policy with fewer samples. However, Q-learning lacks the stability of policy optimization methods.

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \, max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \tag{5.2}$$

3. Model-based RL algorithms, unlike the first two categories (known as model-free RL), attempt to learn an explicit model of the underlying dynamics of the environment. The model is further used for look-ahead planning or as a virtual sample generator. This class of solutions can offer a great sample efficiency with orders of magnitude less required data in published researches (Chebotar et al. 2017, Finn et al. 2017). However, the quality of the RL agent heavily depends on the accuracy of the dynamics model. Therefore, while there are many successful examples of this approach for robotics and games with perfect environments such as chess, many state-of-the-art algorithms in this class fail in high-dimensional spaces (e.g., pixel-level visual inputs) and uncertain environments.

Note that these categories are not mutually exclusive. In fact, many of the successful existing examples in the literature use a combination of these approaches. Most famously, actor-critic methods simultaneously parametrize and train both policy and value functions. For example, A2C (Mnih et al. 2016) follows the policy gradient theorem while using the value function to reduce the variance of gradient estimation and provides a stable solution for continuous action spaces.

In this work, we investigate a number of leading model-free and model-based algorithms that show promising results in challenging high-dimensional domains such as Atari games. Our toolpath design system allows exploring an unknown dynamic physics through virtual experiments in a reward driven manner.

5.2. Reinforcement Learning Framework for Toolpath Design

In this chapter, we propose toolpath design by formulating the problem in a RL framework. We design an agent to dynamically design toolpaths and iteratively learn the strategies that lead to favorable toolpaths. The reward-driven nature of this methodology provides massive flexibility in the toolpath design task as the reward can be based on a geometric feature (e.g., to finish the section as fast as possible), simulation-based mechanical performance, or in-situ experimental signals.

RL is a subfield of artificial intelligence which focuses on training agents that can interact with an environment and maximize the rewards that the agent collects through this interaction. The field of RL have experienced major breakthroughs since Mnih et al. (Mnih et al. 2013) showed that AI can achieve superhuman capability in playing Atari games. Later, several successful innovations

advanced the field in game environments such as Chess and Go as well as in robotic tasks (Silver et al. 2014, Silver et al. 2016).

In an RL schema, the agent is responsible for determining the actions at each time step (a_t), which influence the environment causing it to move from its current state (s_t) to a new state (s_{t+1}). In this context, state refers to any representation of the environment that is visible to the agent. The reward (r_t) that the agent receives encourage or discourage the agent from exploring certain state and action spaces through the training process and try to maximize the long-term reward that the agent receives in its lifespan. The overall schematic of our envisioned framework is depicted in **Figure 5.1**, where an agent designs the toolpath, which is later executed on the environment, and the resulting data is stored in a dynamic replay buffer to further improve the agent. This process is repeated until favorable strategies are found, or the agent stops progressing.

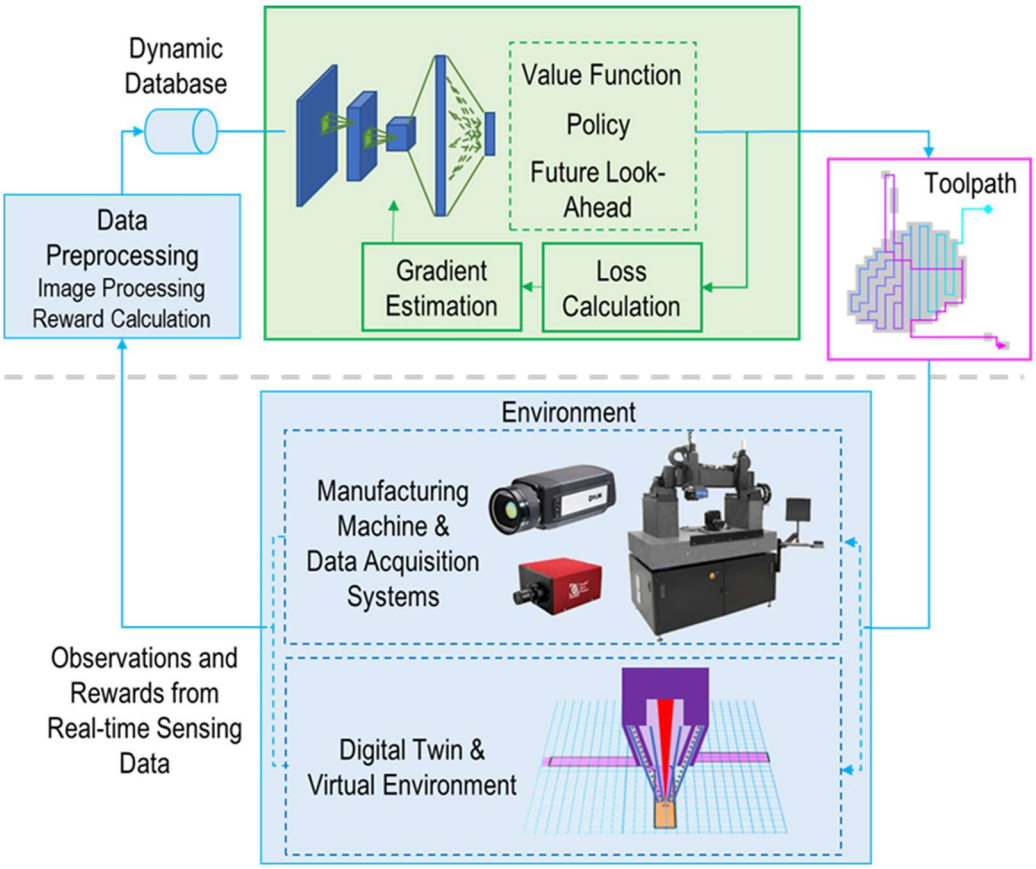


Figure 5.1. Schematics of the proposed toolpath design framework. In this framework, the agent takes an action determining the toolpath in each time step. The action would be executed in an AM (or equivalently virtual AM) environment. The resulting observation of the state and its corresponding reward would be stored in a dynamic database, which will be later used to train neural networks and achieve better planning for future iterations.

5.2.1 Additive Manufacturing Virtual Environment

We develop a virtual environment of an AM process, resembling Directed Energy Deposition (DED) processes, to collect data and perform the training process. The virtual environment considers two-dimensional sections on which materials need to be deposited. As we want the strategies learned by the RL agent to be geometry-agnostic, we develop a database of CAD

geometries representing a wide range of spatial structures. The CAD geometries are then processed into multiple sections by cutting them at different heights and converted into over 400 two-dimensional sections with 32x32 pixels, which are used to train the agent. A sample of considered CAD geometries acquired from Thingiverse online repository (Thingiverse) and one of their corresponding sections are demonstrated in **Figure 5.2**.

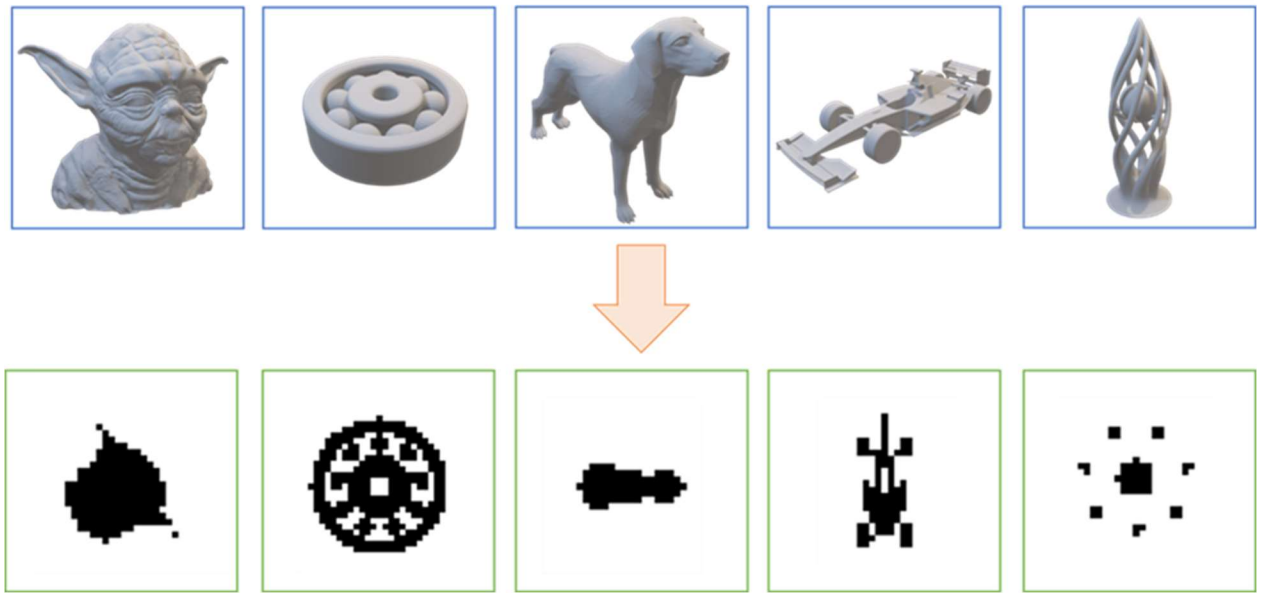


Figure 5.2. Sample CAD geometries (top row) and pixelized two-dimensional sections (bottom row) for the AM virtual environment.

While evaluating the virtual environment, one section is randomly selected and the RL agent is asked to design the toolpath for it one action at a time. Eight actions are available for the agent to explore, including four directions of motion each with two deposition states (on/off). Here, the policy distribution determines the probability of each of the eight actions to be executed. The

environment keeps track of the desired section, filled section, and the location and status of the nozzle. A representation of the environment state space (s_t) is accessible to the agent at each time step. A schematic of the environment is depicted in **Figure 5.3**. Once the agent finishes its assigned task for a section (e.g., depositing material on all pixels of the desired section), a new section is randomly selected, and the agent is asked to start over. To avoid excessively long episodes of training on one section, a maximum of 400 actions are selected for each section.

As can be seen from **Figure 5.3**, we assume a pixelized section representation and discrete action spaces. These two assumptions are not inherently restrictive for the proposed methodology as the representation of the section can be replaced with other continuous or discrete heuristics and the action space can be easily extended to a higher number of options (e.g., 8 or 16 directions) or continuous action spaces with minimal change to the algorithm.

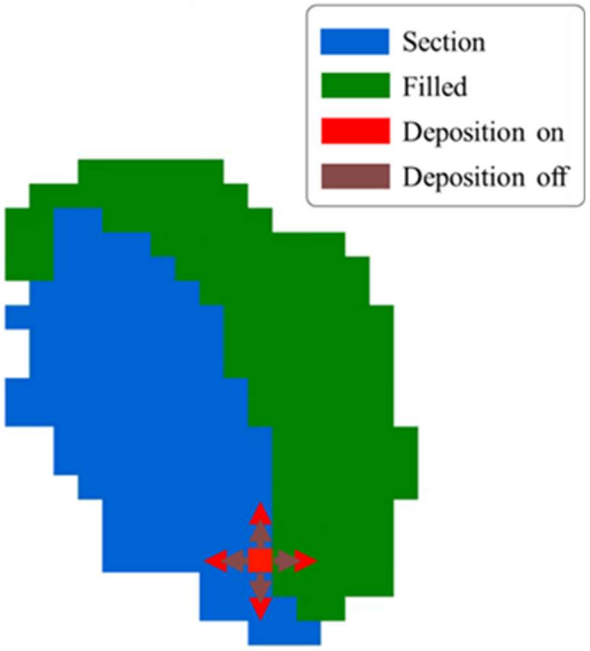


Figure 5.3. Schematic of the AM virtual environment including section (in blue), filled partition (in green), and nozzle location and status. The red point indicates the location of the nozzle with "on" status. Valid actions are shown with eight arrows for "on" (red) and "off" (brown) status and four directions.

We design the state representation (s_t) as a single-channel two-dimensional image with a shape of $[32, 32, 1]$, where the unfilled section has a value of 1.0 and the rest of the pixels are zero. Additionally, we provide the network with a one-hot encoded list of 10 most recent action histories. The image is first processed through three layers of a convolutional neural network, then concatenated with the action history input, followed by two fully connected neural network layers for policy and value networks in each algorithm.

5.2.2 Analysis Cases

We consider two scenarios for the tasks and their corresponding reward systems in this study:

1. *Dense reward system*. In this analysis, we consider a scenario in which a reward can be assigned based on the interaction of agents and environments at each time step. Designing a toolpath that deposits material in all desirable locations of the section in optimal time is an example of a dense reward system. In this case, we assign a reward of 1.0 to any desirable material deposition, -1.0 to material deposition in incorrect locations, and -0.5 to motions without deposition to provide an incentive to finish the toolpath in optimal time. It is noteworthy that dense reward structures are not limited to static properties of the environment, such as finishing the toolpath. Other examples of dense reward structures include rewards that are assigned based on the meltpool size or shape from an online thermal imaging system.

2. *Sparse reward system.* As many interesting properties of AM processes can be only measured and evaluated after the part is made, a reward can only be assigned to the completed toolpath at the last time step of the episode, which results in a sparse reward system. To simulate this scenario in the developed virtual environment, we consider the sequence of ordered actions (up, up, right, down, and down in this order) as a potential desirable pattern and assign a reward at the end of each episode of the simulation based on the similarity of the toolpath generated by the agent with the selected pattern. Note that this pattern is completely hidden from the agent, i.e., the agent can only interpret the pattern through the sparse reward it receives at the end of each episode. The similarity between the toolpath and the hidden pattern is measured by counting the

occurrence of the completed or partially completed (with a minimum of three consecutive actions) hidden patterns in the toolpath history. To encourage the agent to finish the toolpath while learning the hidden pattern, a dense reward of 0.1 and -0.1 is assigned for correct and incorrect material deposition respectively. While this specific sequence is selected as a demonstration in this work, the formulation does not depend on it, and the reward structure can be based on any unknown physics of the environment, e.g., continual deposition status, behavior on the boundaries of section.

5.3. Model-Free Approach Towards Design

In this section, we first introduce our proposed variations of three state-of-the-art model-free RL algorithms to design AM toolpaths. Model-free methods are interesting as they are computationally efficient (compared to their model-based counter parts) and do not rely on a known physics, which is difficult to obtain in the toolpath design application. Later, we discuss our results for two cases of dense and sparse reward structures and compare the performance of each method.

5.3.1 Model-Free Algorithms and Variations

We investigate three model-free RL methods, namely, deep Q-network, proximal policy optimization, and soft actor-critic. The underlying formulation, proposed variation, and algorithmic structure of each method is discussed hereunder:

1. Deep Q-network (DQN) (Mnih et al. 2015) is a Q-learning approach that parametrizes action-value function, $Q(s,a)$, using a neural network and iteratively solves the Bellman equation (Eq. 5.2) while using a number of numerical techniques to overcome problems associated with training the neural networks in RL non-stationary settings. As neural networks generalize, the Bellman equation (Eq. 5.2) tails a dynamic target (i.e., both $Q^*(s_t, a_t)$ and $Q^*(s_{t+1}, a_{t+1})$ change while training), which impedes the training process. DQN uses an additional neural network as a target network to estimate the action-value for future states $Q^*(s_{t+1}, a_{t+1})$ and solve the Bellman equation in a more supervised fashion. While the neural network training theories stand on the assumption of independent and identically distributed (i.i.d.) data, the successive data collected in RL settings are greatly correlated. To overcome this issue, DQN uses a replay buffer that stores all transactions of the environment and randomly draws samples from them during the training process. To induce exploration of the environment into the agent, DQN uses the epsilon-greedy strategy. In this algorithm, a probability for exploration (ε) is initialized. A random number between zero and one is generated at each time step. If the randomly generated number is less than ε, the agent explores (i.e., a random action will be chosen), otherwise the best predicted action so far ($argmax_a\ Q(s,a)$) will be considered. By annealing ε from one to zero over the training process, the agent acts more according to its predicted model while keep exploring new solutions.

   In this work, we consider a variation of the original DQN paper that empirically showed enhanced performance for this application. A corrected replay buffer, as proposed in (Zhang et al. 2017), is used where the last added sample into the replay buffer will be

added to the randomly selected batch to eliminate the need for an excessively large replay buffer. To reduce the overestimation bias of the Q value caused by the maximizing operation in Eq. 5.2, action selection and action-value estimation are performed using two separate neural networks, as proposed by (Van Hasselt et al. 2016). The gradient of the neural network is clipped at each training step to a value of 0.5 to avoid harmful oscillations of the neural network parameters. Finally, the hard copy operation in the original DQN paper is replaced by a moving average copy to smoothen the training process. It is noteworthy that a number of other DQN improvements in the literature, such as dueling networks (Wang et al. 2015), noisy network (Fortunato et al. 2017), prioritized experience replay (Schaul et al. 2015), are investigated but since they provided small to no improvement on the results they are not reported here. The algorithm of the implemented DQN is presented in **Algorithm 5.1**.

Algorithm 5.1: DQN algorithm for AM virtual environment

1. Initialize AM virtual environment with random sections env, replay buffer D, action-value model Q and target action-value model Q^t
2. Initialize evolving parameters including ε for epsilon-greedy, learning rate, and clipping range
3. Copy parameters of Q into Q^t
4. Reset env with random section and observe s_1
5. For iteration = 1, max number of iterations
    a. Generate a random value $rand$ between 0 and 1
    b. If $rand < \varepsilon$ then a_t= random selection between all feasible actions else a_t= $argmax_a\ Q(s_t, a)$
    c. Execute the action on env and store $(s_t, a_t, s_{t+1}, r_t, d_t)$ in D
    d. Sample a $batch_{size} - 1$ samples uniformly from D and concatenate it with the last stored sample in D to generate a corrected batch sample B
    e. For epoch = 1, number of epochs
        i. $return = r + (1 - d)\gamma Q^t\left(s_{t+1}, argmax_{\acute{a}}\ Q(s_{t+1}, \acute{a})\right)$

ii. $loss = huber\ loss(Q(s_t, a_t), return)$
iii. Take one optimization step to minimize $loss$ using $Adam$ optimizer with clipped gradients

f. End for
g. Soft update of Q^t toward Q
h. Reset env with a random section if d_t is $True$, otherwise $s_t = s_{t+1}$

6. End for

2. Proximal policy optimization (PPO) (Schulman et al. 2017) is a widely successful actor-critic method that builds on top of the policy gradient formulation to update its stochastic policy (π_θ):

$$L_\theta^\pi = \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] \tag{5.3}$$

where $\hat{A}_t$ is the advantage function and represents the difference between the value function of the selected action, $Q(s_t, a_t)$, and the average value function for that state over actions. Intuitively, maximizing Eq. 5.3 encourages the policy to increase the probability of action if the selected action performed better than average (i.e., the advantage is positive) and decreases the probability of relatively worse actions. However, this vanilla formulation tends to collapse the training process as taking large steps can easily move the policy into unrecoverable bad parameter spaces. To solve this issue, PPO restricts the ratio between current policy and previous policy by pessimistically clipping its value according to Eq. 5.4:

$$L_\theta^{\pi,clip} = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t\right), clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\right]$$
$$r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{5.4}$$

where ϵ determines the clipping range. Furthermore, PPO loss (Eq. 5.5) has two additional terms to train the advantage value $\hat{A}_t$ using the generalized advantage estimation method (L_θ^A) (Schulman et al. 2015) and to maximize the entropy (L_θ^{Ent}) of the policy to encourage exploration.

$$L_\theta^{PPO} = \mathbb{E}\left[L_\theta^{\pi,clip} + c_1 L_\theta^A + c_1 L_\theta^{Ent}\right] \tag{5.5}$$

PPO can only be trained using samples generated from its current policy (i.e., on-policy algorithm). This characteristic of PPO causes this algorithm to require a larger number of samples compared to off-policy algorithms where historic data can be reused for training the agent through the use of a replay buffer. Empirically, the effectiveness of the PPO algorithm relies on collecting independent samples from multiple streams of environments often performed in parallel virtual environments. We implemented PPO according to the presented algorithm in **Algorithm 5.2**.

Algorithm 5.2: PPO algorithm for AM virtual environment

1. Initialize policy π_θ and value function V_θ networks
2. Initialize N parallel AM virtual environments with synchronized random sections env,
3. For iteration = 1, max number of iterations
    a. Execute M action steps for N env workers according to current stochastic policy π_θ and record rewards $r_{n,m}$, actions $a_{n,m}$, terminal states $d_{n,m}$, states $s_{n,m}$, values $v_{n,m}$, log probability of actions $\log\_pi_{n,m}$ for each n, m in N, M accordingly
    b. Calculate advantages $\hat{A}_t$ given values $v_{n,m}$, rewards $r_{n,m}$ and terminal states $d_{n,m}$ using generalized advantage estimation (Schulman et al. 2015)
    c. For epoch = 1, number of epochs
        i. Create mini-batches states s_{batch}, actions a_{batch}, advantages adv_{batch}, values v_{batch}, and log probabilities $\log\_pi_{batch}$

ii. $r_t = \exp\left(\log\_pi\left(\pi_\theta(a_{batch})\right) - \log\_pi_{batch}\right)$
iii. $L_\theta^{\pi,clip} = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t\right), clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\right]$
iv. $L_\theta^{Ent} = reduce_{mean(S[\pi_\theta](s_t))}$
v. $return_{batch} = v_{batch} + adv_{batch}$
vi. $v^{clipped} = v(s_{batch}) + clip\left(v(s_{batch}) - v_{batch}, -clip_{range}, clip_{range}\right)$
vii. $L_\theta^A = reduce_{mean\left(\max\left((v(s_{bat}) - return_{batch})^2, (v^{clipped} - retu_{batch})^2\right)\right)}$
viii. Maximize the global loss function $L_\theta^{PPO} = \mathbb{E}[L_\theta^{\pi,clip} + c_1 L_\theta^A + c_1 L_\theta^{Ent}]$ with respect to θ using $Adam$ optimizer with clipped gradients

d. End for

4. End for

3. Soft actor critic (SAC) (Haarnoja et al. 2018) is an off-policy actor-critic method that aims to maximize an alternate action-value function, called soft action-value, that considers not only the accumulative reward but also the entropy of its stochastic policy. Theoretically, soft action-value formulation encourages the agent to explore states with uncertain results. The soft action-value loss is presented in Eq. 5.6.

$$L_\theta^Q = \mathbb{E}_{s_t, a_t, s_{t+1}, r_t, d_t \sim D}\Bigg[\Big(\min_{i=1,2} Q_{\theta_i}(s_t, a_t) - r_t + \gamma(1-d_t)\Big(\min_{i=1,2} Q^t{}_{\bar{\theta}_i}(s_{t+1}, a_{t+1}) - \alpha \log\left(\pi_\phi(a_{t+1}|s_{t+1})\right)\Big)\Big)^2\Bigg] \tag{5.6}$$

where θ, $\bar{\theta}$ and ϕ indicate the neural network parameters for the online action-value function, the target action-value function, and the policy, respectively. The temperature parameter, α, determines the importance of the entropy. SAC compensates for the

overestimation of action-value functions by training two independent neural networks and taking the minimum of the two for loss calculations.

As it is proven that the optimal policy can be approximated by the softmax of action-value function (Nachum et al. 2017), the policy loss is defined as the KL-divergence between the current policy and action-value softmax. To calculate the gradients of parameters through the stochastic node of policy sampling, the reparameterization trick is used. While the SAC only applies to environments with continuous action spaces, we developed a modified version of this algorithm that uses Gumble-softmax (Jang et al. 2016) to perform the reparameterization for categorical action spaces (Eq. 5.7):

$$L_{\phi}^{\pi} = -\mathbb{E}_{s_t \sim D, \xi \sim G}\Big[\min_{i=1,2} Q_{\theta_i}\Big(s_t, \tilde{a}_{\phi}(s_t, \xi)\Big) - \alpha \log\Big(\pi_{\phi}\big(\tilde{a}_{\phi}(s_t, \xi)\big|s_t\big)\Big)\Big] \tag{5.7}$$

where ξ is an independent noise sampled from a Gumble-softmax distribution and $\tilde{a}_{\phi}$ is the reparametrized action. While the temperature parameter α can be potentially kept as constant, the SAC authors devise a formulation to simultaneously train this parameter in order to constrain it to a minimum target entropy H (Eq. 5.8). The full algorithm for our SAC implementation is provided in **Algorithm 5.3**.

$$L^{\alpha} = \mathbb{E}_{s_t \sim D, a_t \sim \pi}\big[-\alpha \log \pi_{\phi}(a_t|s_t) - \alpha H\big] \tag{5.8}$$

Algorithm 5.3: SAC algorithm for AM virtual environment

1. Initialize policy π_ϕ, two action-value $Q_{\theta_{i=1,2}}$ and two target action-value $Q^t{}_{\bar{\theta}_{i=1,2}}$ networks
2. Initialize AM virtual environment with random sections env and replay buffer D
3. Copy parameters of Q into Q^t
4. Reset env with random section and observe s_1
5. For iteration = 1, max number of iterations
    a. Generate a action using current policy $\pi_\phi(s_t)$
    b. Execute the action on env and store $(s_t, a_t, s_{t+1}, r_t, d_t)$ in D
    c. Sample a $batch_{size} - 1$ samples uniformly from D and concatenate it with the last stored sample in D to generate a corrected batch sample B
    d. For epoch = 1, number of epochs
        i. Calculate $a_{t+1,B}, \log\_pi_{t+1,B}$ using $\pi_\phi(s_{t,B})$
        ii. $return = r_{t,B} + \gamma(1 - d_{t,B})\left(\min_{i=1,2} Q^t{}_{\bar{\theta}_i}(s_{t+1,B}, a_{t+1,B}) - \alpha \log\_pi_{t+1,B}\right)$
        iii. $L^Q_{\theta_{i=1,2}} = mean((Q_{\theta_i}(s_{t,B}, a_{t,B}) - return)^2)$
        iv. Calculate $\tilde{a}_{\phi_{s_{t,B}}}$ and $\log\_pi_{s_{t,B}}$ by reparametrizing $\pi_\phi(s_{t,B})$
        v. $L^\pi_\phi = mean\left(\min_{i=1,2} Q_{\theta_i}\left(s_{t,B}, \tilde{a}_{\phi_{s_{t,B}}}\right) - \alpha \log\_pi_{s_{t,B}}\right)$
        vi. $L^\alpha = mean(-\alpha \log\_pi_{s_{t,B}} - \alpha H)$
        vii. Take one optimization step to minimize $L^Q_{\theta_{i=1,2}}$, L^π_ϕ, L^α with respect to $\theta_{i=1,2}$, ϕ and α respectively using $Adam$ optimizer
    e. End for
    f. Soft update of target action values $\bar{\theta}_{i=1,2} = \rho\bar{\theta}_{i=1,2} + (1 - \rho)\theta_{i=1,2}$
    g. Reset env with a random section if d_t is $True$, otherwise $s_t = s_{t+1}$
6. End for

Although the above-mentioned algorithms have inherent differences, we attempted to keep the hyperparameters of the algorithms as consistent as possible. The rest of the hyperparameters of each algorithm is individually tuned to maximize the achieved scores.

5.3.2 Model-Free Results and Discussion

We implemented the three discussed model-free algorithms with the proposed modifications using Tensorflow deep learning library. Each algorithm is trained to design the toolpath in two cases of dense and sparse reward structures as detailed in Section 5.2.2. The learning curve of each algorithm is demonstrated in **Figure 5.4A** and **B** for a dense and sparse reward structure respectively. The reported score averages resulted scores for all training geometries from random initial nozzle location. Since the on-policy nature of the PPO algorithm requires far more episodes of toolpath generation than the two off-policy algorithms, the PPO results are plotted on a different horizontal scale for the number of training episodes.

As can be seen from **Figure 5.4A** while the three algorithms gradually learn to improve their toolpath designs, SAC achieves a notably inferior performance. DQN and PPO reach a close performance easily surpassing a manually coded zig-zag toolpath. While the final performance of the PPO algorithm is 3 scores higher than DQN, DQN reaches a stable solution using 10 times fewer samples.

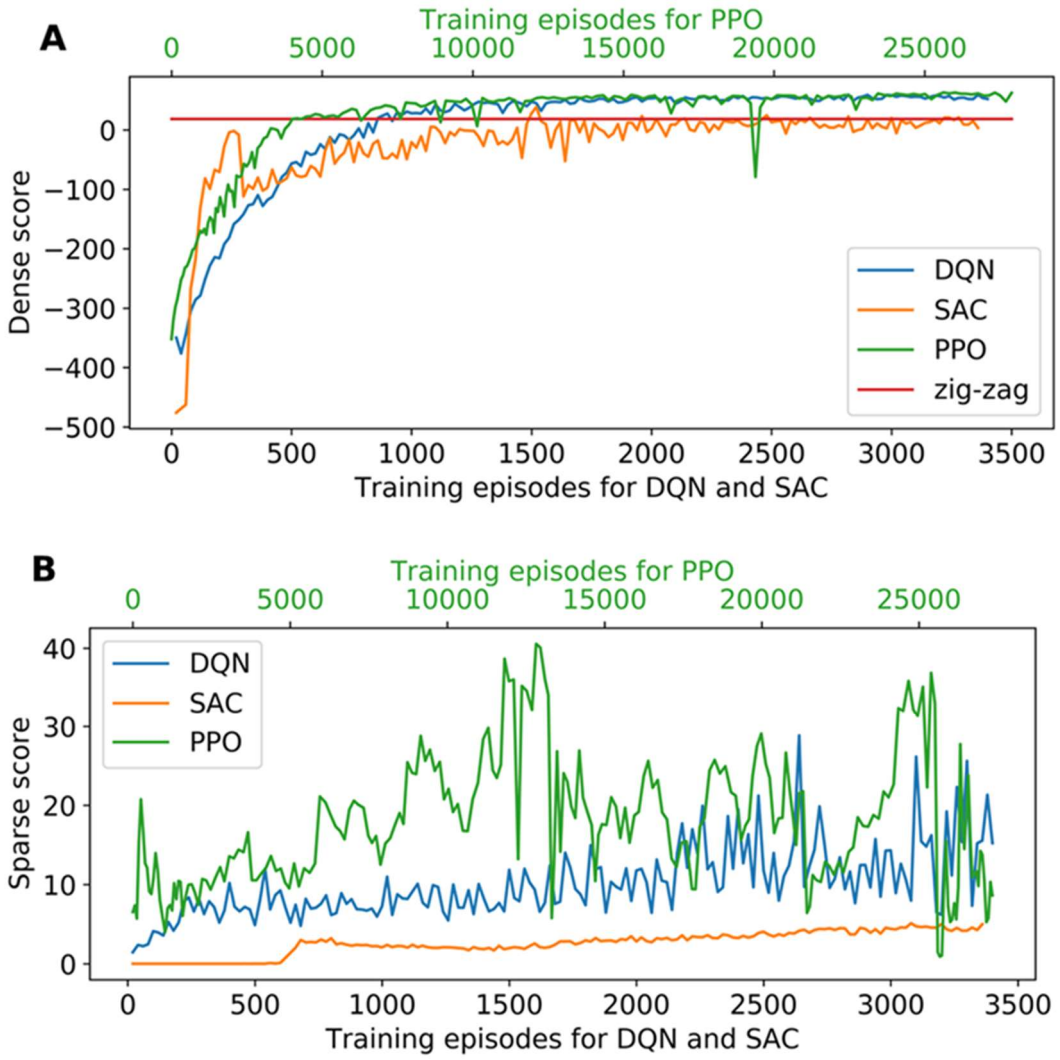


Figure 5.4. Learning curves of the toolpath design system with the three DQN, SAC, and PPO algorithms for (A) dense and (B) sparse reward systems. The horizontal axes for the PPO results are plotted at a different scale (shown on the top of each plot) from the DQN and SAC results (shown at the bottom of each plot). As the manual zig-zag toolpath strategy is plotted as a baseline for the dense reward system, such an engineered solution does not apply for the sparse reward system.

For the sparse case (see **Figure 5.4B**), the score achieved by the PPO algorithm surpasses the two other algorithms, and similar to the previous case, SAC results in the worst performance. The

highest score of the algorithms for the two cases is reported in **Table 5.1** and three samples of the designed toolpaths with the trained PPO algorithm is demonstrated in **Figure 5.5**.

Table 5.1. Highest score of model-free algorithms for two reward structure cases.

Algorithm	Dense reward	Sparse reward
DQN	59.31	28.83
SAC	38.71	5.11
PPO	62.98	40.54

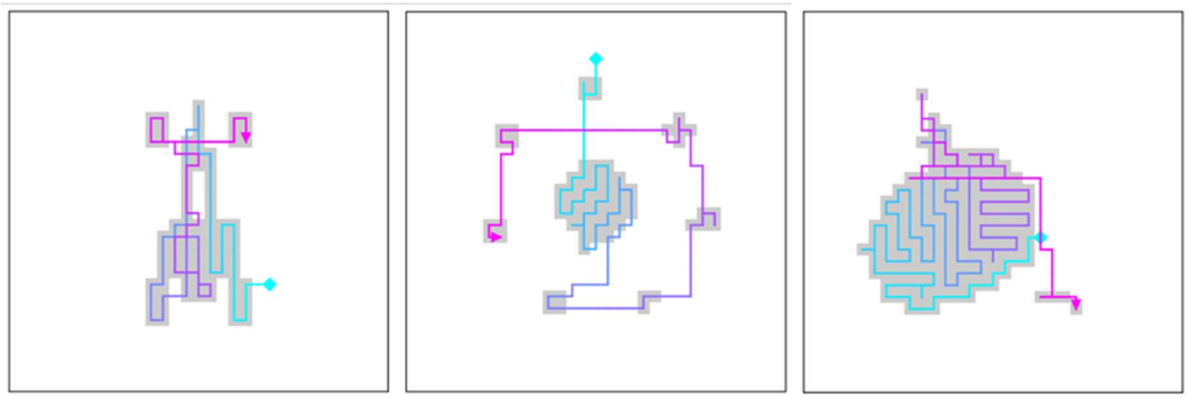


Figure 5.5. Three samples of the designed toolpaths by the trained PPO algorithm for random sections and starting locations. The section is depicted in light grey. The toolpath motion starts from the blue diamond shape, following a color gradient ending in a pink arrow shape.

Our results show that model-free reinforcement learning is a feasible approach for high-dimensional manufacturing design systems, such as toolpath design tools, especially if a dense reward system exists or it is feasible to engineer such a feature by breaking the task into meaningful step-by-step reward increments. DQN-based algorithms show great potential in this realm as they offer decent accuracy and sample efficiency. Although the SAC algorithm is reported to produce state-of-the-art benchmarks in many robotics tasks (Haarnoja et al. 2018), it is incapable of handling the intricacies of toolpath design. We believe this is because the maximizing entropy formulation used in the SAC algorithm incentivizes destructive averaging of the value function that prevents the algorithm from learning delicate features in high-dimensional environments.

In the case of a sparse reward structure, the investigated model-free approaches struggle to optimize the toolpath. PPO learns the only acceptable solution; however, its excessive on-policy sample requirement makes this algorithm only applicable to cases where a robust simulation of the physics exists.

5.4. Model-Based Approach Towards Toolpath Design

In this section, we introduce a model-based RL method for toolpath design. Model-based approaches explicitly learn the dynamics of the environment and utilize the dynamics for long-term planning through future look-ahead. The model-based method presented here offers unique advantages compared to its model-free counterparts. For instance, we expect it to better generalize to a wide range of environments due to the extracted model and to be less dependent on random exploration, which is an important feature when exploration is challenging, or meaningful rewards are sparse. However, both of these advantages heavily rely on the quality of the learned dynamic

model. Here, we first discuss the developed model-based formulation in Section 5.4.1 and later present the respective results in Section 5.4.2.

5.4.1 Model-Based Algorithm

Our implementation of the model-based toolpath design system is inspired by Muzero (Schrittwieser et al. 2020), which introduces an RL method to plan in Chess and Go games. Muzero includes two major parts: (i) planning through future look-ahead, and (ii) training neural networks. In the planning phase, we execute rollouts of the simulation, where at each step the result of a search determines the best action to take. The results of the simulation rollouts are stored in a dynamic replay buffer. In the training phase, we produce batches of observations and targets from the rollouts in the replay buffer and train a data-driven network to accurately model the environment and its dynamics. These two phases are executed iteratively, where the current model is used for better planning, and the result of the planning is used to train a better model. This process is repeated until we capture a precise model of relevant environment features that allows effective planning for hundreds of time steps in the toolpath design process. To reduce the computational time of model-based RL, both planning and training steps are heavily parallelized, where multiple agents generate and save rollouts of the simulations, and at the same time, the batching and training processes are performed asynchronously. Note that here the model-based method is trained in a purely data-driven fashion.

The planning starts from the current state observable to the agent. Here, we use a representation network to convert the state into a denser encoded state. This allows us to process information that is more relevant to the planning task and avoid the computational costs of directly working in the

state space. Then, a Monte Carlo Tree Search (MCTS) is performed in the encodes state space, where we look ahead for a number of simulation steps and based on the deduced results select the best action at each time step. To perform MCTS, at each simulation step, we select the node with the maximum upper confidence bound (UCB) score, which establishes a balance between exploring nodes with high value estimations and nodes with high uncertainties. The policy distribution resulted from the UCB is used to draw an action and expand the node. Finally, we predict a value estimation for the newly expanded node and backpropagate the value estimation and visit counts of all nodes in the tree. Repeating the mentioned MTCS process for a number of simulation steps results in a decision tree where the action corresponding to the most root child visits is selected as the best action. A schematic of this process is depicted in **Figure 5.6**.

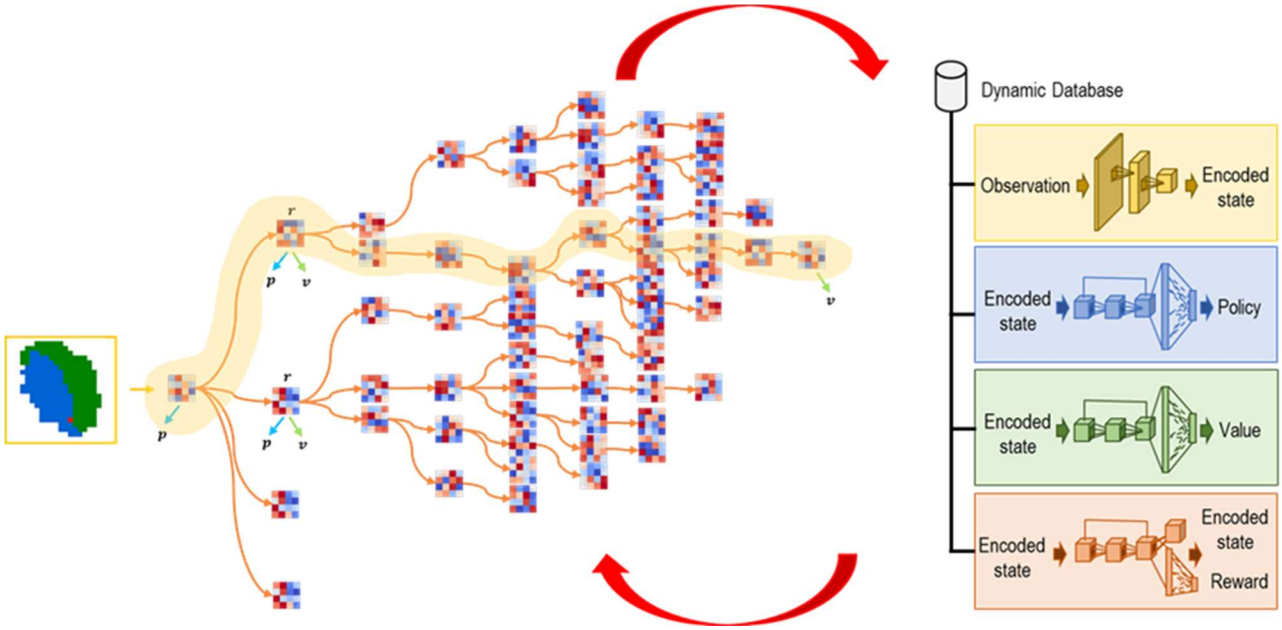


Figure 5.6. Schematics of the model-based toolpath design system which includes two major parts, i.e., MCTS planning and model training. These two parts are performed iteratively until reward convergence is achieved.

In principle, MTCS with a correct process simulator and adequate simulation steps converges to the optimal solution. However, future look-ahead for several hundreds of time steps in AM toolpath environment is computationally infeasible due to exponential growth of possibilities. Instead, we use the policy distribution, represented by a neural network, to prune the decision tree. Furthermore, we stop future look ahead with a significantly smaller depth (tens of steps instead of hundreds) and use the value to estimate potential future reward collections. Given these modifications to MTCS using a dynamically trained model, we achieve a tractable decision tree search to determine the best action at each time step. The selected action is executed in the virtual AM environment and the results of the interaction are saved in a replay buffer.

Four networks are needed in the previously discussed MCTS: a representation network, a policy network, a value network, and a dynamics network. Each network is constructed with a combination of residual, convolutional, and fully connected layers. A schematic of the network architectures is presented in **Figure 5.6**. The representation network converts the observation to the encoded state. The policy and value networks predict policy distribution and value estimations respectively, given an encoded state. The dynamics network predicts the next encoded state and the reward generated from this transition given the current encoded state and an action.

During each training iteration, we compute a batch of unrolled targets and simultaneously train the four networks. The target of the reward and value estimations are generated from the rollouts of simulations in the replay buffer using temporal-difference formulation. The target of the policy network is set to the output of the MCTS, and the dynamics network is implicitly trained to

subsequent rewards, values, and policies. Our implementation of Muzero algorithm is summarized in **Algorithm 5.4**.

Algorithm 5.4: SAC algorithm for AM virtual environment

1. Initialize networks, including representation network h, value network v, policy network p, and dynamic network g.
2. Initialize AM virtual environment with random sections env and replay buffer D
3. For iteration = 1, max number of iterations
    a. Synchronize networks and make it available to CPU
    b. Update the temperature parameter, determining the randomness is drawing actions from policy distribution
    c. Execute environment rollouts in parallel
        i. Create a new virtual environment with random section
        ii. Create a new container object to hold environment history
        iii. While environment is not finish or maximum action does not pass
            1. Create a Monte Carlo Tree Search
                a. Store the root node
                b. Use networks to calculate s_{encode}, v, p
                c. Expand root and add Dirichlet noise to p
                d. For $num\_simulations$
                    i. Add root to the search path and set to current node
                    ii. While node expanded, select a child using UCB, replace current node, and add to the search path
                    iii. Use networks to calculate $s_{encode,next}, v, p, r$
                    iv. Expand node
                    v. Backpropagate value and visit count
            2. Draw an action from distribution of actions weighted by the visit count of corresponding child nodes
            3. Execute action in the environment and store the observation, action, and reward
    d. Execute test rollouts in parallel (similar to part c while eliminating exploration noise)
    e. Store container rollouts in replay buffer D
    f. Generate training batches including observations, actions, values, rewards, and policies for n_unroll steps
    g. Train network with batches
        i. Update the learning rate
        ii. Transfer batches to GPU
        iii. Generate a computational graph using networks for predicting values, rewards, and policies for n_unroll steps
        iv. Compute a Huber loss (Girshick 2015) for value and cross entropy losses for reward and policy distributions
        v. Take an optimizer step for all networks using Adam optimizer

h. Save statistics and report reward collection progress
4. End for

5.4.2 Model-Based Results and Discussion

We implemented the model-based RL algorithm for toolpath design according to **Algorithm 5.4** using the PyTorch library and trained the model for a dense reward structure proposed in Section 5.2.2. The result of the training is demonstrated in **Figure 5.7** and shows that the Muzero method initially lags behind model-free methods such as DQN. This is due to the fact that the performance of the Muzero method heavily depends on the quality of the neural network model and capturing meaningful and generalizable dynamics requires sufficient environment samples. However, once the models are adequately trained the performance of the Muzero method reaches 72.24 dense scores surpassing DQN and PPO best scores at 59.31 and 62.98 respectively (see **Figure 5.7A**). The comparison between DQN and Muzero is summarized in **Table 5.2**.

Additionally, we demonstrate the evolution of the loss components in **Figure 5.7B** during the training process which resulted in a reward accuracy of 99.75%, a cross-entropy policy loss of 11.95, and a mean-squared-error value loss of 34.13. This result shows that the model is sufficiently capable of unrolling hypothetical dynamics of toolpaths during MCTS for approximately 20-30 steps into the future.

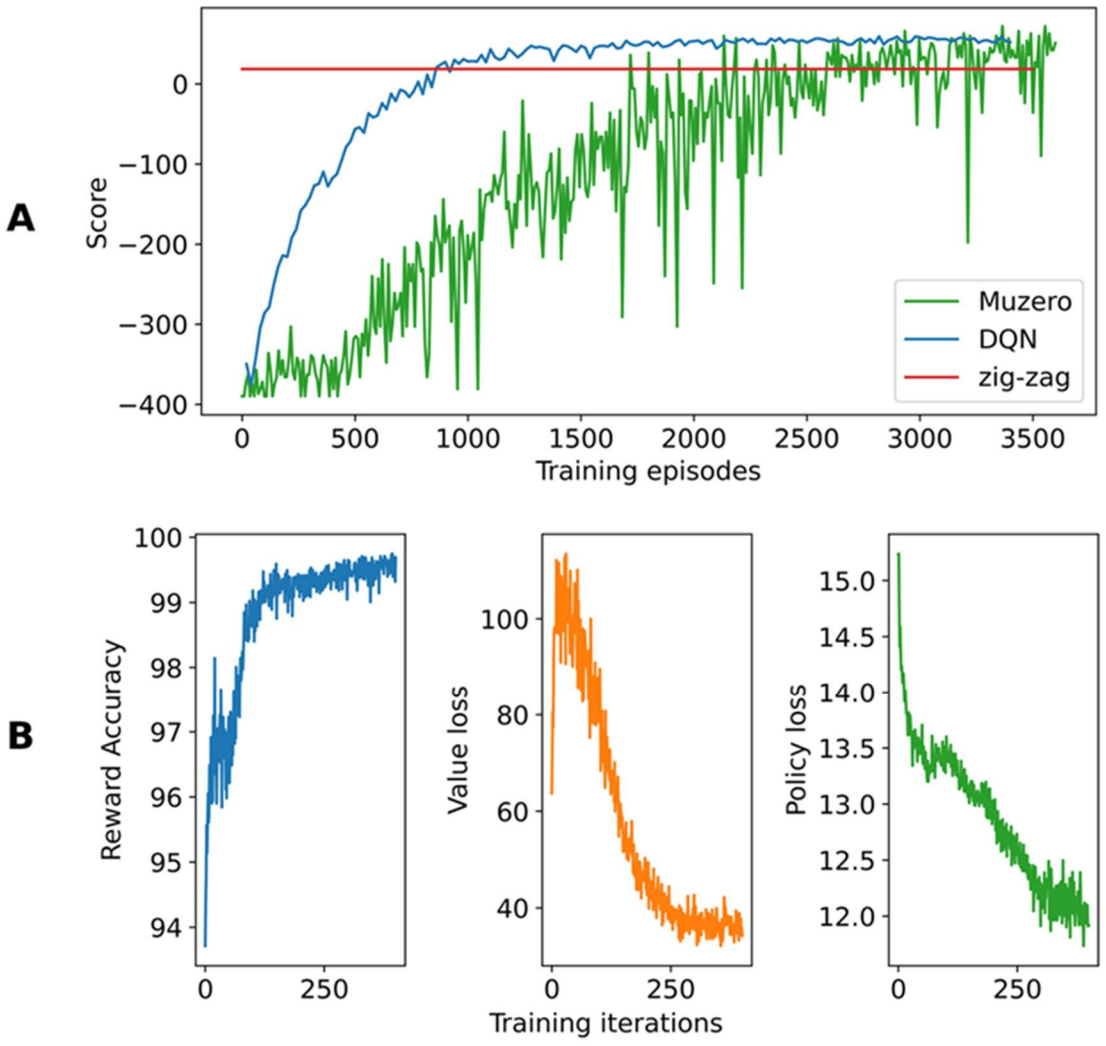


Figure 5.7. The evolution of reward score during training for Muzero and its comparison to DQN and zig-zag toolpaths (A), and the evolution of losses for reward, value, and policy terms during network training using the Muzero method (B).

Table 5.2. Comparison between model-based Muzero and model-free DQN methods.

Algorithm	Score	Computational cost	Interaction episodes
Muzero	72.24	~5 days	3,500
DQN	59.31	~18 hours	3,500

The resulting toolpaths for three representative sections are demonstrated in **Figure 5.8**, where each toolpath starts from the blue diamond and ends in the pink arrow. Our model-based method can effectively navigate both hollow and filled geometries starting from a random initial position. Furthermore, the tree search generated for the initial position of the section on the top right of **Figure 5.8**, is also depicted in this figure. This figure shows how the MCTS selectively branches out and explores different toolpath possibilities and eventually finds the optimal path by looking at 28 steps into the future.

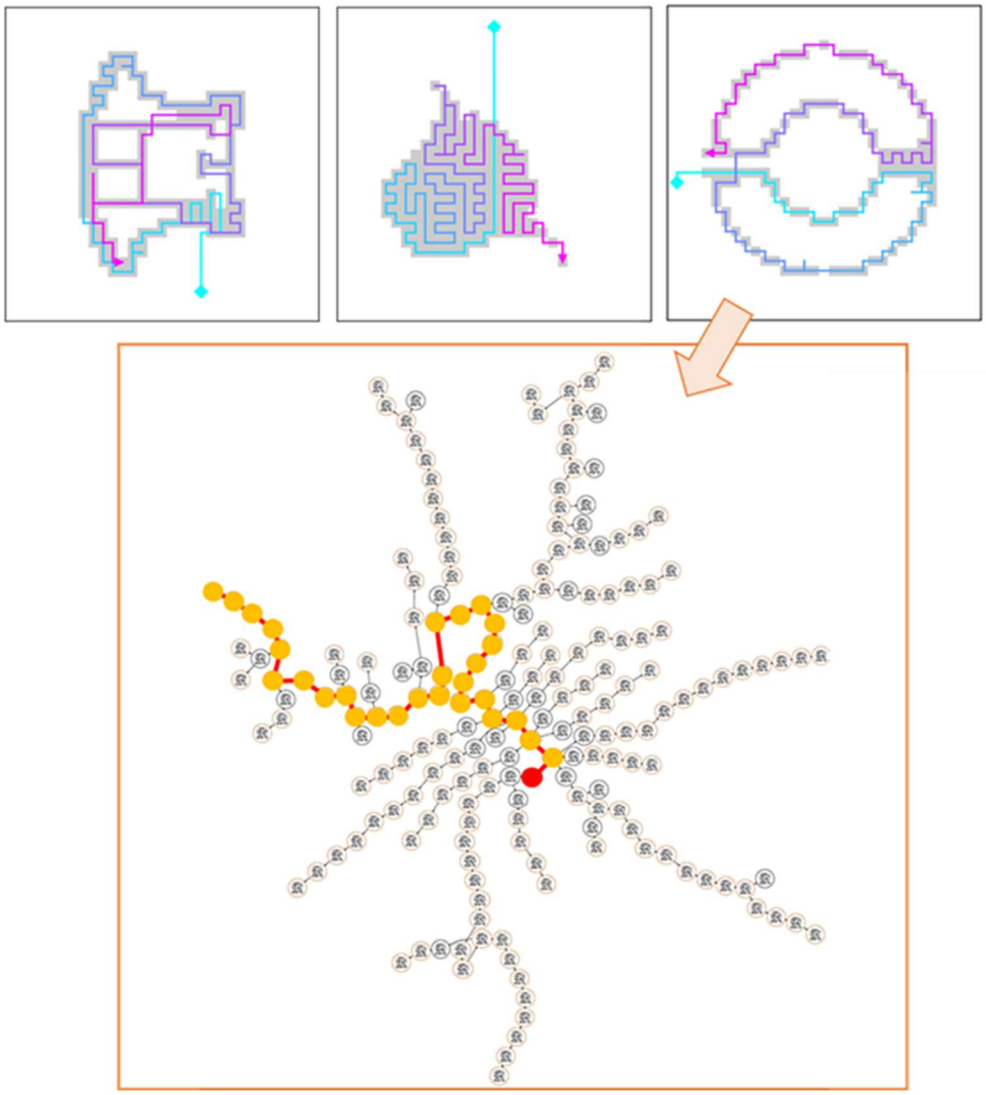


Figure 5.8. Three samples of the toolpaths designed by the Muzero method, where the toolpath starts from the blue diamond and ends in the pink arrow (top). A demonstration of generated Monte Carlo Tree Search in the initial position of the section on the top right. The root is highlighted in red and the optimal path from the root is highlighted in yellow (bottom).

Our results indicate that model-based RL methods can be deployed in design solutions of manufacturing processes as they offer unique capabilities to step beyond purely explorative methods and effectively utilize planning methods even in large design spaces. However, these

methods require a careful design and training process as they are prone to early saturation or deterioration. The dynamical database for training the models can impose many challenges that prevent us from reaching the necessary accuracy for proper planning. For example, the database is initially heavily biased toward samples with low reward and values, which generates a negative bias during network training. If left unhandled this bias severely limits the agent's explorations, as many possibilities are wrongly estimated as unfavorable. In our experience, generating a positive bias for reward and value by customizing the corresponding loss functions during the early stages of training can be an effective strategy to mitigate this issue. Additionally, model-based methods are significantly more computationally expensive as they require additional training operations to learn the dynamics of the environment.

5.5. Conclusions and Future Work

In conclusion, we proposed a new framework for the toolpath design of metal-based additive manufacturing processes by formulating a reinforcement learning problem. Modified versions of three state-of-the-art model-free and one model-based RL algorithms are used to develop toolpaths in a virtual additive manufacturing environment. Our results indicate that model-free RL methods such as DQN and PPO achieve high scores especially when a dense reward structure exists. However, achieving the highest benchmarks using on-policy methods, such as PPO, requires large sample sizes, which can be restrictive. Model-based methods, such as Muzero, offer an exciting alternative whereby learning a model of the dynamics of the environment we can deploy planning into the design problem and surpass model-free scores while using a few thousand samples.

I believe there are many interesting aspects of RL in mechanics that are unexplored and require future research directions. The first aspect is to expand the current capabilities toward more sparse structures as our results show that this is an area that results in poor performance. Another vital aspect of RL is to move toward using reducing sample requirements to enable these methods to go beyond virtual environments and interact with experimental setups. We envision that advancements in these two areas will enable RL to be deployed not only in AM but also in a variety of design and decision-making problems in the manufacturing field.

CHAPTER 6

Additive Manufacturing Process Design via Differentiable Simulations

6.1. Introduction to Differentiable Simulations

While pure data-driven modeling approaches can offer computational efficiency and flexibility in many applications, they often introduce errors in predicting challenging scenarios outside of their training domain. On contrary, physics-based modeling can conserve known physical laws over arbitrary domains. Due to the difference in weaknesses and strengths of these two approaches, one can imagine that a combination of the two, i.e., a physic-informed data-driven model, can lead to superior performance. One way to achieve such a hybrid model is to decompose the simulation into subtasks, some of which are solved using physics-based simulations while using a data-driven approach for others. For example, a physics-based model can be deployed when a subtask of simulation involves a trustworthy known physics with affordable computational cost exists, whereas parts with unreliable physics or conventionally expensive simulations can be replaced with data-driven modeling. As most modern data-driven approaches involve gradient-based optimization, to optimize hybrid simulations both physics- and data-driven-based tasks need to be differentiable, i.e., allow calculation of the gradient of their outputs with respect to their inputs and internal variables. In addition to that, differentiable physics-based simulations have many stand-alone applications in scientific computing as they provide instantaneous access to high-dimensional gradients which are necessary for most solvers and optimizers.

In recent years, differentiable simulations are increasingly used in robotics to advance the modeling and control capabilities. Hu et al. (Hu et al. 2019) used a differentiable kinematic model

in a model-based reinforcement learning setting for soft robots. Heiden et al. (Heiden et al. 2019) devised a differentiable simulation of robotic rigid body motion which led to an accurate system identification model with visual inputs. Other noteworthy publications in this field include (Liang et al. 2019, Holl et al. 2020, Qiao et al. 2020); however, the state-of-the-art studies are limited to robotics and particle-based systems.

In this chapter, we present a differentiable computational paradigm for process design in manufacturing processes that incorporates differentiable physics-based simulation and data-driven responses to optimize manufacturing process parameters in high-dimensional temporal and spatial design spaces. In particular, we aim to answer two key questions: (i) can the gradients of desired build performance be efficiently computed in manufacturing processes, and (ii) would gradient-based optimization provide an effective tool to optimize manufacturing processes in challenging environments. In a general manufacturing process, the simulation tool determines the interaction of a set of workpieces (including start and target geometry, material, etc.) and manufacturing tools as demonstrated in **Figure 6.1**. Given the initial process parameters, one can compute the performance of the physics-based simulation model by going through a forward pass (green arrows in **Figure 6.1**) of the computational scheme. Using a differentiable simulation, the gradients of a loss function (defined based on the desired performance) with respect to any of the workpiece, tools, or process parameters can be computed (red arrows in **Figure 6.1**), which can be used in an optimization setting to design manufacturing inputs.

In what follows, we first introduce the background on automatic differentiable in Section 6.2. We provide details of our methodology for differentiable finite element simulations in Section 6.3 and demonstrate the capability of our proposed approach through three illustrative case studies in

Section 6.4. Finally, we conclude this chapter by summarizing our findings and laying down our vision for future research topics in Section 6.5.

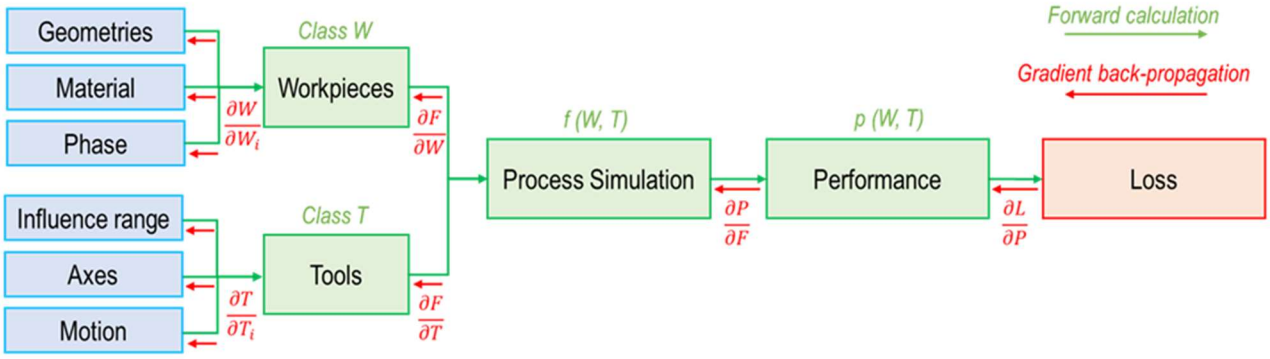


Figure 6.1. Differentiable manufacturing process simulation capable of calculating the gradients of performance loss with respect to workpiece, tool, and process parameters.

6.2. Automatic Differentiation and Libraries

The surge in the field of artificial intelligence and neural network predictive modeling is partially due to heterogeneous high-performance computing capabilities and graph-based automatic differentiation, which enables us to calculate the gradients of an arbitrary loss function with respect to any of the internal weights in a neural network and, therefore, efficiently navigate through high-dimensional weight spaces. A core idea in this research is to develop the computational graph for a physics-based simulation of manufacturing and utilize the gradients of various high-dimensional process parameters with respect to the desired performance to come up with novel design solutions.

In an automatic differentiation scheme, we construct the computational process as a composition of operations, where the gradient of each operation is known. Each operation represents a directed node in the computational graph. The forward pass computes the outputs of the simulation given the inputs while storing intermediate results. The backward pass starts from the output node and recursively computes the gradient of parameters by multiplying the incoming gradient from the next node and the partial derivation of the node evaluated at the current value. More generally, the gradient between any two parameters on the same computational graphs can be calculated by (1) finding a computational path between parameters and (2) multiplying the gradient contribution of each operation along the way. Note that such a computational path is unique by construction.

As a simple example, consider the computation of the loss function in a regression task with the following formula:

$$C = Y - \tanh(W.X + b) \tag{6.1}$$

where X is the input, W and b are trainable weight and bias parameters, $tanh$ is the hyperbolic tangent function, and Y is the true label. A computational graph of this computation can be constructed as demonstrated in **Figure 6.2**, where operations stem from input nodes in blue and gradually build toward the cost function.

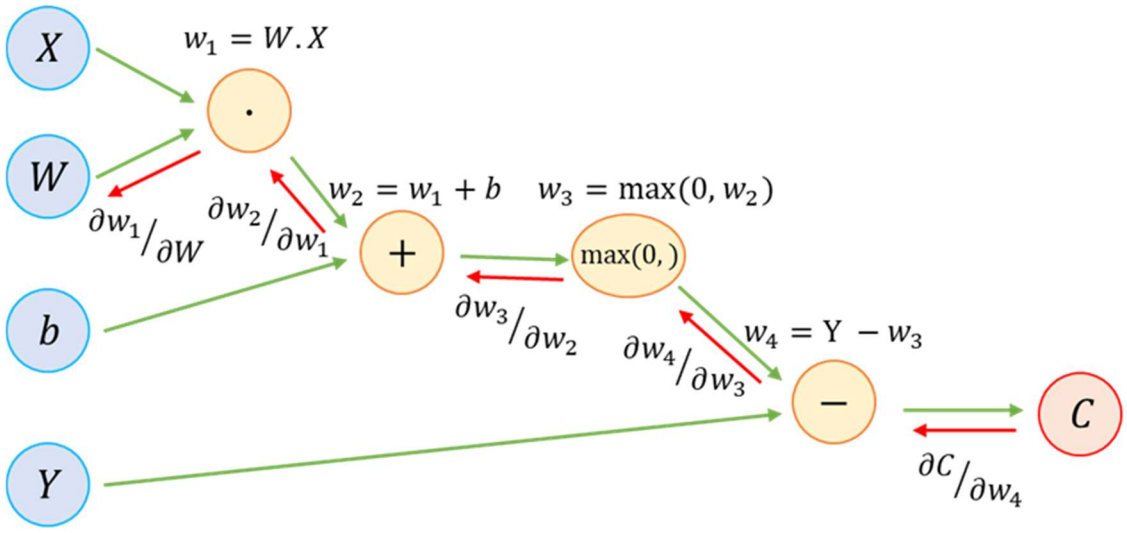


Figure 6.2. Schematic of a computational graph for computing the cost function (C) in $C = Y - \tanh(W.X + b)$. This computational graph can be utilized the forward calculation of cost function as well as backpropagation calculation of gradients.

Assuming initial values of $X = 1, W = 2, b = 3$, and $Y = 6$, we can sequentially compute and store the intermediate variables and the cost as $w_1 = 2, w_2 = 5, w_3 = 5, w_4 = 1$, and $C = 1$. For computing the gradients of the cost with respect to the weight, $\partial C/\partial W$, we can find a path between these two graph nodes. Starting from the last node (C), we can compute the gradient of the cost with respect to all the nodes on the path. Performing this operation in the reverse orders allows effective use of dynamic programming where the gradient of each node only depends on the upstream gradient (which is readily available due to the reverse order of calculations) and local gradient of that node (which only depends on a known partial derivative function and the value of the node calculated during the forward path). Thus, the gradients can be computed as shown in **Table 6.1**.

Table 6.1. Backpropagation steps for gradient calculations.

$\partial C/\partial w_4 = 1$
$\partial C/\partial w_3 = \partial C/\partial w_4 \times \partial w_4/\partial w_3 = 1 \times -1 = -1$
$\partial C/\partial w_2 = \partial C/\partial w_3 \times \partial w_3/\partial w_2 = -1 \times 1 = -1$
$\partial C/\partial w_1 = \partial C/\partial w_2 \times \partial w_2/\partial w_1 = -1 \times 1 = -1$
$\partial C/\partial W = \partial C/\partial w_1 \times \partial w_1/\partial W = -1 \times X = -1$

Several libraries efficiently construct and compute computational graphs for automatic differentiation in both forward and backward passes, such as Theano (Bastien et al. 2012), Torch (Collobert et al. 2011), TensorFlow (Abadi et al. 2016), PyTorch (Paszke et al. 2017), JAX (Bradbury et al. 2020), and Taichi (Hu et al. 2019). Each of these libraries offers different architecture choices, capabilities, compatibility ecosystem, and performance in different applications.

6.3. Proposed Methodology

As a representative manufacturing process physics-based simulation, we select AM thermal simulation to investigate the capabilities, flexibility, and limitations of differentiable simulations for manufacturing design. As mentioned before, the thermal profile of AM processes is a pivotal characteristic of this class of manufacturing processes as it determines microstructural evolution and geometric accuracy. We use a finite element formulation to solve transient heat transfer equations over the simulation domain.

In this analysis, we first define the geometry through CAD software and produce hexagonal unstructured meshes. While we implemented this method for hexagonal mesh structures, this is merely an implementation choice, and the formulation is capable of capturing other mesh structures such as tetrahedral and higher-order approximations as well. After the mesh, to generate the toolpath, we developed a Python script that slices the CAD geometry at predefined intervals in the vertical direction and produces a toolpath for each 2-dimensional section using a handful of hard-coded strategies, e.g., moving inward from boundaries, zig-zag strategy. We considered an hourglass part as a testbed for this chapter. You can see the geometry and generated toolpath for it in **Figure 6.3**. We process the geometry and toolpath to generate an element birth file which indicates the time at which each element would be born in the simulation. These three files (mesh, toolpath, and element birth), along with process parameters including laser characteristics, material properties, and simulation time step will be passed to a differentiable finite element simulation to determine the thermal responses of the AM process.

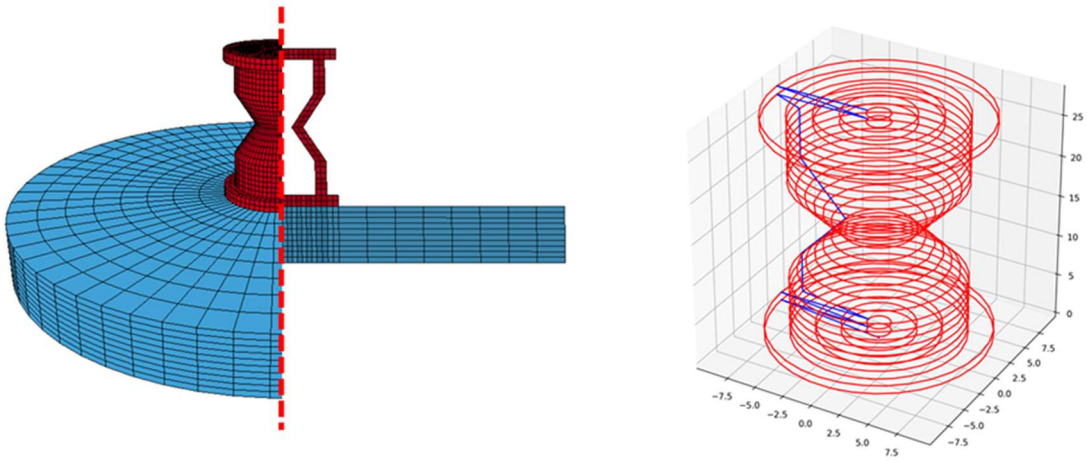


Figure 6.3. Test case geometry and its cross-section view where red elements represent the build and blue elements are the substrate (left) and toolpath pattern (right) for the differentiable AM. thermal simulations test case. The red lines on toolpath plot indicate nozzle moves while laser is on, while the blue lines indicate motion when laser is off.

In the finite element formulation, we aim to solve the partial differential equation (PDE) for transient heat transfer and the boundary conditions over the discretized geometry domain. The heat transfer equations and considered boundary conditions including fixed temperature boundary (i.e., Dirichlet boundary condition), radiation, convection, and laser power flux are provided in **Eqs. 6.2-6.6**.

$$\rho c_p \frac{\partial T}{\partial t} - \nabla \cdot (k \cdot \nabla T) - s = 0 \tag{6.2}$$

(1) Dirichlet $\quad T = T_1(x, y, z)$ on Γ_1 (6.3)

(2) Neumann $q = -q_s$ on Γ_2 (6.4)

(3) Convection $q = -h(T - T_{amb})$ on Γ_3 (6.5)

(4) Radiation $q = -\varepsilon\sigma(T^4 - T_{amb}^4)$ on Γ_4 (6.6)

where ρ is the material density, c_p is the specific heat capacity, T is temperature, t is time, k is the material conductivity, and s is the heat generate rate per unit volume. q_s is the external heat flux, h is the convection coefficient, T_{amb} is the ambient temperature, ε is the surface emissivity constant, σ is the Stefan-Boltzmann constant, and Γ_1, Γ_2, Γ_3 and Γ_4 are sets of surfaces that each of these boundary conditions are applied on. Using the aforementioned PDE and the boundary conditions, we drive the finite element weak form and discretize it for each element using shape function (N^e) and the derivative of the shape function (B^e). Finally, the forward time integration of thermal response can be derived as

$$\{T^{n+1}\} = \{T^n\} + \Delta t[\mathrm{M}]^{-1}[\{\mathrm{R_G}\} - \{\mathrm{R_F}\} - \{\mathrm{R_C}\} - \{\mathrm{R_R}\} - [\mathrm{K}]\{T^n\}] \quad (6.7)$$

where [M] is the capacitance matrix, [K] is the conduction matrix, $\{\mathrm{R_G}\}$ is the internal heat vector, $\{\mathrm{R_F}\}$ is the external flux vector, $\{\mathrm{R_C}\}$ is the convection vector, and $\{\mathrm{R_R}\}$ is the radiation vector. Δt is the time step and $\{T^{n+1}\}$ and $\{T^n\}$ are nodal temperatures at time steps $n+1$ and n, respectively. Each of these global matrix and vectors can be computed by assembling the local

contributions of each element to them individually. Interested readers can find a more in-depth derivation of the finite element formulation in Section 7.2 of this thesis. To simulate AM processes, we need to dynamically keep track of active elements and surfaces at each time step and the local contributions of only active objects will be assembled in global matrices.

To use gradient-based optimization methods, it is ideal for all operations to be continuous and differentiable. In addition to that, a careful design is needed when working with operations that can eliminate and saturate the gradient propagation. For instance, a step function stops the propagation of the gradients from a smooth function as its local gradients are zero in all continuous points. As another example, functions such as sigmoid and hyperbolic tangent although generate valid and smooth gradients, if called recursively in the form of $f(\dots f(f(x)))$ become gradually closer to a step function where they produce gradients very close to zero for all inputs except for a very narrow region where their gradients are very large. This phenomenon is known as the vanishing/exploding gradients and prevents effective optimization using gradient signals. Many sources of discontinuities exist in thermal analysis of AM processes, especially due to the discontinuous nature of material deposition in a meshed domain. Here, we hypothesize that by fixing the geometric and boundary-related discontinuities in the simulation, the main derive behind thermal responses would be the continuous material and process parameters which can be optimized using a differentiable simulation.

As mentioned before, in recent years, many libraries have been developed to create computational graphs and differentiable numerical solutions using automatic differentiation, most of which are focused on the operations common in deep learning such as convolution and various matrix operations. Physics-based simulations inherently require a more diverse set of operations

such as random indexing and large-scale atomic operations. Therefore, there is a need to investigate the performance and capability of the existing libraries to perform thermal simulations in a dynamic domain of AM processes.

In this research, we developed four implementations of AM simulators for PyTorch, TensorFlow, JAX, and Taichi libraries and optimized each implementation according to each library's guidelines. The result of this analysis is summarized in **Table 6.2**. To the best of our knowledge, TensorFlow version 2.1 lacks the flexible indexing capability required to perform assembly operations in FEM, and therefore, we were not able to develop a successful implementation of differentiable AM simulation using this library. While the other three libraries showed adequate capabilities to build and differentiate through the simulation stack, we observed a substantial gap in the memory consumption and processing time between these three libraries. On a benchmark task, PyTorch used an unreasonable amount of memory (127 GB), JAX operation time is unacceptable with one simulation taking over 10 hours. This is because these libraries offer highly optimized high-level operations in computer vision and natural language processing applications, but large-scale usage of low-level operations, which are ubiquitous in AM simulations, leads to inefficient buffer allocations and GPU kernel launches and severely hurts the overall performance. These performance issues prevent these implementations to be realistically used in even moderately large simulation scenarios. In our investigation, Taichi library showed a favorable performance with 2X smaller memory usage and 8X smaller processing time compared to the best performance of other libraries as they provide efficient support for GPU mega-kernels, flexible indexing, and atomic operations. Therefore, the Taichi library is used to perform automatic differentiation in the rest of this chapter.

Table 6.2. Performance comparison of prominent automatic differentiation libraries for manufacturing simulations.

Automatic Differentiation Libraries	Calculation Capability Support for Operations Needed in FEM	Memory Usage	Calculation Time for One Optimization Iteration
PyTorch	Yes	127 G	40 mins
TensorFlow	Lack of support for matrix assembly	---	---
JAX	Yes	2 G	10 hours
Taichi	Yes	1 G	5 mins

6.4. Optimization Process and Results

Here, we investigate the capability of the developed differentiable AM simulation to optimize various process parameters and material properties of the process in three case studies. In the first case study, we test our framework to optimize static parameters with partially observable data. The second case study optimizes the entire thermal history of the build by manipulating time-series laser power during the build. Finally, the third case study investigates the capability of our framework to stabilize the melt pool depth as a derived feature from the thermal response by optimizing time-series laser power. The details of case studies and their results are elaborated in the following subsections:

6.4.1 Parameter inference based on partial data

We devised a case study where we optimize a set of static parameters including material properties and process parameters to obtain a predefined thermal behavior using the build process. The properties investigated in this case include heat capacity, conductivity, convection coefficient, static laser power, and laser beam radius. We initialize the investigated parameters using a uniform distribution over a reasonable range of each parameter. Then, the differentiable simulation generates the output thermal history corresponding to the current parameter set. The loss value is then calculated based on the mean-squared-error (MSE) difference between the target and current responses of the nodes on the top layer of the build at each time step. Finally, the gradient of the loss function with respect to all investigated parameters is calculated using automatic differentiation and the gradient is used to update each parameter using the Adam optimization method (Kingma et al. 2014). This process is repeated for a set number of iterations, also known as epochs, until we observe a good match between the target and current thermal responses. A schematic of this case study is demonstrated in **Figure 6.4**.

The goal of this analysis is two-fold. First, this case is intended to resemble a model calibration with experimental data, where only part of the build is observable through sensory data such as an IR camera. Therefore, this framework allows us to infer material and process parameters from the process thermal response. Second, the selected set of parameter covers a wide range of operations in FEM analysis, and therefore, this task demonstrates the capability of automatic differentiation to handle many critical operations throughout the constructed computational graph including matrix operations, assembly, lumping, distribution calculation, temporal mapping, to name a few.

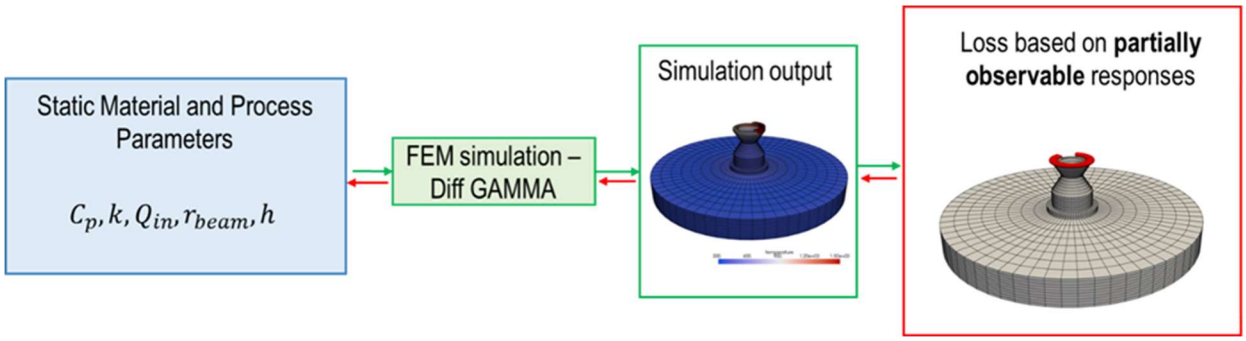


Figure 6.4. Schematic of the first case study where the partially observable loss function based on the thermal responses of top build layer at each time step is optimized. The optimization paramters include heat capacity, conductivity, convection coefficient, static laser power, and laser beam radius.

The optimization results show that by performing 60 optimization iteration, we reach an error of 1e-3 MSE on the partially observable loss. Moreover, as shown in **Figure 6.5**, each parameter effectively converges to (or at least moves in the direction of) the parameters that generated the target in the loss function. Note that the target parameters in **Figure 6.5** are solely provided as verification of gradient directions and the optimization method does not have access to them; rather, it interprets them using the target thermal response. Overall, this result shows the proposed differentiable method can infer various simulation parameters even when given access to a fraction of simulation responses and elucidates the high potential of differentiable finite element simulations for describing unknown process and simulation parameters. While more optimization steps would bring parameters such as heat capacity closer to their target values, we do not expect it to converge to the exact target as the interaction between optimization parameters and the thermal response is highly coupled. For instance, a similar thermal response can be achieved by underestimating the heat capacity and over-estimating the laser input. Therefore, we believe the

fact that all parameters move in the correct direction to collectively reduce the loss to close to zero is a more important result than tuning the optimization steps in a way that each individual parameter reaches its target.

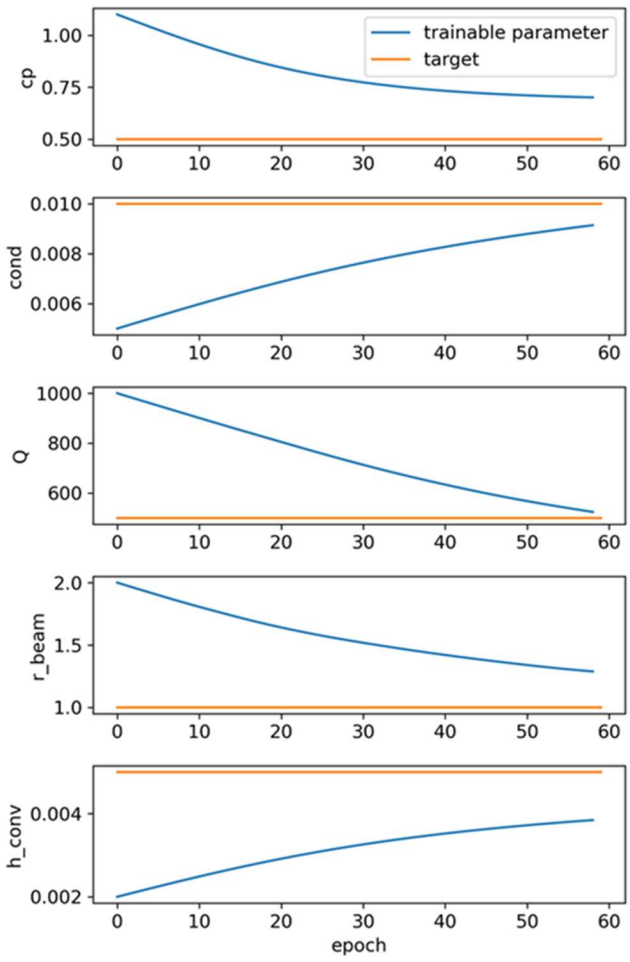


Figure 6.5. Evolution of the investigated process paramters over 60 iterations of optimization.

6.4.2 High-Dimensional temporal design for thermal history behavior

In the second case study, we explore the capability of differentiable simulation to design high dimensional temporal aspects of the additive manufacturing processes. We use a fully connected neural network architecture to represent the time-series laser power. The neural network receives the time at each time step and outputs the laser power for that time step using two hidden layers of 50 neurons with hyperbolic tangent as the nonlinear activation function to map the output to a range of $0 - 1{,}000$ W.

We selected this network setting as it allows generating sufficiently complex behavior of laser power over approximately $20{,}000$ time steps of the simulation. Note that these hyperparameters can be adjusted according to the desired nonlinearity in the response. Using this approach, the neural network controls the temporal evolution of the laser power. The computed laser power is then fed into the differentiable simulation and an MSE loss function is defined between the current thermal response and an ideal predefined thermal response. In this case, the ideal thermal profile is developed by simulating the process with a complex laser power pattern. This laser power pattern is not used in the optimization process and is only later used to validate the answer found by differentiable optimization. The optimization task entails computing the gradient of the loss function with respect to weights and biases of neural network and iteratively updating these parameters to minimize the loss. Stainless steel material properties are assigned to the simulation in this case study and unlike the previous case study, they are kept constant during the optimization process. A schematic of this case study is provided in **Figure 6.6**.

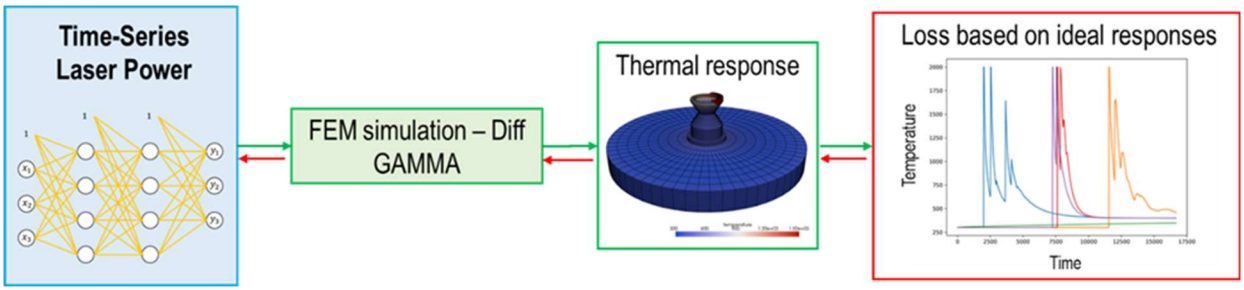


Figure 6.6. Schematics of the second case study. In this case, a neural network structure determines the time-series laser power of the AM process, and it is optimized to produce an ideal thermal behavior during part build.

Similar to the previous case, the Adam (Kingma et al. 2014) algorithm is used to optimize the loss. The evolution of MSE loss over 300 iterations is demonstrated in **Figure 6.7A** which shows the loss decreases about 3 orders of magnitude as the result of optimization. We observe a favorable optimization behavior with the loss function rapidly declining and minimal loss jumps, which shows the suitability of gradient-based methods (albeit with momentum and learning rate scaling) for optimizing time-series parameters in the AM simulation. The evolution of time-series laser power is depicted in **Figure 6.7B** where the true laser power target used to produce the ideal thermal history in the loss function is plotted in black line. This true laser power target is intentionally designed to show sharp changes and complex evolution during the build time. The evolution of the output of the neural network including the initial state, five intermediate states, and the final state after 300 optimization iterations are plotted on the top, middle, and bottom subplots of **Figure 6.7B** correspondingly.

These results indicate that our proposed approach can optimize the time-series values with high accuracy to match an arbitrary target, which is a unique feature of differentiable simulation as it can access accurate gradients of high-dimensional spaces. Additionally, this result exhibits the

natural integration of differentiable physics-driven manufacturing simulation with powerful data-driven modeling techniques as another impactful benefit of this approach for the development of physics-informed data-driven methods.

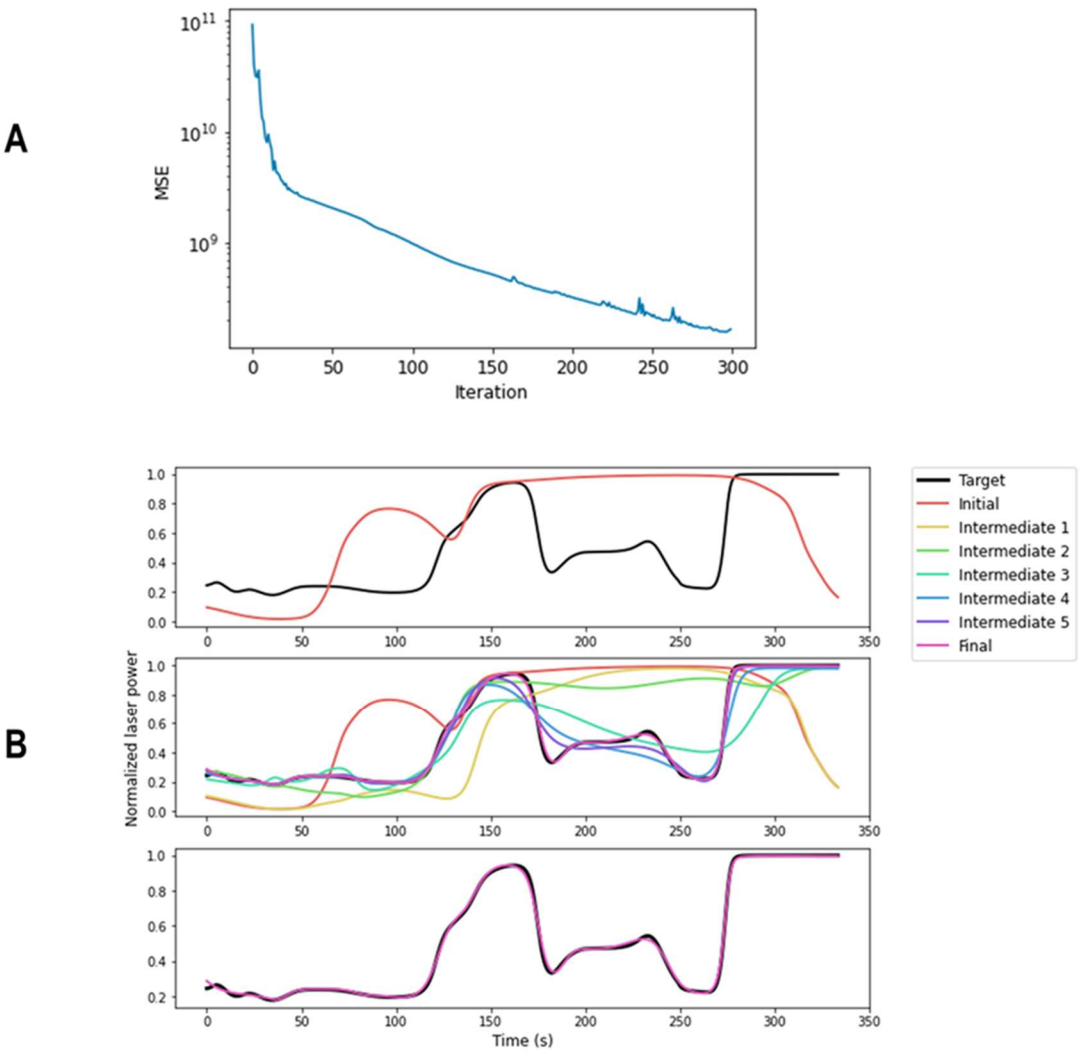


Figure 6.7. Optimization results for the second case study. (A) evolution of the MSE loss function over 300 optimization iterations. (B) evolution of time-series laser power with the initial laser power plotted in red (see top row), five intermediate laser power patterns during the training (see middle row), and the final pattern found by differentiable optimization after 300 iterations and its comparison with the true target (see bottom row).

6.4.3 High-Dimensional temporal design for melt pool behavior

In the last two cases, we demonstrated that thermal history can be used as a target for process optimization; however, our proposed computational design approach can be extended to any derivative feature of thermal history that can be computed through a differentiable formulation. In this case, we aim to achieve a target melt pool depth by manipulating time-series laser power. Similar to the second study, we utilize a fully connected neural network to produce time-series laser power that is parameterized by neural network weights and biases. As can be seen from the schematics in **Figure 6.8**, the produced laser power is used in the differentiable AM thermal simulation. Later, the thermal responses are used to calculate the melt pool depth at each time step and an MSE loss function is defined that penalizes the melt pool depth deviations from a predefined depth throughout the build. As melt pool characteristics significantly affect the geometric accuracy of AM processes, investigating systematic solutions to design melt pool features is an important step toward AM parts with customized properties.

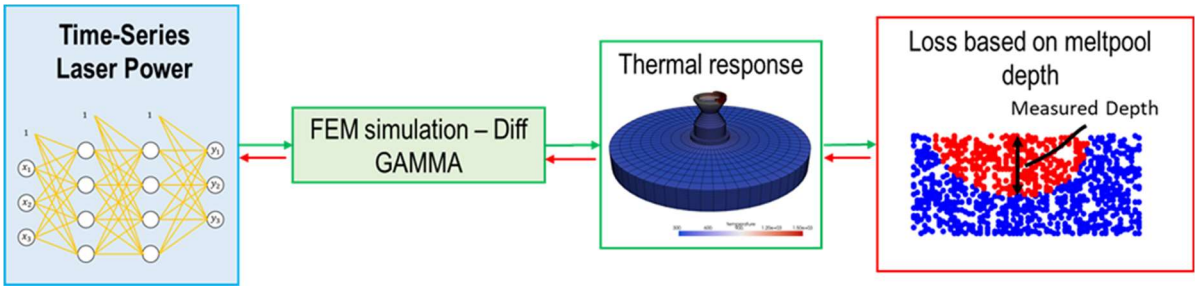


Figure 6.8. Schematics of the third case study. In this case, we stabilize the melt pool depth throughout the build time by adjusting time-series laser power. The laser power is determines using a fully connected neural network as a universal function approximator and the parameters of the network are tuned using a gradient-based optimization method.

The key to performing this task is to develop a mapping between thermal features and melt pool depth in a way that produces meaningful gradients. For example, calculating the melt pool solely based on the deepest node with a temperature higher than melting temperature although differentiable, does not lead to a helpful optimization method. This is because using the previously mentioned method depth changes similar to a step function which produces zero gradients at each continuous point and therefore stales the gradient-based optimization process. Instead, to compute the continuous representation of the melt pool depth, we dynamically find nine nearest neighboring nodes to the laser location at four height levels starting from the top build layer. At each height level, we interpolate the temperature in the location below the laser beam by solving a ninth-degree system of equations (see **Figure 6.9B**). The temperature bellow laser at each height is then used to compute the continuous melt pool depth using a pairwise linear solver (see **Figure 6.9C**).

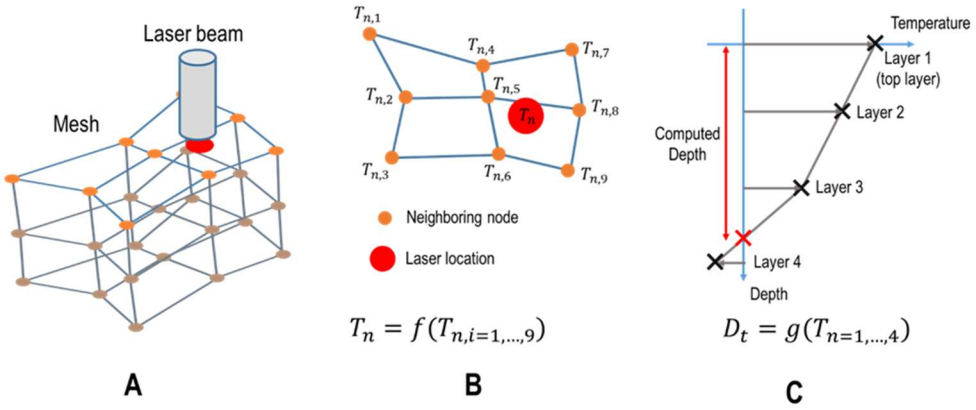


Figure 6.9. Differentiable melt pool calculation scheme. (A) schematics of a 3 layer mesh structure and the location of laser beam. (B) nodal temperature of nine neighboring nodes are used to compute the temperature corresponding to laser location at each height. (C) a linear pairwise solver is used to compute continuous melt pool depth at each time step.

Our results shown in **Figure 6.10** indicate that starting from a randomly initialized laser power pattern, we can learn a high-dimensional time-series laser power to control melt pool depth over thousands of FEM time steps. The evolution of MSE loss between desired and predicted melt pool depth is plotted in **Figure 6.10A**. Without optimization, we see a rapid increase in melt pool depth due to the heat accumulation especially halfway during the simulation as the laser builds the bottleneck of the hourglass geometry (see the blue curve in **Figure 6.10C**). However, after optimization, the laser power sharply decreases after the first few lasers to keep melt pool depth close to the target depth and gradually increases it toward the end of the build to account for additional material deposition of the top layers of the hourglass geometry (see the red curves in **Figure 6.10B-C**).

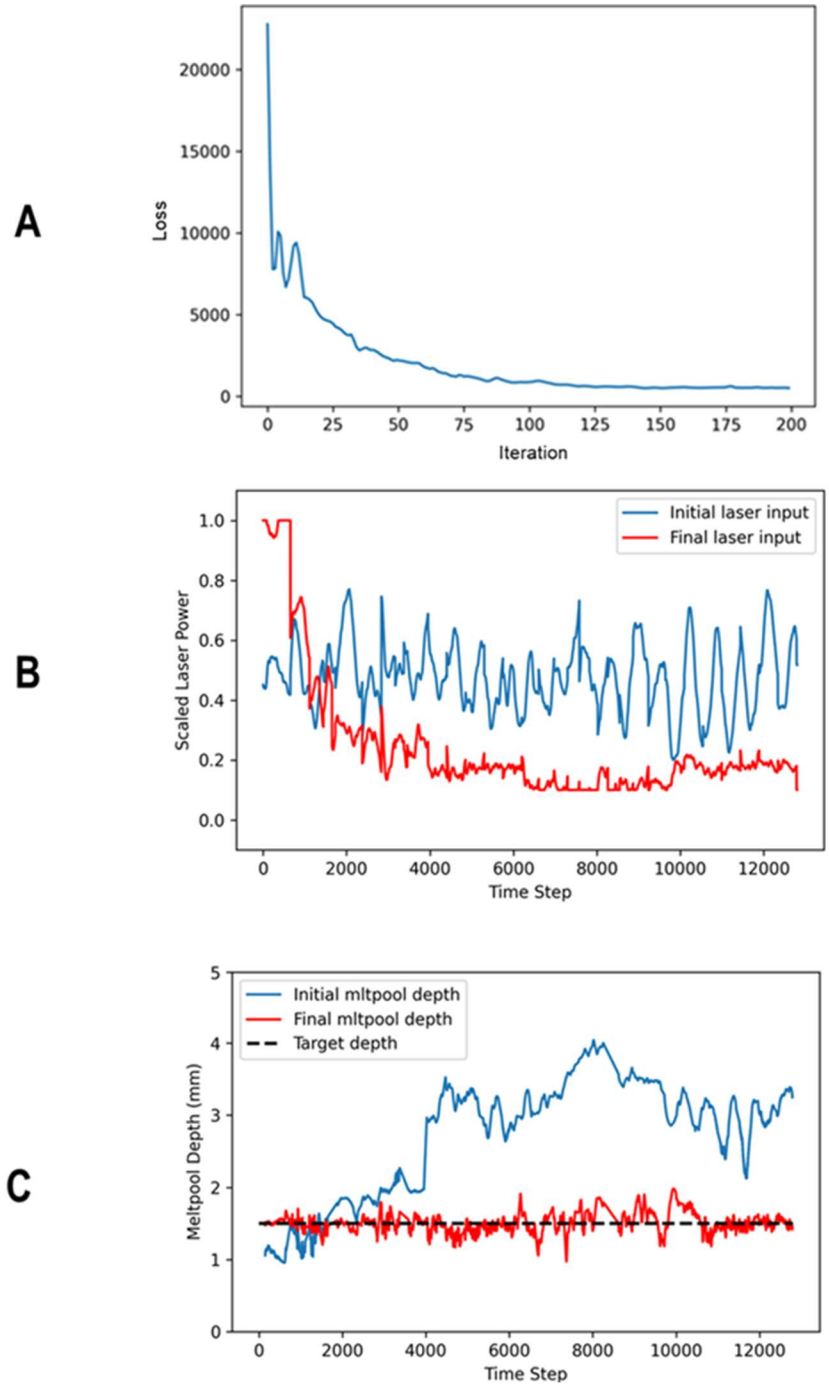


Figure 6.10. Optimization results for the third case study. (A) the evolution of MSE loss function between the desired melt pool depth and achieved depth. (B) the initial and final laser power after 200 optimization iterations on neural network parameters. (C) the initial melt pool depth, final depth after 200 optimization iterations, and target depth used in loss function definition.

6.5. Conclusions and Future Work

In this chapter, we laid out our vision on differentiable physics-based simulation in manufacturing processes and particularly in AM. We demonstrated the capability of differentiable simulations to design and optimize the various process and material properties of the process in three representative case studies for (i) inferring static material and process parameters from partially observable data, (ii) designing time-series laser input to obtain a predefined thermal response, and (iii) designing time-series laser input to stabilize melt pool depth during the build. In all three cases, we showed that one can calculate the gradients using automatic differentiation and the gradients do not suffer from saturation or corruption even over tens of thousands of time steps. Therefore, the gradients can be effectively used in gradient-based optimization methods, such as Adam, to obtain favorable responses and eliminates the need for approximated ad-hoc solutions. This approach is particularly helpful in designing high-dimensional parameters, such as time-series parameters, where other optimization methods fail to provide a viable solution.

While we believe this approach shows great promise, many research avenues require further investigation. The first issue with the widespread application of differentiable simulations is that not all operations are inherently differentiable. For instance, we found it difficult to establish differentiable operations to perform a search and find the last time that a material undergoes the melting process. Note that one can pre-compute the step that the remelting happens and hard-code this information into a differentiable solution; however, developing a differentiable system to dynamically find this solution remains unsolved. Therefore, a main future research direction involves developing differentiable alternatives for many discontinuous algorithms and formulations in scientific computing. Finally, as differentiable simulation and optimization rely on

local gradients, it is prone to stagnation in locally optimal solutions, and it can be heavily influenced by the initialization. Moreover, as in all general non-convex optimization methods, the solution is not unique. Therefore, careful design of the optimization process and initialization method is often needed to ensure satisfactory results.

CHAPTER 7

Acceleration Strategies for Physics-based Modeling of Additive Manufacturing Processes using Graphical Processing Units

7.1. Introduction

As mentioned in Chapter 1, the uncertainty in predicting the final properties of the products is one of the most critical challenges of AM technologies. Many computational methods have been proposed to address this issue using macro-scale (Parry et al. 2016, Schoinochoritis et al. 2017), meso-scale (Khairallah et al. 2016, Rai et al. 2016) or multi-scale modeling (Wolff et al. 2017, Yan et al. 2018). The finite element method (FEM) is a key component in most physics-based based predictive models for AM, which can be used for predicting the thermal history of the process (Schoinochoritis et al. 2017), residual stresses (Zaeh et al. 2010), distortions (Neugebauer et al. 2014), and porosity (Yin et al. 2012), to name but a few. However, a common problem with the existing predictive methods for AM is their enormous computational cost that might take weeks or months of simulation time (Francois et al. 2017), which makes these computational models orders of magnitude slower than the experiment itself and impossible to use in any time-sensitive application such as real-time control or optimization procedures. Therefore, investigating methods to accelerate AM predictive models is vital for overcoming existing barriers and achieve wider application of AM technologies in the industry.

One approach to overcome this hurdle is by accelerating the AM prediction computations using parallelization practices on computer clusters or more recently Graphical Processing Units

(GPUs). GPUs are traditionally designed to handle computer graphics and their hardware is designed to perform optimally for that task. With the emergence of the General-Purpose GPU (GPGPU) concept, the application of GPUs extended to many science fields and revolutionized computations in finance, bioinformatics, machine learning and computer vision (NVIDIA 2016).

FEM calculations consist of two major tasks: (i) creating a large system of equations based on the physics-based partial differential equations on a discretized domain and (ii) solving the system of equations. GPUs can be used to accelerate the process of solving FEA systems of equations. Efficient GPU-accelerated libraries such as THRUST exist that handle the iterative procedure of solving matrix-based equations. Solving sparse systems of equations on GPUs is extensively investigated (Bolz et al. 2003) and well-developed libraries are publicly available such as cuSPARSE. Recently, commercial FEM software such as ABAQUS, COMSOL, etc. use this technique to boost the performance for their analysis. A benchmark of the acceleration performance of different matrix solvers for the simulation of polymer actuator's electromechanical response is developed in (Price 2013). An implicit simulation of the automobile battery thermal runaway is accelerated using THRUST and PARALUTION libraries as equation solvers in (Pichler et al. 2017).

A more effective strategy would be do both creating the FEM systems of equation and solving them on the GPU. A fundamental investigation of this method is presented in (Cecka et al. 2011) for the simulation of steady heat equation. They considered different work distributions and memory arrangements for the implementation which resulted in 30 times speed-up (depends on the element order and simulation size). A high-level domain-specific language was developed for implementation of FEM simulation on both CPUs and GPUs in (Markall et al. 2010, Markall et al.

2013). Using the developed platform, called Unified Form Language (UFL), they investigated the memory storage and access patterns that led to optimal performances in CPU and GPU implementations. Further, they introduced the Local Matrix Approach (LMA) as an alternative assembly algorithm to eliminate the necessity for atomic operations. In (Dziekonski et al. 2011, Dziekonski et al. 2012, Dziekonski et al. 2016) the matrix generation method is divided into three consequent tasks of: (i) numerical integration, (ii) assembly is COO format and (iii) conversion into CRS format. They used GPU computing for simulating 9-pole microwave electromagnetic responses by distributing the GPU work based on each FEM integration point. The performance of the sparse systems of equation solver is improved using the Conjugate Gradient Method (CGM) and preconditioners, which lead to 81 times speed-up of the simulation. Global memory accesses and calculations are interleaved to achieve 100 billion floating-point operations per second in (Knepley et al. 2013, Knepley et al. 2016). They also proposed a mapping between elements and integration points to eliminate the reduction operations. Georgescu et al. (Georgescu et al. 2013) discussed the existing works and potentials of the GPU computing for the structural analysis components including model conversion, meshing operation, solver, and visualizers.

As it can be seen from the presented literature review, most existing studies are focused on the steady state or implicit solutions of FEM and there is a gap in the knowledge for acceleration strategies that lead to optimal performance for explicit simulation. Further, the investigated FEM problems often have relatively simple boundary conditions. In this research, we investigate the strategies and data structures that can lead to an optimal acceleration of the thermal analysis of the DED processes, which has complex boundary conditions due to the dynamic element and surface

creations and destructions. An explicit solution is considered for this problem because of its superior convergence characteristics and compatibility with nonlinear boundary conditions.

In this chapter, a summary of the finite element formulations for transient heat transfer is presented in Section 7.2 while an overview of the GPU execution and its hierarchical memory model is discussed in Section 7.3. Acceleration strategies to overcome challenges associated with explicit FEA and boundary conditions are discussed in Section 7.4. The results of the acceleration on multiple test cases and verification of the calculations are presented in Section 7.5 and the future path for this research is discussed in Section 7.6.

7.2. Finite Element Formulation for Transient Heat Transfer

In this section, a summary of the underlying FEA formulation for thermal analysis of AM is presented. First, the weak form of the transient heat equation will be derived from the governing equation and the boundary conditions. This weak form will then be discretized using the Galerkin method for each element. Then the Gauss quadrature and explicit time integration schemes will be used to solve the global system of equations assembled from the local system of each element. Here, only the key formulations are highlighted while the detailed mathematical steps can be found in (Fish et al. 2007, Belytschko et al. 2013).

The governing equation for the transient heat transfer that is to be solved can be written as:

$$\rho c_p \frac{\partial T}{\partial t} - \nabla \cdot (k \cdot \nabla T) - s = 0 \tag{7.1}$$

where ρ is the material density, c_p is the specific heat capacity, T is temperature, t is time, k is the material conductivity, and s is the heat generate rate per unit volume. The following boundary conditions are considered in this work:

(1) Dirichlet $T = T_1(x, y, z)$ on Γ_1 (7.2)

(2) Neumann $q = -q_s$ on Γ_2 (7.3)

(3) Convection $q = -h(T - T_{amb})$ on Γ_3 (7.4)

(4) Radiation $q = -\varepsilon\sigma(T^4 - T_{amb}^4)$ on Γ_4 (7.5)

where q_s is the external heat flux, h is the convection coefficient, T_{amb} is the ambient temperature, ε is the surface emissivity constant, σ is the Stefan-Boltzamann constant, andΓ_1, Γ_2, Γ_3 and Γ_4 are sets of surfaces that each of these boundary conditions are applied on.

By multiplying Eq. (7.1) by a differentiable weight function $\omega(x)$ with $\omega(x) = 0$ on Γ_1 and using the chain rule and divergence theorem, the weak form of the transient heat transfer is obtained as follows:

$$\int_{\Omega} \rho c_p \frac{\partial T}{\partial t} \omega \, dV + \int_{\Omega} (\nabla \omega) \cdot (k \, \nabla T) dV - \int_{\Omega} s\omega \, dV - \int_{\Gamma_2} (q_s \cdot n)\omega \, dA$$

$$+ \int_{\Gamma_3} h(T - T_{amb})\omega \, dA + \int_{\Gamma_4} \varepsilon\sigma(T^4 - T_{amb}^4)\omega \, dA = 0 \qquad (7.6)$$

$$\forall \omega(x) \; with \; \omega(x) = 0 \; on \; \Gamma_1$$

where $T = T_1(x, y, z)$ on Γ_1.

To discretize the domain into elements, shape functions and their derivatives are used on temperature T and weight function $\omega(x)$, which result in:

$$T^e = [N^e][L^e]\{T\}, \qquad \omega^e = [N^e][L^e]\{\omega\} \qquad (7.7)$$

$$T^e = [N^e][L^e]\{T\}, \qquad \omega^e = [N^e][L^e]\{\omega\} \qquad (7.8)$$

where $[N^e]$ is the matrix of shape function, $[B^e]$ is its derivative for each element, $[L^e]$ is the gather matrix, $\{T\}$ is the vector of temperatures at the nodes, and $\{\omega\}$ is the vector of weight function values. The discretized form of Eq. (7.6) can be written as:

$$
\begin{aligned}
&\underbrace{\left(\sum_{e=1}^{n_{el}}[L^e]^T\underbrace{\int_{\Omega^e}\left(\rho^e {c_p}^e\,[N^e]^T[N^e]\right)dV}_{[\mathbf{M^e}]}[L^e]\right)}_{[\mathbf{M}]}\frac{\partial\{T\}}{\partial t}\\
&\quad+\underbrace{\left(\sum_{e=1}^{n_{el}}[L^e]^T\underbrace{\int_{\Omega^e}(k^e\,[B^e]^T[B^e])dV}_{[\mathbf{K^e}]}[L^e]\right)}_{[\mathbf{K}]}\{T\}\\
&\quad-\underbrace{\sum_{e=1}^{n_{el}}[L^e]^T\underbrace{\int_{\Omega^e}(s^e\,[N^e]^T)dV}_{\{\mathbf{R_G^e}\}}}_{\{\mathbf{R_G}\}}+\underbrace{\sum_{e=1}^{n_{el}}[L^e]^T\underbrace{\int_{\Gamma_2^e}({q_s}^e[N^e]^T)dA}_{\{\mathbf{R_F^e}\}}}_{\{\mathbf{R_F}\}}\\
&\quad+\underbrace{\sum_{e=1}^{n_{el}}[L^e]^T\underbrace{\int_{\Gamma_2^e}\left(h^e[N^e]^T([N^e][L^e]\{T\}-\{T_{amb}^e\})\right)dA}_{\{\mathbf{R_C^e}\}}}_{\{\mathbf{R_C}\}}\\
&\quad+\underbrace{\sum_{e=1}^{n_{el}}[L^e]^T\underbrace{\int_{\Gamma_2^e}\left(\varepsilon\sigma[N^e]^T(([N^e][L^e]\{T\})^{\circ 4}-\{T_{amb}^e\}^{\circ 4})\right)dA}_{\{\mathbf{R_R^e}\}}}_{\{\mathbf{R_R}\}}\\
&=0
\end{aligned}
\tag{7.9}
$$

where n_{el} is the number of elements in the domain, superscript 'e' indicates the parameter is associated with element e, ° is the element-wise product operation, [M] is the capacitance matrix, [K] is the conduction matrix, $\{R_G\}$ is the internal heat vector, $\{R_F\}$ is the external flux vector, $\{R_C\}$ is the convection vector, and $\{R_R\}$ is the radiation vector. This equation shows that FEA matrices and vectors can be calculated separately for each element and then assembled into global variables for solving the weak form equation.

The Gauss quadrature method (Golub et al. 1969) is used to simplify the numerical evaluation of integrals in Eq. (7.9). To do so, elements need to be transformed into an isoparametric coordinate system, then the integrals can be calculated by summing up each integrand over the integration points. By applying this transformation, the elemental matrices and vectors defined in Eq. (7.9) can be reformulated as:

$$[\mathrm{M^e}] = \sum_{m=1}^{n_G} \left(\rho c_p [N_m^e]^T [N_m^e] \omega_m |J_m|\right) \tag{7.10}$$

$$[\mathrm{K^e}] = \sum_{m=1}^{n_G} (k [B_m^e]^T [B_m^e] \omega_m |J_m|) \tag{7.11}$$

$$\{\mathrm{R_G^e}\} = \sum_{m=1}^{n_G} (s [N_m^e]^T \omega_m |J_m|) \tag{7.12}$$

$$\{\mathrm{R_F^e}\} = \sum_{n=1}^{n_F} (q_s [N_n^e]^T \omega_n |J_n|) \tag{7.13}$$

$$\{\mathrm{R_C^e}\} = \sum_{n=1}^{n_F} (h [N_n^e]^T ([N_m^e]\{T_n\} - \{T_{amb}\}) \omega_n |J_n|) \tag{7.14}$$

$$\{\mathrm{R_R^e}\} = \sum_{n=1}^{n_F} \left(\varepsilon\sigma [N_n^e]^T \left(\{[N_m^e]\{T_n\}\}^{\circ 4} - \{T_{amb}\}^{\circ 4}\right) \omega_n |J_n|\right) \tag{7.15}$$

where n_G = 8 and n_F = 4 are the number of Gauss quadrature integration points for 8-node hexahedron elements used in this work, ω_m= ω_n= 1 are the weights of integration points and $[N_m^e]$ and $[B_m^e]$ are the shape function and its derivative for 8-node elements in isoparametric coordinate system, $|J_m|$ is the determinant of Jacobian matrix for the transformation from the Cartesian to the isoparametric coordinate system for 8-node elements. $[N_n^e]$ and $|J_n|$ are the isoparametric shape functions and the determinant of the Jacobian matrix for 4-node surfaces.

A forward time integration scheme is used to approximate the temperature derivative as presented hereunder:

$$[\mathrm{M}]\left(\frac{1}{\Delta t}(\{T^{n+1}\}-\{T^n\})\right)=\{\mathrm{R_G}\}-\{\mathrm{R_F}\}-\{\mathrm{R_C}\}-\{\mathrm{R_R}\}-[\mathrm{K}]\{T^n\}$$
$$\{T^{n+1}\}=\{T^n\}+\Delta t[\mathrm{M}]^{-1}[\{\mathrm{R_G}\}-\{\mathrm{R_F}\}-\{\mathrm{R_C}\}-\{\mathrm{R_R}\}-[\mathrm{K}]\{T^n\}] \quad (7.16)$$

where Δt is the time step and $\{T^{n+1}\}$ and $\{T^n\}$ are nodal temperatures at time step $n+1$ and n respectively. The summation of $\{\mathrm{R_F}\}$, $\{\mathrm{R_C}\}$, and $\{\mathrm{R_R}\}$ is called the external flux and $[\mathrm{K}]\{T^n\}$ is called the conduction flux. Further, $\{\mathrm{R_G}\}$ = 0 for DED processes since elements do not generate internal heat.

A common approach to solve Eq. (7.16) more efficiently is to convert [M] into a diagonal matrix. This can be done by considering the summation of each row as the diagonal value (Zhu 2013). This operation, also called lumping, not only will eliminate the need to calculate the inverse

of a large matrix, but also makes the calculations for the temperature of each node independent from other nodes.

7.3. Massively Parallel Computing with CUDA: Execution and Memory Model

While CPUs consist of a few processing cores that have been optimized for sequential and complex processing, GPUs consist of thousands of smaller cores designed for highly parallel tasks. CUDA is an API provided by NVIDIA that enables developers manage devices and memories on both CPUs and GPUs to solve complicated problems efficiently. CUDA can be used through compiler directions, CUDA-enabled libraries, and multiple programming languages such as Fortran, Python, C, C++. The present work is developed with CUDA C/C++ compiler which is included in the NVIDIA CUDA Development Kit 9.

GPUs are mainly made from multiple Streaming Multiprocessors (SMs) with the key components of computing cores, logical and memory operational units, scheduler, and on-chip memories. Each SM can execute hundreds of threads at the same time based on the resources available to them. Kernels are launched in a user-defined grid of thread blocks, where each block can be up to 1024 threads in recent GPUs. Once a kernel is launched, its thread blocks will be schedules to be run on different SMs and will remain on the SM scheduler until its execution completes. SMs execute thread blocks in groups of 32 threads called warps. Ideally, all the threads in a warp execute memory and logical operations concurrently, which would lead to the most efficient utilization of GPU resources (Cheng et al. 2014).

GPUs use a programmable hierarchical memory structure that allows developers to optimize the performance of memory operations using multiple types of memories with different capacity,

latency, and bandwidth. The main memory types in order of decreasing bandwidth are: registers, shared memory, texture memory, local memory, constant memory, and global memory (see **Figure 7.1**).

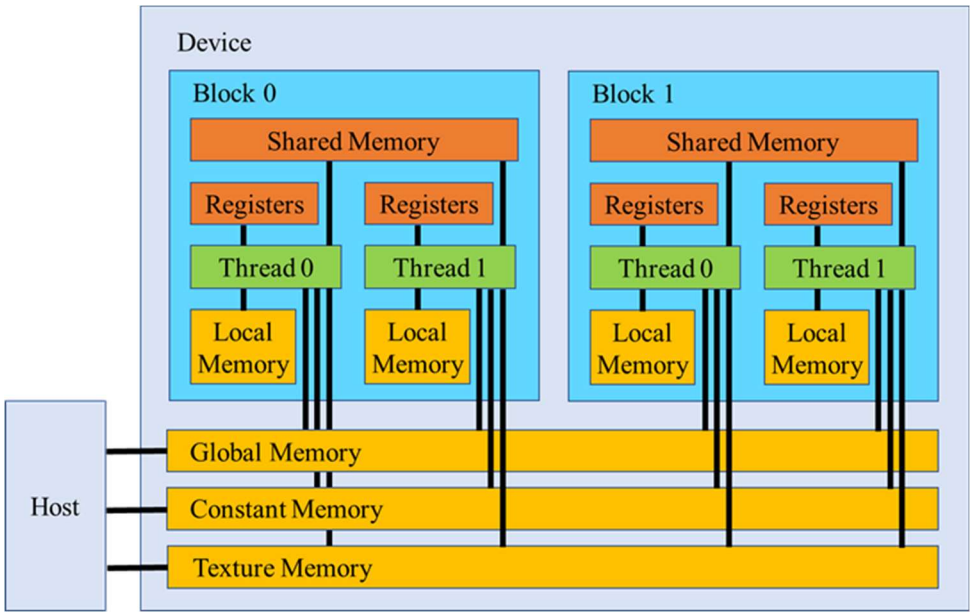


Figure 7.1. CUDA hierarchical memory model; Device (GPU) can communicate with host (CPU) through global, constant, and texture memories, accessible to all threads. Registers and shared memory are low latency memories exclusively visible to a thread and a block respectively (Mozaffar et al. 2019).

7.4. Acceleration Strategies

As seen in Section 7.2, the FEA formulation for the thermal analysis of DED processes discretizes the domain into a large number of elements and performs very similar calculations on them, which makes this routine well-matched with the massively parallel architecture of GPUs. However, some operations for this formulation are not inherently parallel and different calculations need to be done on subsets of elements, nodes, and surfaces, especially considering the dynamic nature of this simulation including birth and death of entities and its advanced boundary conditions,

which impose serious disadvantages for GPU computing. In the following sections, first the computational framework for the thermal analysis of DED processes are introduced and then the challenges associated with the GPU acceleration of DED processes are discussed and acceleration strategies for work distribution, memory management, and optimized data-structures are presented to avoid or mitigate the mentioned challenges.

The overall routine for the thermal analysis of DED processes is demonstrated in **Figure 7.2**, including preprocessing, domain initialization, solver, and outputting steps. The analysis starts with preprocessing the mesh and toolpath files to determine the birth time for each element in the mesh file. Domain initialization creates element, node and surface classes and fills them with information such as element/node IDs, positions, birth and death time, material properties associated with them, and so on. This step is also responsible for assigning the aforementioned boundary conditions to different sections of the mesh and calculating the critical explicit time step to ensure the stability of the simulation. The explicit time stepping is done during the solver step. In this step, the capacitance matrix, the conduction flux, and the external flux are calculated for each element separated and assembled into global matrixes and vectors. In the next time step temperatures for all the nodes are calculated as formulated in Eq. (7.16). Finally, it is necessary to frequently save the results of the simulation into files on the disk.

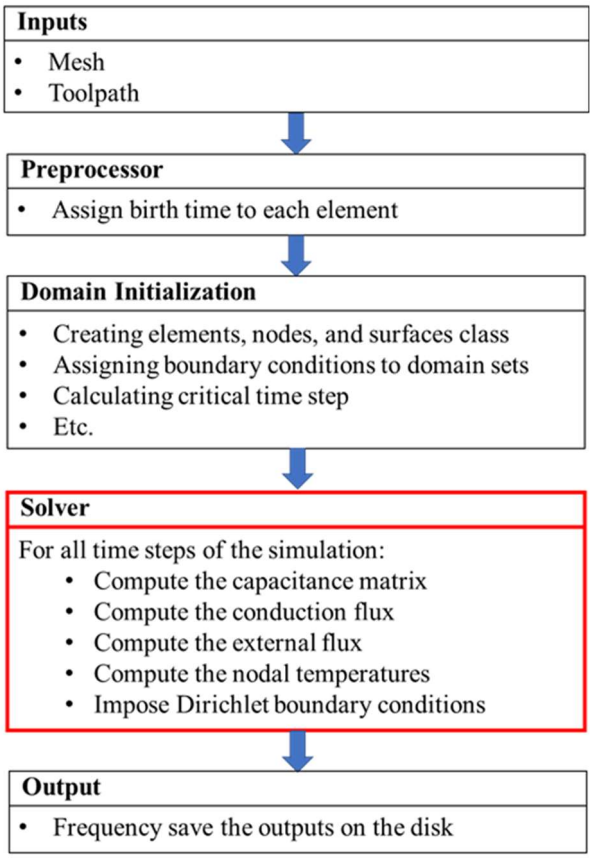


Figure 7.2. Computational FEA framework for the thermal analysis of the DED process; the routine includes preprocessor, domain initialization, solver, and outputting steps with the solver step as the most computationally expensive one (Mozaffar et al. 2019).

Although there is a potential in accelerating the preprocessing and the domain initialization steps, these steps run only once for each simulation and their execution time is relatively negligible compared to the time needed for calculation of the solver steps which repeat for all time steps of the simulation. Therefore, in this work the preprocessing and domain initialization steps are executed on a CPU and the focus of the GPU acceleration is put on the solver step and its efficient interaction with the outputting step.

7.4.1 Assembly Strategy to Avoid Race Condition

An important part of calculating capacitance, conductivity, and flux matrices is assembling, where the contribution of all elements connected to a node should be calculated and summed up to its global variable. However, considering that these elemental calculations are done concurrently, it is possible that multiple elements access the global variable of a shared node between them at the same time and cause a race condition. Race conditions occur when more than one thread attempts to write to a memory location at the same time, in which the output is undetermined. Three assembly strategies similar to the concepts proposed by Cecka et al. (Cecka et al. 2011) and Markall et al. (Markall et al. 2013) are considered in this work. This investigation is unique because of the dynamic nature of FEA analysis of DED processes and the recent advances in both hardware and software capabilities of GPU computing, which makes the conclusions drawn from the literature not reliable for this application.

As the first strategy, the data structure shown in **Figure 7.3** is considered in this work that assigns a separate memory location for each element connected to a node. Using this data structure, each element will write its contribution to a unique memory and avoid the race condition. The assignment of this unique location is done using **Algorithm 7.1** and stored in global memory. A separate kernel is used for reducing node-element data structure by assigning the work of summing all contributions associates with each node to a thread.

As it can be seen from **Figure 7.3B**, the node-element structure contains unused spaces, which may cause inefficient use of global memory especially if the mesh structure contains nodes that are connected to a large number of elements. This problem associated with unequal connected elements can be solved by packing the subarrays for each global node into rows, or bins, of another array using a bin packing algorithm (Lee et al. 1985), or a more efficient packing algorithm such

as the Largest-Processing-Time (LPT) (Graham 1969). However, since the current work is focused on the acceleration of the process, this inefficiency in global memory storage is not investigated.

Thread 0
Thread 1

Node ID	Global Capacitance
1	E1, E2
2	E1, E3, E4,E5
3	E1, E2, E3, E5
4	E2, E3, E4, E6
5	E3, E6, E8
6	E4

(a)

Node ID	Elemental Contributions to Global Capacitance			
1	E1	E2	0	0
2	E1	E3	E4	E5
3	E1	E2	E3	E5
4	E2	E3	E4	E6
5	E3	E6	E8	0
6	E4	0	0	0

(b)

Figure 7.3. Assembly strategies for global capacitance; (a) direct assembly to global capacitance causes may cause a race condition, while (b) the node-element data structure considers separate placeholders for contribution of each element to a node solves this issue (Mozaffar et al. 2019).

Algorithm 7.5: Arrange unique ID for node-element data structure

5. Initialize $A(n_e, n_i)$ to a zero matrix, where n_e and n_i are the number of elements and nodes in an element respectively and A is the matrix containing the node-element unique IDs
6. Initialize $I(n_g)$ to a zero matrix, where n_g is the number of global nodes
7. Loop over all the elements e
    a. nodes ← get nodes in the element e
    b. Loop over the nodes n
        i. j ← get the global index of the node n
        ii. $A(e, n) = I(j)$
        iii. Increment $I(j)$ by one to generate a unique ID next time an element accesses this node
    c. End loop over the nodes
8. End loop over the elements

For the second assembly strategy, the global matrices calculations are divided into smaller subtasks, where each subtask is responsible for the calculations of a predefined group of elements. By choosing the predefined groups in a way that no two elements in the same group have any shared nodes and performing the calculations for each group sequentially, one ensures that

contributions of the elements to any node are written to their allocated memories at different times and, therefore, avoid the racing condition. This strategy is known as coloring the mesh as one can arrange the groups by assigning different color codes to the elements in a way that no two adjacent elements have the same color. A disadvantage of this approach is that arranging the colors adds a significant overhead to the domain initialization step of the analysis.

The third approach is to use Atomic operations to perform the assembly. Atomic operations are special types of read-modify-write actions that allows memory addresses to be accessed by only one thread at a time (NVIDIA 2008). In the literature, the other alternatives provide better acceleration than Atomic operations because of the steep performance cost associated with them. However, recent advances in GPUs with compute capability of 3X or higher improved the performance of Atomic operations. Thus, it is important to reinvestigate the use of Atomic operations for the assembly.

7.4.2 Mitigating Warp Divergence

Another issue that might severely affect the performance of explicit FEA for DED processes is warp divergence. Unlike CPUs that use complex branch prediction, GPUs have a simpler flow control mechanism that tries to execute the exact same instructions for all threads in a warp simultaneously. Executing instructions such as an if-else condition causes the if block and the else block to be performed in sequence instead in parallel, which adversely affects computational performance.

Explicit FEA for DED processes inherently causes such conditional statements. One major source of the conditional statements is the fact that elements and surfaces might get born or die as

the time of the simulation goes on due to the nature of the process that deposits new elements while building the part. To mitigate this issue, both elements are surfaces are sorted based on their birth time in the domain initialization step and by adjusting kernel execution boundaries one can control the range of birth time associated with elements and surfaces that are accessible by each kernel. Considering that the elements stay activated after their birth time, the kernel execution boundaries are dynamically updated in each time step to only perform the kernels on the active elements. The kernel boundary update is implemented using a binary search to efficiently locate the last active element for any time step.

Surfaces can die after their birth time in the FEA for DED processes because an active surface for flux calculations should be on the exterior of the build and the exterior changes dynamically in the building process. Therefore, the boundary update of kernels associated with surfaces is not enough to exclusively select active surfaces. However, the boundary update can bound the kernel execution range from all surfaces to the surfaces with passed birth time. Thus, the surface flux calculations are performed on all surfaces within the boundary of the kernels.

Two strategies are considered to be combined with the dynamic boundary update for surface flux calculations. The first strategy is to minimize the branch divergence by executing the flux calculations on all the surfaces in the execution boundary and canceling the effect of inactive surfaces on $\{R_F\}$, $\{R_C\}$ and $\{R_R\}$ in Eq. (7.16) using a switch. The switch is an integer variable which has the value of 1 for active surfaces and 0 for inactive ones. Using a switch limits the warp branching to only a single operation. The second strategy is to prevent the GPU from performing the calculations for inactive surfaces by using a conditional statement in the beginning of the

kernel. This strategy would result in less computations while inducing a severe branch divergence to warps.

Another source of warp divergence ae the different boundary conditions associated with different subsets of the domain. While the Dirichlet boundary condition is usually applied after calculation of temperatures as shown in **Figure 7.2**, surface flux boundary conditions (i.e., external laser flux, convection and radiation) can be applied on different sets of surfaces. Considering that the majority of the flux operations, such as Jacobian and shape function calculations, are similar for the three types, fluxes are calculated in active surfaces for all three types of boundary conditions and the effect of each boundary condition is controlled by using precomputed switches to avoid warp divergence.

7.4.3 Further Optimization Considerations

Further optimizations applied to this work will be discussed in this section. Most of the device data reside in the global memory and an efficient access to this memory is essential for achieving high bandwidth in data transactions and proper kernel performance. In the CUDA execution model, memory operations are issued per warp. The most efficient access pattern to the global memory of GPUs is aligned coalesced access. In this access pattern, 32 threads in a warp access a contiguous section of memory starting from an even multiple of the cache size (Cheng et al. 2014). In this case, a single memory load/write operation is needed for all the threads in a warp, which will cause 100 percent of bus utilization. Using uncoalesced or non-aligned data structures would cause the same memory load/write to be done with multiple separate operations.

The data associated with different nodes of an element are normally stored next to each other as demonstrated in **Figure 7.4A**. While executing kernels on elements, the GPU warp scheduler will try to execute a single task, for example, calculating the capacitance matrix, for hundreds of elements at the same time. Therefore, all threads in a warp will run memory access for the same index node of all the elements together. This access pattern will cause a significant efficiency penalty due to uncoalesced access.

To maximize the efficiency of global memory reads and writes, data is rearranged in the domain initialization step to access elemental matrices and vectors in a coalesced manner as depicted in **Figure 7.4B**. A similar rearrangement is applied for all element, node, and surface global variables such as the nodal coordinates, the connectivity matrices, the element capacitance and conductivity matrices, and nodal temperatures to ensure efficient global memory transactions. While using the coloring assembly strategy this rearrangement should be done for each color separately.

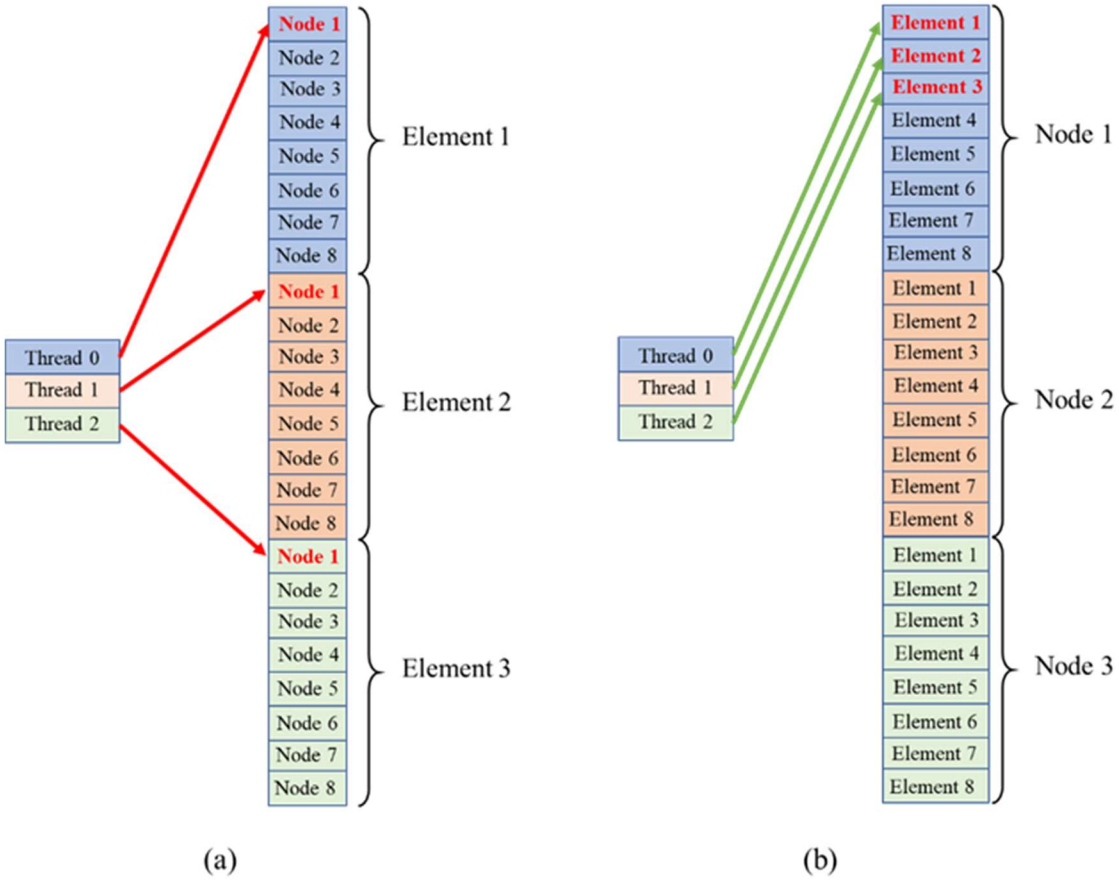


Figure 7.4. Global memory access pattern; (a) an uncoalesced access pattern is caused when threads access memories of nodal data for different elements, and (b) a coalesced memory access pattern achieved by rearranging data based on their kernel access (Mozaffar et al. 2019).

Efficient use of memory hierarchy in GPUs increases computation performance by maximum low-latency, high-bandwidth memory usage. Specifically, constant memory and shared memory are used in the present work to decrease the number of registers used in each kernel and avoid spilling registers into local memory, which has a high latency. Constant memory is used for accessing material properties such as density, solidus and liquidus temperatures, specific heat, etc. Since all the threads will load these variables together, the constant memory will broadcast the corresponding values to all the threads at the same time and cause a desirable access pattern.

Shared memory is used for calculating the shape function and the Jacobean of each element. This is because these variables are called many times inside the kernels and having them in the lowest latency memories are essential while keeping them in registers will use too many registers in each block and limit the number of warps that can be executed in each block.

Another major acceleration consideration implemented in this work is to asynchronously lunch kernels using CUDA streams to overlap calculations done on the CPU and GPU, overlap data transfer and kernel execution, and concurrent execution of GPU kernels. As mentioned before, kernel execution boundaries need to be calculated before execution of each kernel on the CPU. By overlapping these calculations with previous kernel execution both the CPU and GPU can work at the same time to completely hide the time needed for CPU calculation. Overlapping data transfer with kernel execution can decrease the time needed for saving the simulation outputs by concurrently copying data from GPU global memory to CPU accessible memory (RAM) while continuing the calculation of the next time step. Finally, concurrent execution of GPU kernels will increase the device occupancy by increasing the number of warps scheduled to be run. The asynchronous execution of kernels on different CUDA streams and the overlap between data transfer and device calculations is demonstrated in **Figure 7.5**, which is the output of the NVIDA Visual Profiler tool.

As shown in **Figure 7.5**, the data transfer between the GPU and CPU memories is performed concurrently with initialization of the FEA matrices, which means that this data transfer does not add significant overhead to the simulation time and can be performed as frequently as desired and other calculations can be carried on the data on the CPU side. This is particularly useful for

calculations that are inherently conditional or not suitable to be performed on the GPU such as computing cooling rates.

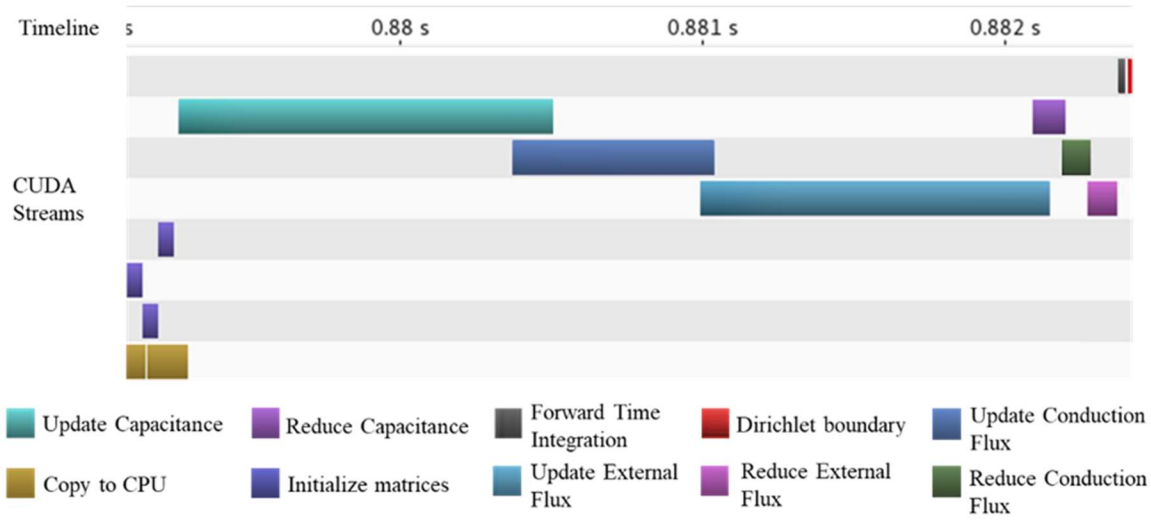


Figure 7.5. visualization of asynchronous kernel execution using NVIDIA Visual Profiler; rows represent different CUDA streams and each color represent a kernel execution (Mozaffar et al. 2019).

7.5. Acceleration Results and Verification

To test the acceleration strategies discussed previously, four samples of DED processes are investigated. The samples include three LENS builds of a cubic, a cruciform, and a thin-wall and a powder-bed SLM build. The mesh and geometry of the samples are demonstrated in **Figure 7.6**, where the blue meshes represent the substrate and the red meshes represent the build. The difference between the simulation setup of the SLM process and LENS processes is that for SLM, an entire layer of elements is born at the same time, while for LENS, the elements are born gradually following the laser focal point.

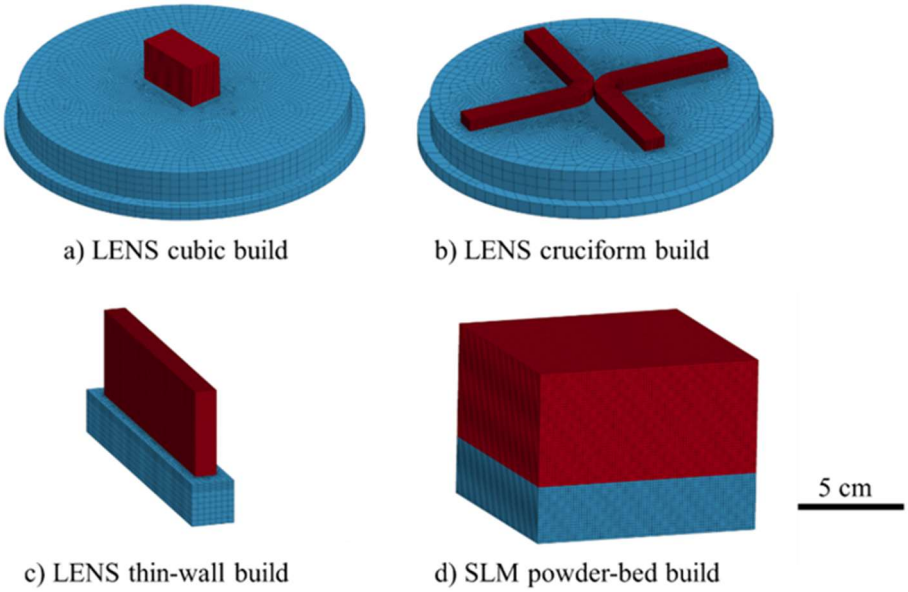


Figure 7.6. Geometries and meshes of the test samples where blue meshes represent the substrate and red meshes represent the build for a) LENS cubic, b) LEN cruciform, c) LENS thin-wall, and d) SLM powder-bed geometries (Mozaffar et al. 2019).

The simulation parameters of the samples considered to verify the proposed algorithms include the number of elements in the range of around 80,000 to 400,000 elements, simulation time in the range of 42 to 3,007 seconds, and the stainless steel 316L and Titanium alloy Ti-6Al-4V materials as provided in **Table 7.1**.

Table 7.1. Summary of simulation parameters for the test samples (Mozaffar et al. 2019).

	Cubic	Cruciform	Thin-Wall	Powder-Bed
Number of Elements	84,346	205,618	193,944	384,000
Number of Nodes	93,748	232,447	210,000	400,221

Minimum Time Step	9.78e-4 s	1.96e-3 s	1.69e-2 s	1.38e-3 s
Simulation Time	1,195s	3,007s	2,741s	42s
Material	SS316L	SS316L	Ti-6Al-4V	SS316L
Laser Power	1,050 W	1,050 W	1,500 W	120 W
Hatch Spacing	1.1 mm	1.1 mm	1.9 mm	0.5 mm

To determine the effect of assembly strategy on acceleration, the three mentioned GPU assembly approaches are used to simulate each sample and they are compared with an optimized single CPU implementation of the same calculations. The optimized CPU implementation considers elements as non-deformable and material properties as fixed values. Using these simplifying assumptions, the CPU implementation calculates element and surface Jacobians as well as the element local conduction matrices only once and uses the stored values at each time-step. This implementation is used to be able to simulate the samples in a feasible time frame since the version without simplification is an order of magnitude more computationally expensive. However, all GPU implementations perfume these calculations at each time-step which makes them suitable for simulation with deformable elements and temperature dependent material properties.

The node-element data structure GPU implementation output for the temperature field of the samples during the build is visualized in **Figure 7.7**, in which the range of color bars is set from

300 K to the liquidus temperature of the material; therefore, the red color region represents the melt pool.

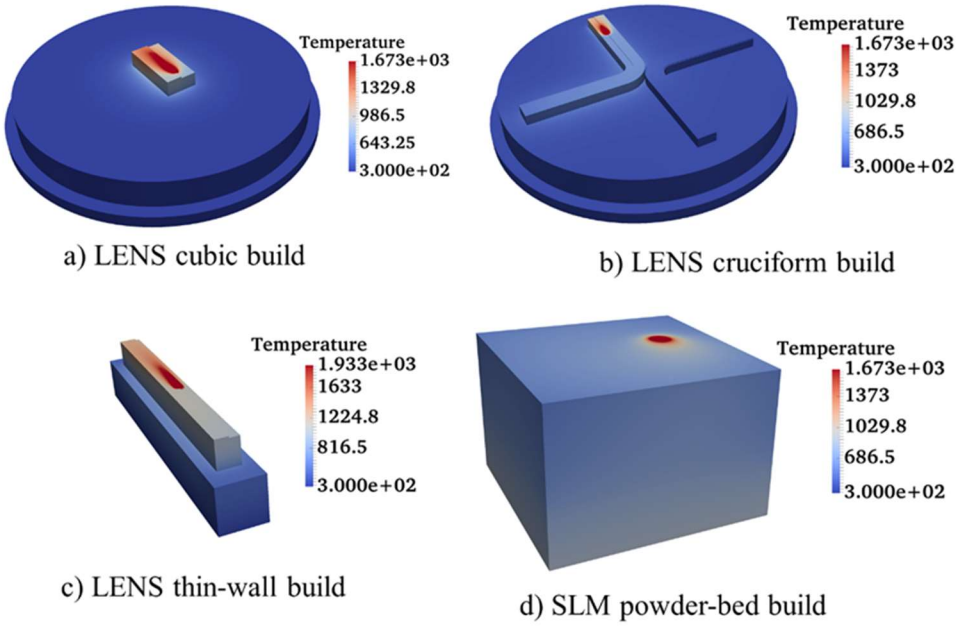


Figure 7.7. Visualization of the test simulation outputs for a) LENS cubic, b) LENS cruciform, c) LENS thin-wall, and d) SLM powder-bed builds (Mozaffar et al. 2019).

The result of the simulation for the optimized single CPU and GPU enabled implementations for the assembly strategies is provided in **Figure 7.8**. The results are produced using the NVIDIA GeForce GTX TITAN Black graphics card, which has 2880 CUDA cores, bus support of PCI Express 3.0, 6 GB of global memory, and compute capability of 3.5. The CPU used for this work is an Intel(R) Xeon(R) CPU E5-2687W with the clock speed of 3.10 GHz.

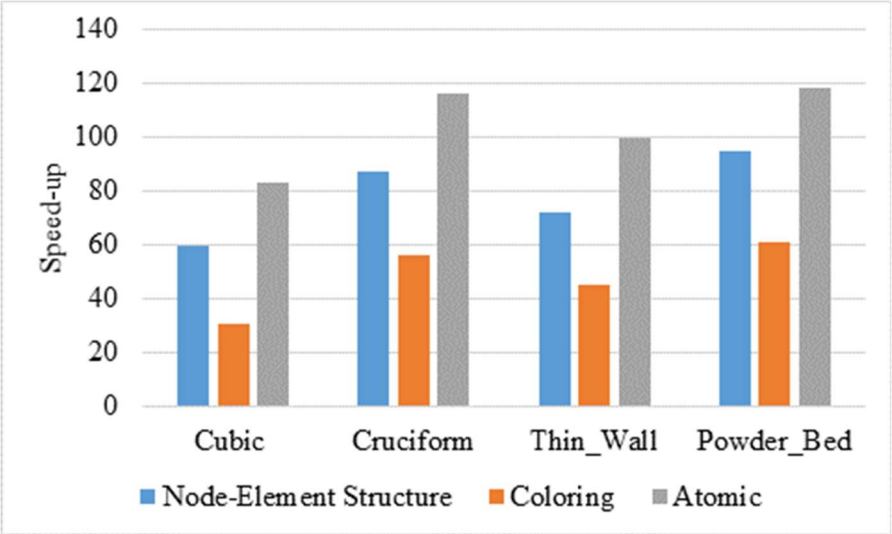


Figure 7.8. Acceleration results of the assembly strategies for test samples (Mozaffar et al. 2019).

Although many factors can affect the speed-up of a simulation such as the frequency of outputting, the distribution of boundary conditions between surfaces, and the birth strategy of the build, these results indicate a correlation between the size of the simulations, which can be represented by the number of elements or nodes, and the speed-up, which is demonstrated in **Figure 7.9**. This is because the more elements and nodes the model has, the more parallel works exist for the GPU and the overall simulation becomes more suitable for the massively parallel architecture of the GPU.

The thin wall and cruciform builds have a similar number of nodes, but there is a significant difference between their speed-ups. As it can be seen in **Figure 7.6**, a larger portion of the total nodes of the geometry is associated with the build in the thin-wall simulation with respect to the cruciform simulation. Considering that the simulation works on active elements and nodes during each time step, the effective size of the simulation for the thin-wall is significantly smaller than

for the cruciform build at the beginning time steps, which is one reason for the better performance of the cruciform build.

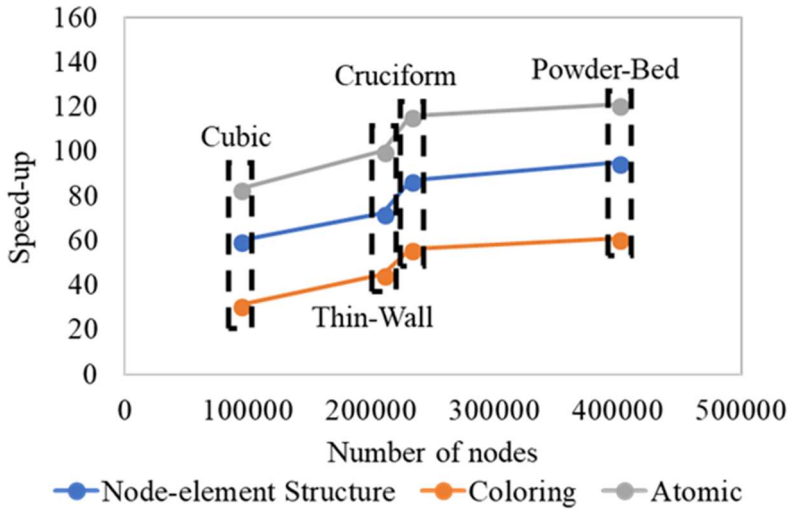


Figure 7.9. Correlation between the number of nodes and the achieved speed-up in test samples (Mozaffar et al. 2019).

The assembly strategy using coloring leads to the worst performance between the investigated approaches. This is because the kernel execution is done on each color separately and the size of the domain visible to the kernel is only a fraction of it. As already seen, this reduction in the domain size severely affects the performance, especially considering that only a portion of the domain is born at any time step. Also, the calculations of the dynamic boundary conditions need to be performed for each color. The strategy using the node-element data structure out-performs the coloring approach. However, since this strategy needs to access and store data structures with multiple sizes of the assembled vector, the calculations are significantly bounded by the memory access operations to global memory. Further, this strategy requires an additional reduction kernel that increases the execution time. The strategy using Atomic operations consistently provides the best performance for all the samples. This is because racing condition does not happen in the

assembly procedure for all the nodes since the physical execution of warps can happen at separate times. Using the Atomic operations allows the GPU to halt only the memory operations with racing conditions and avoid costly explicit synchronization. This capability is particularly improved in recent GPUs with compute capability of 3 or higher. The Atomic strategy not only has lower execution time with respect to the other strategies, it also requires the least amount of preprocessing and global memory storage.

The results of the flux calculation strategies are provided in **Figure 7.10** using the Atomic assembly strategy. The results show that for the LENS test samples using the least kernel computation strategy would lead to a significant increase in the performance with respect to the least warp divergence strategy, while the two strategies perform almost identically for the SLM test sample. This is because that gradual generation of elements from the laser focal point, for LENS processes, generates more external surfaces to be born and die during the simulation than the generation of a whole layer of elements together, for the SLM process. This result indicates that in the case of powder-bed simulations the cost associated with warp divergence and redundant kernel computations are balanced. However, in the case of LENS processes which have more dynamic surfaces the cost of excessive redundant calculations exceeds the warp divergence penalty. Therefore, considering the large number of surface births and deaths in DED processes, it is beneficial to use conditional statements to avoid redundant calculations for inactive surfaces. This benefit is more important in LENS processes due to the greater number of intermediate surfaces.

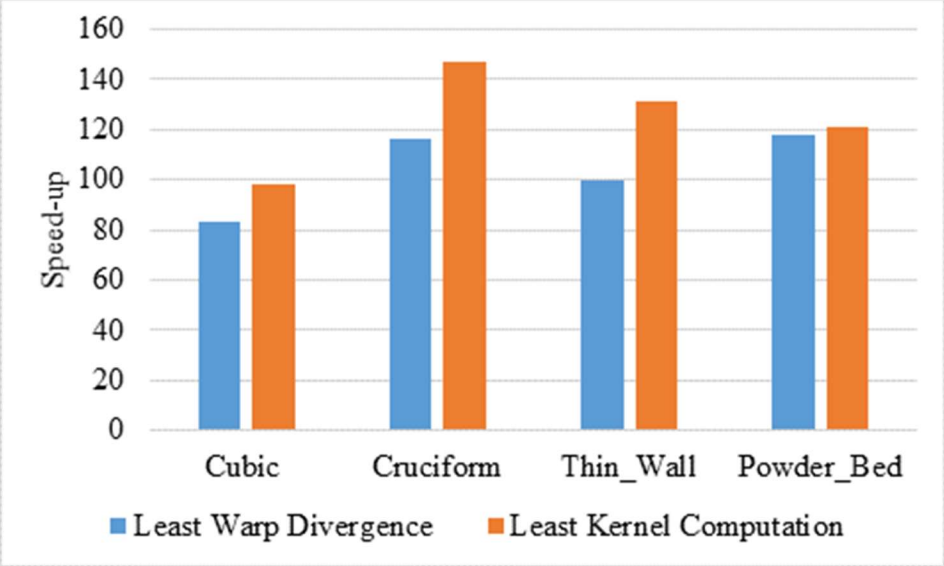


Figure 7.10. Acceleration results of the flux calculation strategy for test samples (Mozaffar et al. 2019).

The accuracy of the results is validated by comparing the temperature outputs for the GPU implementations and the CPU one. This validation is demonstrated on a NU-shape build with nearly 200,000 nodes and 150 s of the simulation time as depicted in **Figure 7.11A**, where the yellow cross represents the probe point. A comparative figure for the output temperature of the prob point calculated on the CPU and the GPU implementation with Atomic operation and the least kernel computation strategies as shown in **Figure 7.11B**, which verifies the accuracy of the GPU calculations. The mean absolute error of different strategies is summarized in **Table 7.2** for the NU-shaped sample, which indicates the correctness of the calculations presented in this work considering that operations are done on 32 bits floating point numbers with 6 significant decimal digits.

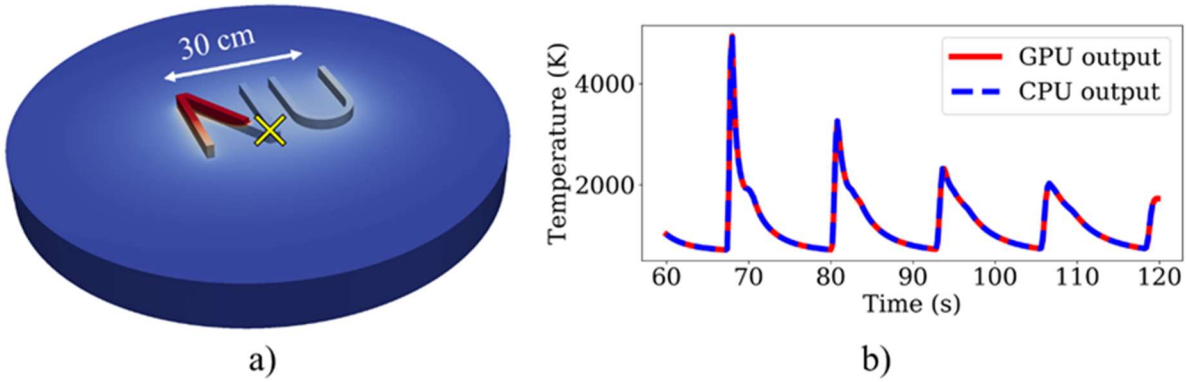


Figure 7.11. Validation of the accuracy of the GPU calculations; a) a demonstration of the test geometry and a screenshop of its thermal profile during the build, where the yellow cross represents the probe point, and b) the comparison between the GPU and CPU outputs for the thermal hisotry of the prob point (Mozaffar et al. 2019).

Table 7.2. Accuracy of the GPU strategies with respect to the CPU calculations for the NU-shape build (Mozaffar et al. 2019).

Strategy	Node-element structure and least divergence	Atomic and least divergence	Coloring and least divergence	Atomic and least computation
Mean Absolute Error	8.538e-6	7.088e-6	9.360e-6	7.096e-6

7.6. Conclusions and Future Work

In conclusion, this research presents a methodology for accelerating the FEA calculations for explicit thermal analysis of DED processes. Different strategies to avoid race conditions and warp divergence and maximize memory access efficiency are discussed and their advantages and disadvantages for complex boundary conditions in DED processes investigated. Further, memory hierarchy and host-device concurrency are used to optimize the use of GPU resources. The

implementations are tested on multiple DED processes which led to speed-ups of about 98-147 X with respect to an optimized CPU implementation for the strategy of assembling by using Atomic operations and flux calculations using the least kernel computation approach along with the proposed optimization.

The developed FEM simulation can partially model the physics involved in the AM process. In the future, more attempts will be dedicated to adding multi-physics formulations into the GPU accelerated FEM model. In particular, an important coupled physics with the thermal analysis of the AM process is the thermoelastic behavior of the material that generates deformation and residual stresses. Developing thermo-mechanical modeling using explicit and implicit FEM will be investigated. Implicit solutions can be interesting in this case because the critical time-step of the explicit solution can become problematic in multi-physics problems. The thermo-mechanical FEM model of the AM processes can be formulated as a semi-coupled simulation where the displacement field is coupled with the temperature while the temperature field is independent. Furthermore, we will attempt to calibrate the two proposed physics-based models using experimental data from the DED setups mentioned in the previous tasks.

The scalability of the developed FEM package on multi-GPU clusters is another interesting subject that needs further investigation. Considering that GPU clusters are becoming increasingly popular, the scalability performance of the model is critical to problems that require large memory or computational power. There are many communication protocols and memory copy operations (e.g., host side communications, uniformed virtual addressing, peer-to-peer memory copy, etc.) that can be used for sharing and transmitting data between GPU nodes. The algorithms and strategies to obtain optimal performance on such clusters and the trade-off between internal

calculation and data transition are important subjects that need to be studies before these algorithms can be widely used in industry.

CHAPTER 8

Contributions and Future Directions

Manufacturing sciences has experienced significant innovations over the past decades that have enabled massive design freedom even for low-volume productions such as AM and ISF. This flexibility comes with a cost of more complex material behavior and accumulative multi-scale responses as the result of the highly localized interaction between tooling and materials. In the meanwhile, advances in machine learning, data acquisition systems, and networking platforms created an opportunity to utilize the data from manufacturing processes and drastically improve our understanding of the behavior of advanced materials, the influence of manufacturing operations on them, and computational design methods to fully exploit manufacturing capabilities. However, these tasks necessitate intricate developments of computational methods that can (i) integrate knowledge from various sources including theory, experimental, and simulation data, (ii) handle the dynamical, noisy, and unstructured nature of manufacturing data, (iii) scale to high-dimensional spaces with large design spaces, and (iv) mitigate the computational costs and numerical problems associated with computational mechanics.

Motivated by the above-listed challenges, my Ph.D. thesis was devoted to investigating physics-informed artificial intelligence-based predictive models, design tools, and accelerated analysis of metal-based additive manufacturing and forming processes. The research involved several interdisciplinary areas including multi-scale and multi-physics computational modeling, data-driven supervised, unsupervised, and reinforcement learning, high-performance computing, and

cyber-physical systems. In this Chapter, I discuss my novel contributions to the field and elaborate on my insights on impactful future research directions.

8.1. Contributions

Several novel contributions are introduced to the research field as the result of my dissertation study. I developed several methods that utilize artificial intelligence and computational mechanics to enhance current manufacturing capabilities. My major research accomplishments are summarized in the following subsections in two areas, namely, predictive modeling and design methods. I hope that my research presents a significant step forward towards smart and agile manufacturing systems.

8.1.1. Contributions in manufacturing process modeling

- ***Data-driven modeling of thermal responses using recurrent cells (Chapter 3):*** The thermal response of AM is a pivotal characteristic of this process and involves complex spatio-temporal patterns. My work shows that a data-driven recurrent structure can effectively learn an update parameter for the temperature of each nodal point given process parameters, such as laser power, boundary conditions, and geometric features. The data-driven model can roll out simulation results for an arbitrary number of time steps and shows a limited error propagation over time. On the contrary, in my experience, extracting geometric features based on manual feature engineering leads to limited generalizability of the data-driven solution when facing complex parts.

- ***Geometry-Agnostic thermal predictive modeling using graph networks (Chapter 3):*** Extracting meaningful geometric features from a freeform shape is an unsolved problem where current solutions do not scale to the complexity of geometries in AM. To address this issue, I integrated a graph-based feature extraction network with a recurrent neural network structure. In this approach, an end-to-end differentiable model is trained where a graph network, generated from the mesh, extracts neighboring correlations based on the nodal values and their distance while a recurrent structure predicts long-term time-series correlations. My results indicate that this approach leads to an order of magnitude smaller error propagation compared to alternative methods while showing an excellent generalization behavior to unseen industrial-grade geometries.

- ***Constitutive modeling of material elasto-plastic behavior under arbitrary loading (Chapter 4):*** Conventional plasticity involves reducing the stress and strain behavior of materials to effective parameters and utilizing a combination of phenomenological laws and theory-driven formulations to solve for stress updates as the result of loading. In my work, I show that this relationship can be captured by a data-driven method in an accurate and computationally efficient manner. Particularly, I address two critical questions in plasticity data-driven modeling for (i) predicting history-dependent plastic behavior and (ii) capturing nonlinear correlations between material microstructural descriptors and strain paths by proposing a customized recurrent neural network structure. Interestingly, it was shown that this approach is robust enough to track the yield surface evolution without being explicitly trained for. While the initial focus was on composite materials, later, in a collaborative work, it was demonstrated that this

methodology can be expanded to metal alloys especially in sheet metal forming processes. Therefore, this approach works across a wide range of material models and can be used as a unifying interface for material development.

- ***GPU accelerated finite element computing in AM (Chapter 7):*** The computational cost of manufacturing simulations is a key limiting factor in the development and utilization of such technologies. Therefore, developing new methods using heterogeneous computing hardware such as GPUs and TPUs to accelerate computations is an impactful step toward advancing current capabilities. In my research, I developed a GPU accelerated method for finite element analysis of AM processes including DED and SLM. My study shows that the strategy based on atomic operations for matrix assembly and minimum wrap divergence leads to better results compared to previously proposed methods, leading to $100-150$X speedup compared to an optimized implementation on CPU.

8.1.2. Contributions in manufacturing process design

- ***Model-free reinforcement learning framework for toolpath design in AM (Chapter 5):*** Toolpath strategies in AM processes can significantly affect the microstructural and mechanical behavior of the builds. However, in current industrial practices, simple toolpaths such as raster or boundary contours are deployed. My research proposed a new vision in which the toolpath can be used as a source of design customization. I formulated the toolpath design problem as a reinforcement learning method and investigated the effectiveness of model-free methods to design toolpaths under two cases of available dense and sparse reward structures. This research shows that model-

free methods with the proposed modifications can design compelling geometry-agnostic toolpaths in the presence of dense reward structures.

- ***Model-based reinforcement learning method for toolpath design in AM (Chapter 5):*** As an alternative approach, I investigated a model-based reinforcement learning method for the toolpath design task. The model-based method based on the Muzero algorithm shows unique advantages to its model-free counterparts such as model reusability and sample efficiency. In this approach, several neural networks are trained for estimating value, reward, policy, and dynamics which enable one to construct and prune a Monte Carlo Tree Search at each step of the simulation and select the best actions by looking into tens of steps of future possibilities. The results show that the performance of the model-free methods can be surpassed with close to 3,000 simulation episodes.

- ***Differentiable simulation tools for design in manufacturing (Chapter 6):*** Integrating physics-based and data-driven methods have been an increasingly popular scientific pursuit. I developed a differentiable physics-based manufacturing simulation tool and seamlessly integrated it with data-driven methods (i.e., neural networks). Using this method, one can optimize high-dimensional process parameters of manufacturing processes using gradient-based optimizers. Particularly, I show that this method is effective in the design of time-series laser inputs for achieving an arbitrary thermal profile and melt pool depth during AM processes.

8.2. Directions for Future Research

In this final section, I describe my vision of interesting future directions in the field of data-driven methods in the prediction and design of manufacturing processes. I believe many critical manufacturing challenges in quality variability, process efficiency, and high-dimensional design for tailored materials and geometric properties can greatly benefit from recent advances in machine learning and high-performance computing with the potential to drastically alter the capabilities of multi-billion-dollar industries.

- ***Hierarchical physics-aware data-driven modeling in manufacturing:*** While we observe an increasing role of data-driven modeling in manufacturing, current methods are limited to single and often straightforward aspects of the processes. To advance these technologies further, we need new frameworks and algorithms to process the data we collect from manufacturing along with the prior knowledge and known physics of the process. In particular, building hierarchical methods to detect and connect material evolution at different length scales would enable a deeper understanding of the process and unlock unprecedented design capabilities. However, there are many fundamental questions from theoretical to applied topics yet to be answered. Developing new variations of methods that combine data-driven and theory-driven approaches is an active research area. It is important to develop frameworks that transfer information across different scales. Stability issues in dynamical systems, when combined with using data-driven methods, need to be resolved. Successful platforms to increase the explainability of such models would significantly impact their industrial adoption.

- ***Design in sparse settings:*** Many of current manufacturing solutions stem from rules of thumb and legacy practices, which create a substantial opportunity for optimization in the field. This is especially pronounced in many flexible manufacturing processes with many degrees of freedom in time and space to control the material deformation, flow, thermal properties, etc. However, meaningful signals to guide this optimization problem are often few and far between, which makes conventional methods ineffective. I believe advancements in explorations and planning methods to tackle sparse design signals are the necessary steps for moving forward. Furthermore, robust frameworks for knowledge distillation are needed to increase the sample efficiency of the current methods.

- ***Integrated experimental and simulation platforms:*** As most AI methods are extensively data-intensive, developing useful models based on experimental data is not feasible in most manufacturing applications. Furthermore, controlling the distribution of experimental data is expensive and, in some cases, unrealistic, which can cause problematic bias in the model. Combining experimental and simulation data is an exciting way to resolve these issues. The data-driven methodology is a great platform for developing such integrated systems because it allows one to complement the information from different sources. For example, simulation data can be used in places where experimental data is unavailable. One interesting aspect of integrated platforms is that they allow designing robust systems, where the outliers and special cases are over-represented using high-performance simulation tools. As an example, we can over-represent rare keyhole formation patterns to ensure the integrated system is properly informed about them.

REFERENCES

A. S. T. M. (2012). Standard terminology for additive manufacturing technologies. ASTM International F2792-12a.

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean and M. Devin (2016). "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467.

Akram, J., P. Chalavadi, D. Pal and B. Stucker (2018). "Understanding grain evolution in additive manufacturing through modeling." Additive Manufacturing **21**: 255-268.

Armstrong, P. J. and C. O. Frederick (1966). A mathematical representation of the multiaxial Bauschinger effect, Central Electricity Generating Board [and] Berkeley Nuclear Laboratories~….

Arruda, E. M. and M. C. Boyce (1993). "A three-dimensional constitutive model for the large stretch behavior of rubber elastic materials." Journal of the Mechanics and Physics of Solids **41**(2): 389-412.

Barlat, F., J. J. Gracio, M.-G. Lee, E. F. Rauch and G. Vincze (2011). "An alternative to kinematic hardening in classical plasticity." International Journal of Plasticity **27**(9): 1309-1327.

Barlat, F., J. Ha, J. J. Grácio, M.-G. Lee, E. F. Rauch and G. Vincze (2013). "Extension of homogeneous anisotropic hardening model to cross-loading with latent effects." International Journal of Plasticity **46**: 130-142.

Barlat, F., D. J. Lege and J. C. Brem (1991). "A six-component yield function for anisotropic materials." International Journal of Plasticity **7**(7): 693-712.

Bastien, F., P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio (2012). "Theano: new features and speed improvements." arXiv preprint arXiv:1211.5590.

Baturynska, I., O. Semeniuta and K. Martinsen (2018). "Optimization of Process Parameters for Powder Bed Fusion Additive Manufacturing by Combination of Machine Learning and Finite Element Method: A Conceptual Framework." Procedia CIRP **67**(1): 227-232.

Belytschko, T., W. K. Liu, B. Moran and K. Elkhodary (2013). Nonlinear finite elements for continua and structures, John wiley & sons.

Bessa, M., R. Bostanabad, Z. Liu, A. Hu, D. W. Apley, C. Brinson, W. Chen and W. K. Liu (2017). "A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality." Computer Methods in Applied Mechanics and Engineering **320**: 633-667.

Bessa, M., R. Bostanabad, Z. Liu, A. Hu, D. W. Apley, C. Brinson, W. Chen, W. K. J. C. M. i. A. M. Liu and Engineering (2017). "A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality." **320**: 633-667.

Bhardwaj, T. and M. Shukla (2018). "Effect of laser scanning strategies on texture, physical and mechanical properties of laser sintered maraging steel." Materials Science and Engineering: A **734**: 102-109.

Bolz, J., I. Farmer, E. Grinspun and P. Schröoder (2003). Sparse matrix solvers on the GPU: conjugate gradients and multigrid. ACM transactions on graphics (TOG), ACM.

Bostanabad, R., A. T. Bui, W. Xie, D. W. Apley and W. Chen (2016). "Stochastic microstructure characterization and reconstruction via supervised learning." Acta Materialia **103**: 89-102.

Bostanabad, R., Y. Zhang, X. Li, T. Kearney, L. C. Brinson, D. W. Apley, W. K. Liu and W. Chen (2018). "Computational microstructure characterization and reconstruction: Review of the state-of-the-art techniques." Progress in Materials Science.

Bourell, D. L., J. J. Beaman, M. C. Leu and D. W. Rosen (2009). "A brief history of additive manufacturing and the 2009 roadmap for additive manufacturing: looking back and looking ahead." Proceedings of RapidTech: 24-25.

Bradbury, J., R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin and S. Wanderman-Milne (2020). "JAX: composable transformations of Python+ NumPy programs, 2018." URL http://github. com/google/jax **4**: 16.

Bruschi, S., T. Altan, D. Banabic, P. Bariani, A. Brosius, J. Cao, A. Ghiotti, M. Khraisheh, M. Merklein and A. Tekkaya (2014). "Testing and modelling of material behaviour and formability in sheet metal forming." CIRP Annals **63**(2): 727-749.

Caiazzo, F. and A. Caggiano (2018). "Laser Direct Metal Deposition of 2024 Al Alloy: Trace Geometry Prediction via Machine Learning." Materials **11**(3): 444.

Cecka, C., A. J. Lew and E. Darve (2011). "Assembly of finite element methods on graphics processors." International journal for numerical methods in engineering **85**(5): 640-669.

Chaari, I., A. Koubâa, S. Trigui, H. Bennaceur, A. Ammar and K. Al-Shalfan (2014). "SmartPATH: An efficient hybrid ACO-GA algorithm for solving the global path planning problem of mobile robots." International Journal of Advanced Robotic Systems **11**(7): 94.

Chaboche, J. L. (1989). "Constitutive equations for cyclic plasticity and cyclic viscoplasticity." International Journal of Plasticity **5**(3): 247-302.

Chaboche, J. L. (1991). "On some modifications of kinematic hardening to improve the description of ratchetting effects." International Journal of Plasticity **7**(7): 661-678.

Chebotar, Y., K. Hausman, M. Zhang, G. Sukhatme, S. Schaal and S. Levine (2017). Combining model-based and model-free updates for trajectory-centric reinforcement learning. Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org.

Cheng, J., M. Grossman and T. McKercher (2014). Professional Cuda C Programming, John Wiley & Sons.

Chiumenti, M., X. Lin, M. Cervera, W. Lei, Y. Zheng and W. Huang (2017). "Numerical simulation and experimental calibration of Additive Manufacturing by blown powder technology. Part I: thermal analysis." Rapid Prototyping Journal **23**(2): 448-463.

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078.

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. J. a. p. a. Bengio (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation."

Chollet, F. (2015). Keras.

Chung, J., C. Gulcehre, K. Cho and Y. J. a. p. a. Bengio (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling."

Collobert, R., K. Kavukcuoglu and C. Farabet (2011). Torch7: A matlab-like environment for machine learning. BigLearn, NIPS workshop.

Defferrard, M., X. Bresson and P. Vandergheynst (2016). "Convolutional neural networks on graphs with fast localized spectral filtering." Advances in neural information processing systems **29**: 3844-3852.

Dehoff, R., M. Kirka, W. Sames, H. Bilheux, A. Tremsin, L. Lowe and S. Babu (2015). "Site specific control of crystallographic grain orientation through electron beam additive manufacturing." Materials Science and Technology **31**(8): 931-938.

Dziekonski, A., A. Lamecki and M. Mrozowski (2011). "A memory efficient and fast sparse matrix vector product on a GPU." Progress In Electromagnetics Research **116**: 49-63.

Dziekonski, A., A. Lamecki and M. Mrozowski (2016). GPU-accelerated finite element method. Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO), 2016 IEEE MTT-S International Conference on, IEEE.

Dziekonski, A., P. Sypek, A. Lamecki and M. Mrozowski (2012). "Finite element matrix generation on a GPU." Progress In Electromagnetics Research **128**: 249-265.

Feigenbaum, H. P. and Y. F. Dafalias (2007). "Directional distortional hardening in metal plasticity within thermodynamics." International Journal of Solids and Structures **44**(22): 7526-7542.

Fey, M. and J. E. Lenssen (2019). "Fast graph representation learning with PyTorch Geometric." arXiv preprint arXiv:1903.02428.

Finn, C. and S. Levine (2017). Deep visual foresight for planning robot motion. 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE.

Fish, J. and T. Belytschko (2007). "A first course in finite elements."

Fisher, B. A., B. Lane, H. Yeung and J. Beuth (2018). "Toward determining melt pool quality metrics via coaxial monitoring in laser powder bed fusion." Manufacturing Letters.

Fisher, B. A., B. Lane, H. Yeung and J. Beuth (2018). "Toward determining melt pool quality metrics via coaxial monitoring in laser powder bed fusion." Manufacturing letters **15**: 119-121.

Fortunato, M., M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis and O. Pietquin (2017). "Noisy networks for exploration." arXiv preprint arXiv:1706.10295.

Fout, A., J. Byrd, B. Shariat and A. Ben-Hur (2017). Protein interface prediction using graph convolutional networks. Advances in neural information processing systems.

François, M. (2001). "A plasticity model with yield surface distortion for non proportional loading." International Journal of Plasticity **17**(5): 703-717.

Francois, M. M., A. Sun, W. E. King, N. J. Henson, D. Tourret, C. A. Bronkhorst, N. N. Carlson, C. K. Newman, T. Haut and J. Bakosi (2017). "Modeling of additive manufacturing processes for metals: Challenges and opportunities." Current Opinion in Solid State and Materials Science **21**(LA-UR-16-24513).

Francois, M. M., A. Sun, W. E. King, N. J. Henson, D. Tourret, C. A. Bronkhorst, N. N. Carlson, C. K. Newman, T. S. Haut and J. Bakosi (2017). "Modeling of additive manufacturing processes for metals: Challenges and opportunities." Current Opinion in Solid State and Materials Science **21**(LA-UR-16-24513).

Geng, L. and R. H. Wagoner (2002). "Role of plastic anisotropy and its evolution on springback." International Journal of Mechanical Sciences **44**(1): 123-148.

Georgescu, S., P. Chow and H. Okuda (2013). "GPU acceleration for FEM-based structural analysis." Archives of Computational Methods in Engineering **20**(2): 111-121.

Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems, O'Reilly Media.

Gibson, I., D. W. Rosen and B. Stucker (2010). Sheet Lamination Processes. Additive Manufacturing Technologies, Springer**:** 223-252.

Girshick, R. (2015). Fast r-cnn. Proceedings of the IEEE international conference on computer vision.

Golub, G. H. and J. H. Welsch (1969). "Calculation of Gauss quadrature rules." Mathematics of computation **23**(106): 221-230.

Gorji, M. B. and D. Mohr (2019). "A Basic Neural Network Model Describing the Plasticity of Sheet Metal." submitted to Numiform.

Gorji, M. B., M. Mozaffar, J. N. Heidenreich, J. Cao and D. Mohr (2020). "On the potential of recurrent neural networks for modeling path dependent plasticity." Journal of the Mechanics and Physics of Solids: 103972.

Graham, R. L. (1969). "Bounds on multiprocessing timing anomalies." SIAM journal on Applied Mathematics **17**(2): 416-429.

Gu, D., W. Meiners, K. Wissenbach and R. Poprawe (2012). "Laser additive manufacturing of metallic components: materials, processes and mechanisms." International materials reviews **57**(3): 133-164.

Guo, C., W. Ge and F. Lin (2015). "Dual-material electron beam selective melting: hardware development and validation studies." Engineering **1**(1): 124-130.

Haarnoja, T., A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta and P. Abbeel (2018). "Soft actor-critic algorithms and applications." arXiv preprint arXiv:1812.05905.

Haghighi, A. and L. Li (2020). "A hybrid physics-based and data-driven approach for characterizing porosity variation and filament bonding in extrusion-based additive manufacturing." Additive Manufacturing **36**: 101399.

Heiden, E., D. Millard, H. Zhang and G. S. Sukhatme (2019). "Interactive differentiable simulation." arXiv preprint arXiv:1905.10706.

Hermann, M., T. Pentek and B. Otto (2016). Design principles for industrie 4.0 scenarios. System Sciences (HICSS), 2016 49th Hawaii International Conference on, IEEE.

Hill, R. (1990). "Constitutive modelling of orthotropic plasticity in sheet metals." Journal of the Mechanics and Physics of Solids **38**(3): 405-417.

Hinton, G. E., S. Osindero and Y.-W. Teh (2006). "A fast learning algorithm for deep belief nets." Neural computation **18**(7): 1527-1554.

Hochreiter, S. (1991). "Untersuchungen zu dynamischen neuronalen Netzen." Diploma, Technische Universität München **91**(1).

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory." Neural computation **9**(8): 1735-1780.

Hochreiter, S. and J. J. N. c. Schmidhuber (1997). "Long short-term memory." **9**(8): 1735-1780.

Holl, P., V. Koltun and N. Thuerey (2020). "Learning to control pdes with differentiable physics." arXiv preprint arXiv:2001.07457.

Hu, Y., L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley and F. Durand (2019). "Difftaichi: Differentiable programming for physical simulation." arXiv preprint arXiv:1910.00935.

Hu, Y., J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus and W. Matusik (2019). Chainqueen: A real-time differentiable physical simulator for soft robotics. 2019 International conference on robotics and automation (ICRA), IEEE.

Huang, Y., M. B. Khamesee and E. Toyserkani (2019). "A new physics-based model for laser directed energy deposition (powder-fed additive manufacturing): From single-track to multi-track and multi-layer." Optics & Laser Technology **109**: 584-599.

Jang, E., S. Gu and B. Poole (2016). "Categorical reparameterization with gumbel-softmax." arXiv preprint arXiv:1611.01144.

Jin, Y., S. Joe Qin and Q. Huang (2016). "Offline predictive control of out-of-plane shape deformation for additive manufacturing." Journal of Manufacturing Science and Engineering **138**(12).

Kamath, C. and Y. J. Fan (2017). "Regression with small data sets: a case study using code surrogates in additive manufacturing." Knowledge and Information Systems: 1-19.

Karpathy, A. and L. Fei-Fei (2015). Deep visual-semantic alignments for generating image descriptions. Proceedings of the IEEE conference on computer vision and pattern recognition.

Keras. Retrieved 3/5/2019, from https://github.com/keras-team/keras/blob/master/keras/layers/wrappers.py#L114.

Khairallah, S. A., A. T. Anderson, A. Rubenchik and W. E. King (2016). "Laser powder-bed fusion additive manufacturing: Physics of complex melt flow and formation mechanisms of pores, spatter, and denudation zones." Acta Materialia **108**: 36-45.

Khanzadeh, M., P. Rao, R. Jafari-Marandi, B. K. Smith, M. A. Tschopp and L. Bian (2018). "Quantifying Geometric Accuracy With Unsupervised Machine Learning: Using Self-

Organizing Map on Fused Filament Fabrication Additive Manufacturing Parts." Journal of Manufacturing Science and Engineering **140**(3): 031011.

King, W., A. Anderson, R. Ferencz, N. Hodge, C. Kamath, S. Khairallah and A. Rubenchik (2015). "Laser powder bed fusion additive manufacturing of metals; physics, computational, and materials challenges." Applied Physics Reviews **2**(4): 041304.

Kingma, D. P. and J. Ba (2014). "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980.

Kingma, D. P. and J. J. a. p. a. Ba (2014). "Adam: A method for stochastic optimization."

Kipf, T. N. and M. Welling (2016). "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907.

Knepley, M. G., K. Rupp and A. R. Terrel (2016). "Finite Element Integration with Quadrature on the GPU." arXiv preprint arXiv:1607.04245.

Knepley, M. G. and A. R. Terrel (2013). "Finite element integration on GPUs." ACM Transactions on Mathematical Software (TOMS) **39**(2): 10.

Koch, S., A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin and D. Panozzo (2019). Abc: A big cad model dataset for geometric deep learning. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

Lee, C.-C. and D.-T. Lee (1985). "A simple on-line bin-packing algorithm." Journal of the ACM (JACM) **32**(3): 562-572.

Lee, J., B. Bagheri and H.-A. Kao (2015). "A cyber-physical systems architecture for industry 4.0-based manufacturing systems." Manufacturing Letters **3**: 18-23.

Lee, J., E. Lapira, B. Bagheri and H.-a. Kao (2013). "Recent advances and trends in predictive manufacturing systems in big data environment." Manufacturing Letters **1**(1): 38-41.

Lee, M.-G., D. Kim, C. Kim, M. L. Wenner, R. H. Wagoner and K. Chung (2007). "A practical two-surface plasticity model and its application to spring-back prediction." International Journal of Plasticity **23**(7): 1189-1212.

Leem, D., N. Moser, H. Ren, M. Mozaffar, K. F. Ehmann and J. Cao (2019). "Improving the accuracy of double-sided incremental forming simulations by considering kinematic hardening and machine compliance." Procedia Manufacturing **29**: 88-95.

Li, G., C. Xiong, A. Thabet and B. Ghanem (2020). "Deepergcn: All you need to train deeper gcns." arXiv preprint arXiv:2006.07739.

Li, H., M. Ramezani, M. Li, C. Ma and J. Wang (2018). "Effect of process parameters on tribological performance of 316L stainless steel parts fabricated by selective laser melting." Manufacturing Letters.

Li, H., M. Ramezani, M. Li, C. Ma and J. Wang (2018). "Effect of process parameters on tribological performance of 316L stainless steel parts fabricated by selective laser melting." Manufacturing letters **16**: 36-39.

Liang, J., M. Lin and V. Koltun (2019). "Differentiable cloth simulation for inverse problems."

Liu, Z. and J. Zhou (2020). "Introduction to Graph Neural Networks." Synthesis Lectures on Artificial Intelligence and Machine Learning **14**(2): 1-127.

Markall, G., A. Slemmer, D. Ham, P. Kelly, C. Cantwell and S. Sherwin (2013). "Finite element assembly strategies on multi-core and many-core architectures." International Journal for Numerical Methods in Fluids **71**(1): 80-97.

Markall, G. R., D. A. Ham and P. H. Kelly (2010). "Towards generating optimised finite element solvers for GPUs from high-level specifications." Procedia Computer Science **1**(1): 1815-1823.

McCulloch, W. S. and W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity." The bulletin of mathematical biophysics **5**(4): 115-133.

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu (2016). Asynchronous methods for deep reinforcement learning. International conference on machine learning.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller (2013). "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland and G. Ostrovski (2015). "Human-level control through deep reinforcement learning." Nature **518**(7540): 529.

Moser, N. H. (2019). Deformation Mechanisms and Process Planning in Double-Sided Incremental Forming, Northwestern University.

Mozaffar, M., R. Bostanabad, W. Chen, K. Ehmann, J. Cao and M. Bessa (2019). "Deep learning predicts path-dependent plasticity." Proceedings of the National Academy of Sciences **116**(52): 26414-26420.

Mozaffar, M., E. Ndip-Agbor, S. Lin, G. J. Wagner, K. Ehmann and J. Cao (2019). "Acceleration strategies for explicit finite element analysis of metal powder-based additive manufacturing processes using graphical processing units." Computational Mechanics **64**(3): 879-894.

Mozaffar, M., A. Paul, R. Al-Bahrani, S. Wolff, A. Choudhary, A. Agrawal, K. Ehmann and J. Cao (2018). "Data-driven prediction of the high-dimensional thermal history in directed energy deposition processes via recurrent neural networks." Manufacturing letters **18**: 35-39.

Nachum, O., M. Norouzi, K. Xu and D. Schuurmans (2017). Bridging the gap between value and policy based reinforcement learning. Advances in Neural Information Processing Systems.

Neugebauer, F., N. Keller, H. Xu, C. Kober and V. Ploshikhin (2014). Simulation of selective laser melting using process specific layer based meshing. Proc. Fraunhofer Direct Digital Manufacturing Conf.(DDMC 2014), Axel Demmer, Aachen, Germany.

Ning, J., W. Wang, X. Ning, D. E. Sievers, H. Garmestani and S. Y. Liang (2020). "Analytical thermal modeling of powder bed metal additive manufacturing considering powder size variation and packing." Materials **13**(8): 1988.

NVIDIA (2008). "NVIDIA CUDA C Programming Guide." 1-261.

NVIDIA (2016). NVIDIA GPU Accelerated Applications Catalog.

O'Donovan, P., C. Gallagher, K. Bruton and D. T. O'Sullivan (2018). "A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications." Manufacturing Letters.

Ohno, N. and J. D. Wang (1993). "Kinematic hardening rules with critical state of dynamic recovery, part I: formulation and basic features for ratchetting behavior." International Journal of Plasticity **9**(3): 375-390.

Olah, C. (2015). "Understanding LSTM Networks." 2018, from http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Olson, G. B. (1997). "Computational design of hierarchically structured materials." Science **277**(5330): 1237-1242.

Parry, L., I. Ashcroft and R. D. Wildman (2016). "Understanding the effect of laser scan strategy on residual stress in selective laser melting through thermo-mechanical simulation." Additive Manufacturing **12**: 1-15.

Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer (2017). "Automatic differentiation in pytorch."

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein and L. Antiga (2019). Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems.

Pichler, F. and G. Haase (2017). "Finite element method completely implemented for graphic processor units using parallel algorithm libraries." The International Journal of High Performance Computing Applications: 1094342017694703.

Pratama, P. S., J.-W. Kim, H.-K. Kim, S.-M. Yoon, T.-K. Yeu, S. Hong, S.-J. Oh and S.-B. Kim (2015). Path planning algorithm to minimize an overlapped path and turning number for an underwater mining robot. 2015 15th International Conference on Control, Automation and Systems (ICCAS), IEEE.

Price, A. D. (2013). "Multi-GPU Computing with Abaqus: Benchmarking and scaling for multiphysics applications in mechatronics."

PwC. (2020). "https://www.pwc.com/gx/en/issues/data-and-analytics/publications/artificial-intelligence-study.html." from https://www.pwc.com/gx/en/issues/data-and-analytics/publications/artificial-intelligence-study.html.

Qiao, Y.-L., J. Liang, V. Koltun and M. C. Lin (2020). "Scalable differentiable physics for learning and control." arXiv preprint arXiv:2007.02168.

Rai, A., M. Markl and C. Körner (2016). "A coupled Cellular Automaton–Lattice Boltzmann model for grain structure simulation during additive manufacturing." Computational Materials Science **124**: 37-48.

Roy, M. and O. Wodo (2020). "Data-driven modeling of thermal history in additive manufacturing." Additive Manufacturing **32**: 101017.

Sabelle, M., M. Walczak and J. Ramos-Grez (2018). "Scanning pattern angle effect on the resulting properties of selective laser sintered monolayers of Cu-Sn-Ni powder." Optics and Lasers in Engineering **100**: 1-8.

Salimans, T., J. Ho, X. Chen, S. Sidor and I. Sutskever (2017). "Evolution strategies as a scalable alternative to reinforcement learning." arXiv preprint arXiv:1703.03864.

Samuel, A. L. (1959). "Some studies in machine learning using the game of checkers." IBM Journal of research and development **3**(3): 210-229.

Schaul, T., J. Quan, I. Antonoglou and D. Silver (2015). "Prioritized experience replay." arXiv preprint arXiv:1511.05952.

Schoinochoritis, B., D. Chantzis and K. Salonitis (2017). "Simulation of metallic powder bed additive manufacturing processes with the finite element method: A critical review." Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture **231**(1): 96-117.

Schrittwieser, J., I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis and T. Graepel (2020). "Mastering atari, go, chess and shogi by planning with a learned model." Nature **588**(7839): 604-609.

Schulman, J., P. Moritz, S. Levine, M. Jordan and P. Abbeel (2015). "High-dimensional continuous control using generalized advantage estimation." arXiv preprint arXiv:1506.02438.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford and O. Klimov (2017). "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347.

Shamsaei, N., A. Yadollahi, L. Bian and S. M. Thompson (2015). "An overview of Direct Laser Deposition for additive manufacturing; Part II: Mechanical behavior, process parameter optimization and control." Additive Manufacturing **8**: 12-35.

Sheng, X., X. Lu, J. Zhang and Y. Lu (2020). "An analytical solution to temperature field distribution in a thick rod subjected to periodic-motion heat sources and application in ball screws." Engineering Optimization: 1-20.

Shoham, Y., R. Perrault, E. Brynjolfsson, J. Clark, J. Manyika, J. C. Niebles, T. Lyons, J. Etchemendy and Z. Bauer (2018). "The AI Index 2018 Annual Report." AI Index Steering Committee, Human-Centered AI Initiative, Stanford University. Available at http://cdn. aiindex. org/2018/AI% 20Index **202018**.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam and M. Lanctot (2016). "Mastering the game of Go with deep neural networks and tree search." nature **529**(7587): 484.

Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller (2014). Deterministic policy gradient algorithms.

Smith, J., W. Xiong, J. Cao and W. K. Liu (2016). "Thermodynamically consistent microstructure prediction of additively manufactured materials." Computational mechanics **57**(3): 359-370.

Song, L., V. Bagavath-Singh, B. Dutta and J. Mazumder (2012). "Control of melt pool temperature and deposition height during direct metal deposition process." The International Journal of Advanced Manufacturing Technology **58**(1): 247-256.

Steuben, J. C., A. P. Iliopoulos and J. G. Michopoulos (2016). "Implicit slicing for functionally tailored additive manufacturing." Computer-Aided Design **77**: 107-119.

Stevens, E. L., J. Toman, A. C. To and M. Chmielus (2017). "Variation of hardness, microstructure, and Laves phase distribution in direct laser deposited alloy 718 cuboids." Materials & design **119**: 188-198.

Sutton, R. S. and A. G. Barto (2018). Reinforcement learning: An introduction, MIT press.

Swainson, W. K. (1977). Method, medium and apparatus for producing three-dimensional figure product, Google Patents.

Tan, X., Y. Kok, Y. J. Tan, M. Descoins, D. Mangelinck, S. B. Tor, K. F. Leong and C. K. Chua (2015). "Graded microstructure and mechanical properties of additive manufactured Ti–6Al–4V via electron beam melting." Acta Materialia **97**: 1-16.

Thingiverse. "https://www.thingiverse.com/about/." Retrieved 11/10/2019, 2019.

Van Hasselt, H., A. Guez and D. Silver (2016). Deep reinforcement learning with double q-learning. Thirtieth AAAI conference on artificial intelligence.

Vinyals, O., A. Toshev, S. Bengio and D. Erhan (2015). Show and tell: A neural image caption generator. Proceedings of the IEEE conference on computer vision and pattern recognition.

Wang, J., Y. Li, R. Zhao and R. X. Gao (2020). "Physics guided neural network for machining tool wear prediction." Journal of Manufacturing Systems **57**: 298-310.

Wang, X., Y. Ye and A. Gupta (2018). Zero-shot recognition via semantic embeddings and knowledge graphs. Proceedings of the IEEE conference on computer vision and pattern recognition.

Wang, Z. and A. M. Beese (2017). "Effect of chemistry on martensitic phase transformation kinetics and resulting properties of additively manufactured stainless steel." Acta Materialia **131**: 410-422.

Wang, Z., T. Chen, J. Ren, W. Yu, H. Cheng and L. Lin (2018). "Deep reasoning with knowledge graph for social relationship understanding." arXiv preprint arXiv:1807.00504.

Wang, Z., T. A. Palmer and A. M. Beese (2016). "Effect of processing parameters on microstructure and tensile properties of austenitic stainless steel 304L made by directed energy deposition additive manufacturing." Acta Materialia **110**: 226-235.

Wang, Z., T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot and N. De Freitas (2015). "Dueling network architectures for deep reinforcement learning." arXiv preprint arXiv:1511.06581.

Weiss-Cohen, M., I. Sirotin and E. Rave (2008). Lawn mowing system for known areas. 2008 International Conference on Computational Intelligence for Modelling Control & Automation, IEEE.

Wenjun, G., G. Chao and L. Feng (2015). "Microstructures of components synthesized via electron beam selective melting using blended pre-alloyed powders of Ti6Al4V and Ti45Al7Nb." Rare Metal Materials and Engineering **44**(11): 2623-2627.

West, D. and C. Lansang (2018). "Global manufacturing scorecard: How the US compares to 18 other nations." Brookings, July **10**.

Wolff, S. J., S. Lin, E. J. Faierson, W. K. Liu, G. J. Wagner and J. Cao (2017). "A framework to link localized cooling and properties of directed energy deposition (DED)-processed Ti-6Al-4V." Acta Materialia **132**: 106-117.

Xu, B., N. Wang, T. Chen and M. J. a. p. a. Li (2015). "Empirical evaluation of rectified activations in convolutional network."

Xu, H. Y., Y. Li, C. Brinson and W. Chen (2014). "A Descriptor-Based Design Methodology for Developing Heterogeneous Microstructural Materials System." Journal of Mechanical Design **136**(5): 051007.

Yan, W., S. Lin, O. L. Kafka, Y. Lian, C. Yu, Z. Liu, J. Yan, S. Wolff, H. Wu and E. Ndip-Agbor (2018). "Data-driven multi-scale multi-physics models to derive process–structure–property relationships for additive manufacturing." Computational Mechanics: 1-21.

Yang, L., O. Harrysson, H. West and D. Cormier (2012). "Compressive properties of Ti–6Al–4V auxetic mesh structures made by electron beam melting." Acta Materialia **60**(8): 3370-3379.

Yin, J., H. Zhu, L. Ke, W. Lei, C. Dai and D. Zuo (2012). "Simulation of temperature distribution in single metallic powder layer for laser micro-sintering." Computational Materials Science **53**(1): 333-339.

Young, T., D. Hazarika, S. Poria and E. Cambria (2018). "Recent trends in deep learning based natural language processing." ieee Computational intelligenCe magazine **13**(3): 55-75.

Zaeh, M. F. and G. Branner (2010). "Investigations on residual stresses and deformations in selective laser melting." Production Engineering **4**(1): 35-45.

Zhang, S. and R. S. Sutton (2017). "A Deeper Look at Experience Replay." arXiv preprint arXiv:1712.01275.

Zhang, Z., Z. Liu and D. Wu (2020). "Prediction of melt pool temperature in directed energy deposition using machine learning." Additive Manufacturing: 101692.

Zhou, K., A. L. Jensen, C. Sørensen, P. Busato and D. Bothtis (2014). "Agricultural operations planning in fields with multiple obstacle areas." Computers and electronics in agriculture **109**: 12-22.

Zhou, P., Z.-m. Wang, Z.-n. Li and Y. Li (2012). Complete coverage path planning of mobile robot based on dynamic programming algorithm. 2nd International Conference on Electronic & Mechanical Engineering and Information Technology, Atlantis Press.

Zhu, J. (2013). The finite element method: its basis and fundamentals, Elsevier.

Zühlke, D. (2013). Industrie 4.0: From Vision to Reality, tech. report, SmartFactory KL.

Zuo, G., P. Zhang and J. Qiao (2010). Path planning algorithm based on sub-region for agricultural robot. 2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010), IEEE.

APPENDIX A

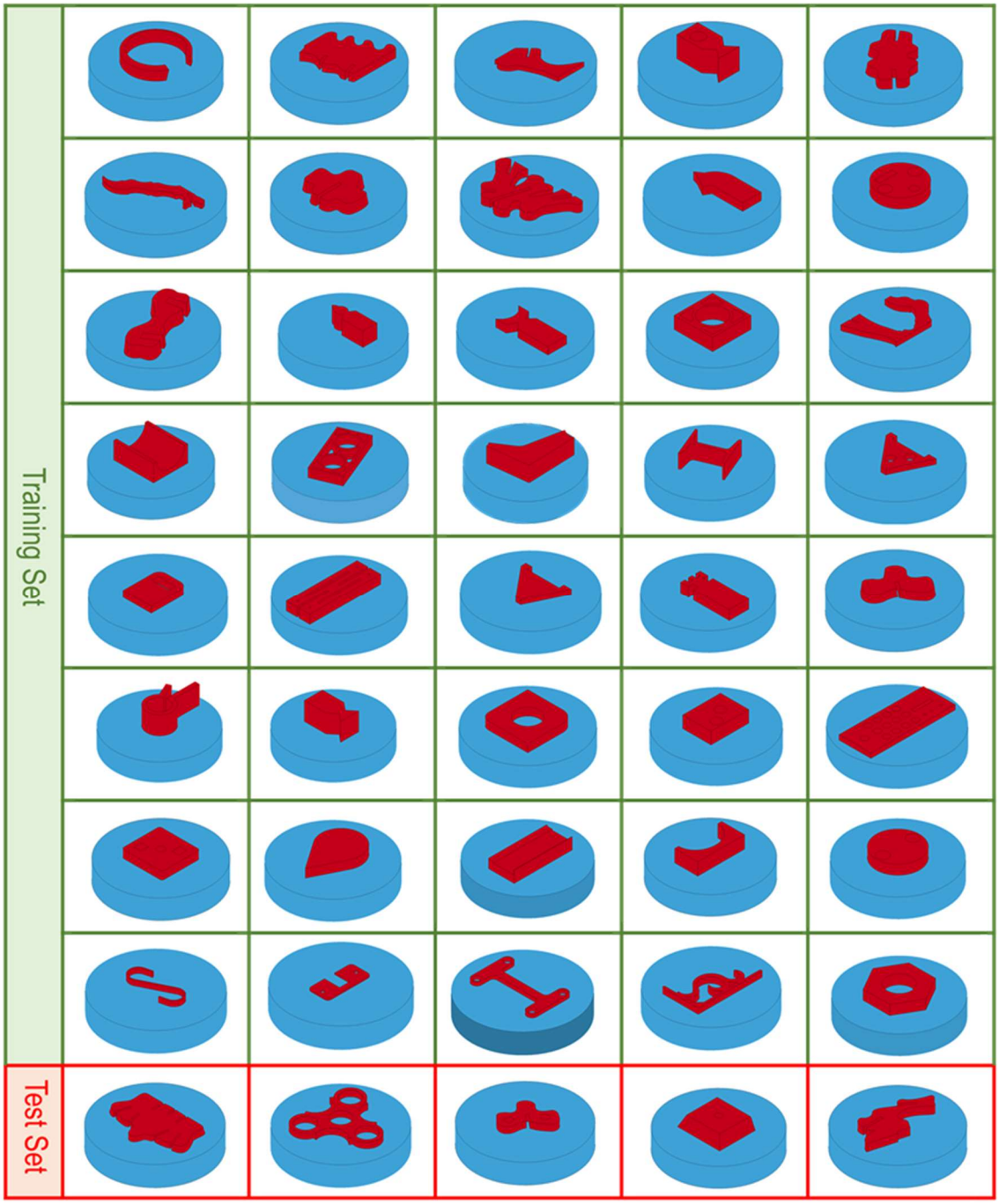


Figure A.1. List of geometries used to produce training and test sets.

APPENDIX B

Yield surface construction for data-driven constitutive modeling

The yield surfaces presented in **Figure 4.8** are constructed by applying 40 linear strain paths, which are uniformly distributed in strain space starting from the initial strain condition. To construct the original yield surface (**Figure A.2A**), strain paths start from the unloaded condition and experience elastic and plastic deformation is different directions. We record the stress state in which each linear path exceeds a plastic energy threshold of $1\ mJ$, which constructs the yield surface. The yield surface of an RVE after it undergoes a certain loading (**Figure A.2B**) is constructed by initially applying the main load (blue solid line in **Figure A.2B**) for all 40 linear strain paths and then loading the RVE in different directions where we detect the stress state when they reach the plastic energy threshold. Note that although all the applied loadings for yield surface constructions are linear and uniform in strain space, the stress responses are neither linear nor uniform which is due to the plasticity of the RVE.

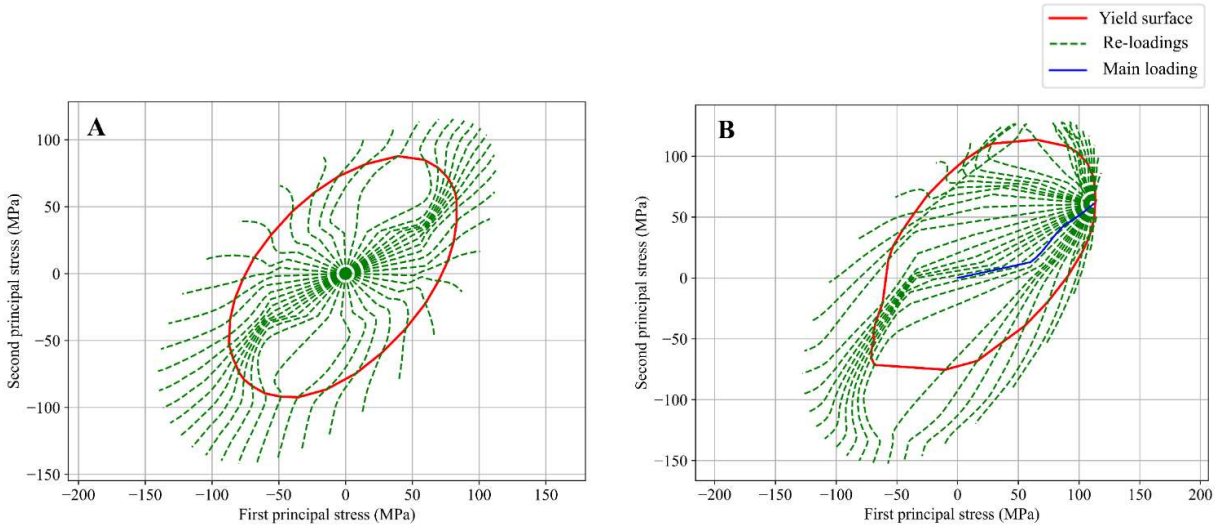


Figure A.2. Yield surface construction process for **(A)** original yield surface and **(B)** yield surface after loading.