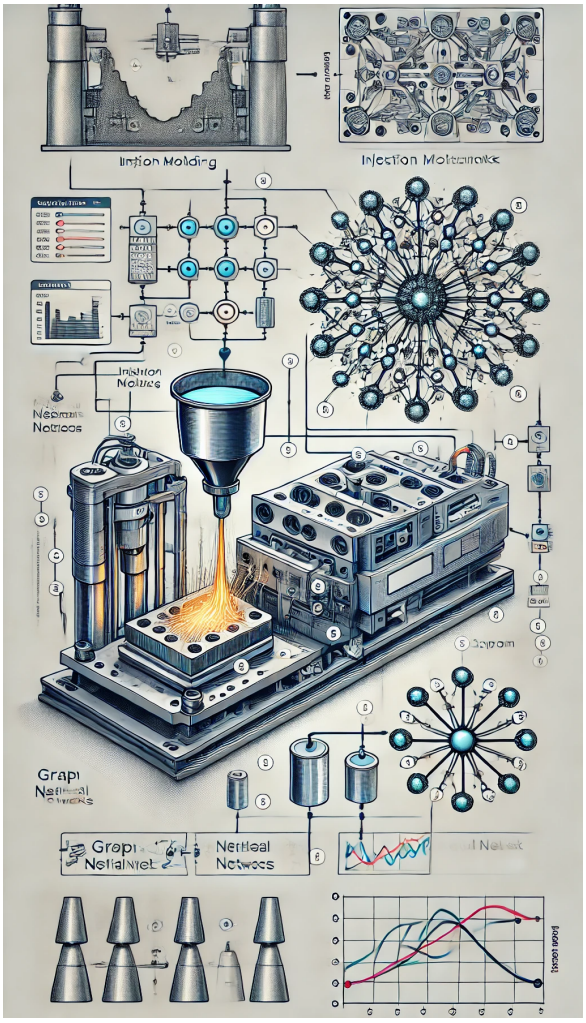
INJECTION MOLDING PROCESS: A SURROGATE BASED ON GRAPH NEURAL NETWORKS

MSc THESIS



INJECTION MOLDING PROCESS: A SURROGATE BASED ON GRAPH NEURAL NETWORKS

MSc THESIS

by

Tharcis ZAARUOLO

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly 26.08.2024 at 14:00

Student Number:	4345258	
Daily supervisor:	Dr. Guillaume Broggi	
Project duration:	May, 2023 - August, 2024	
Thesis committee:	Prof. Dr. M.H.F. Sluiter,	TU Delft, ME faculty
	Prof. Dr. B. Caglar,	TU Delft, AE faculty
	Prof. Dr. M.A. Bessa,	TU Delft, ME faculty
	Dr. A. Rezaei,	TNO
	Prof. Dr. G.A. Filonenko,	TU Delft, ME faculty

This work is performed in collaboration with Brightlands Materials Center (TNO), Geleen, Netherlands

TU Delft Delft University of Technology

Brightlands Materials Center

Keywords: graph neural networks, injection molding, finite volume method

Front Cover: Graph Neural Network surrogate simulating the injection molding process, as envisioned by openAI[1].

An electronic version of this manuscript is available at
http://repository.tudelft.nl/.

[1] **Command to ChatGPT:** Generate an image related to: Injection molding process: a surrogate based on graph neural networks.
Answer ChatGPT: Image format, see front cover.
Unchanged
Date: 17-06-2024

Never confuse education with intelligence,
you can have a MSc degree and still be an idiot.

- Richard Feynman

SUMMARY

Injection molding is a widely used process for manufacturing a large range of polymer parts, including fiber-reinforced polymer composites. The complex nature of this process, combined with the non-Newtonian behavior of molten polymers, implies significant challenges in terms of control, prediction, and optimization of the processing parameters that influence manufacturing quality. While various computational and numerical models have been proposed to simulate the injection molding process and ensure the desired part quality (see, for instance, industrial software such as Moldex3D or Moldflow, they bring their own challenges, including high computational costs, the need for solver tuning, frequent parametric studies, and the economic burden associated with licensing.

In this study, machine learning surrogates are explored as an alternative to conventional modeling software. They are based on Graph Neural Networks (GNNs) that have recently demonstrated remarkable potential for simulating physical systems. In a graph, physical systems are encoded as nodes end edges, capturing their organization. Then, state-of-the-art techniques such as the message passing framework are tested for their potential to learn physics. In the long term, this project aims to establish a novel graph-based framework capable of efficiently simulating complex physics across various material models and processing conditions.

A numerical model is developed for 2D mold filling and quantitively validated against a benchmark. The prediction results demonstrate that the learned simulator is able to capture the physics of melt front behavior. Further, the prediction runtime is significantly faster than the ground truth, completing in under 1 second compared to ~hours, representing a substantial improvement in efficiency. However, the accuracy of the predictions can vary, with errors accumulating especially during testing. While the scalar loss metric provides an estimate of the model's performance, it lacks expressiveness and can be unreliable on its own.

ACKNOWLEDGEMENTS

Given the diverse scope of this project and the involvement of experts from different fields, numerous individuals are to be thanked for their contributions. First and foremost: Dr. Guillaume Broggi. Being a supervisor is one thing, but the way you did it is truly exceptional. Prof. Dr. Baris Caglar, for your guidance and knowledge as the main advisor on this project. Prof. Dr. Miguel Bessa, for being an important pillar for this work, despite geographic constraints. Dr. Ali Rezaei, for providing me with the opportunity to take part in this research topic at Brightlands Materials Center (TNO), Geleen. Your support and insights from the industry's perspective were very valuable. Marcel Sluiter, for giving a critical external perspective. The Bessa research group and the COMMA research group members for adopting me early on and supporting me throughout this project.

I also want to thank others who were not directly involved with the project but made significant contributions: Maciej, for your help generating figures using your expertise with graphic design tools. Melle, for letting me borrow your laptop all this time. Daniel for the chatGPT premium access. 't Kwartje for preventing me from studying too much. Papa and mamma, for being patient with my student career - I am done now! Last but not least, Dilara for being the best girlfriend I could have wished for.

This work marks the end of my long journey as a student. It lasted a decade, and I enjoyed every moment of it. If I could, I would do it all again.

Tharcis Zaaruolo
Delft, August 2024

CONTENTS

LIST OF FIGURES

List of Tables

ACRONYMS

ABS Acrylonitrile Butadiene Styrene

AI Artificial Intelligence

ANN Artificial Neural Network

ANOVA Analysis of Variance

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BC Boundary Condition

CFD Computational Fluid Dynamics

CNN Convolutional Neural Network

COO Coordinate Format

CPU Central Processing Unit

CSV Comma-separated values

CUDA Compute Unified Device Architecture

DEM Discrete Element Method

DNS Direct Numerical Simulation

DoE Design of Experiments

FDM Finite Difference Method

FEM Finite Element Method

FVM Finite Volume Method

GCN Graph Convolutional Network

GNN Graph Neural Network

GPU Graphics Processing Unit

HDF5 Hierarchical Data Format version 5

HDPE High-Density Polyethylene

JSON JavaScript Object Notation

LFT Long Fiber Thermoplastics

MAE Mean Absolute Error
MIMO Multiple Inputs Multiple Outputs
ML Machine Learning
MLP Multi-Layer Perceptron
MPM Material Point Method
MSE Mean Squared Error

NN Neural Network
NS Navier-Stokes

PDE Partial Differential Equation
PE Polyethylene
PEEK Polyether Ether Ketone
PET Polyethylene Terephthalate
PIMPLE Merged PISO-SIMPLE algorithm
PINNs Physics-Informed Neural Networks
PISO Pressure-Implicit with Splitting of Operators
PP Polypropylene
PVT Pressure-Volume-Temperature
PyG PyTorch Geometric

RAM Random Access Memory
RANS Reynolds-Averaged Navier-Stokes
RMSE Root Mean Square Error
RNN Recurrent Neural Network
ROM Reduced Order Model

SFT Short Fiber Thermoplastics

SIMPLE Semi-Implicit Method for Pressure Linked Equations

SMAE Smooth Mean Absolute Error

SPH Smoothed Particle Hydrodynamics

TPC Thermoplastic Composites

VOF Volume of Fluid

VRAM Video Random Access Memory

1

INTRODUCTION

THE injection molding process is a widely used processing technique for polymer-based materials. It ranges from simple, purely polymeric products to two-phase lightweight thermoplastic composites, which allows for a high strength-to-weight ratio family of materials. The basics of polymers and composites can be found in subsection 2.1.2. They are used in everyday household appliances as well as in the automotive and aerospace industries, where the geometry complexity varies correspondingly.

(a) The idler wheel, a part of a performance compound archery bow.

(b) Suspension element of a performance mountain bike.

Figure 1.1: Application examples of thermoplastic composites. Successful weight reduction compared to their original metallic counterparts. Reproduced from [1].

For instance, an idler wheel is a crucial part of a performance archery bow made initially from machined aluminum. Replacing this with a short fiber-reinforced polyamide composite, shown in Figure 1.1a, reduced weight by 50% [1]. A similar result was achieved by replacing metal injection molded magnesium in a performance mountain bike's suspen-

sion with a short carbon fiber-reinforced polyphenylene sulfide composite, as depicted in Figure 1.1b.

The setup of the injection molding process is shown in Figure 1.2. Plastic granules are fed into the hopper and directed on the screw under high temperature and pressure. The melt is injected into the mold, where the part is ejected after cooling. Details are provided in subsection 2.1.1. The processing parameters significantly impact the product's final quality [41]. Due to the multitude of parameters and their mutual influence, coupled with the non-Newtonian behavior of polymer melts, optimizing the whole process is a complex task. Various methods have been developed to tackle this issue, originating from experimental studies to computational methods (occasionally coupled with Machine Learning) to predict and optimize the process.

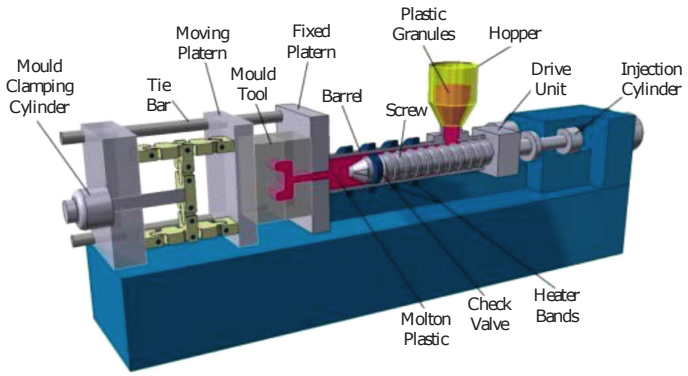


Figure 1.2: Plastic injection molding setup. Reproduced from [2].

As current methods tend to be computationally expensive, efforts have been made to explore alternatives with the help of Machine Learning [2]. Graph Neural Networks (GNNs) have recently gained much attention due to significant developments in simulating complex physics over the past couple of years [31–33]. Trained simulators have been shown to accurately predict complex physical phenomena at a fraction of the computation time required by other computational models. While generalization and extrapolation have always been challenging for machine learning-based approaches, Sanchez-Gonzalez *et al.* [31], Rubanova *et al.* [32], Pfaff *et al.* [33] have shown evidence suggesting that GNNs can learn physics. However, graph-based approaches for physics simulations are still in their early development, so their potential and limitations must be critically assessed. This work aims to address these challenges within the injection molding framework.

This manuscript is structured as follows. In section 2.1 the process and the materials are explained, followed by the importance of using the correct processing parameters to retain the desired final quality of the part. Then, section 2.2 provides an overview of the methods currently used in the industry to analyze and quantify the influence of the inputs on the outcomes, ranging from experimental studies to simulations and the involvement of Machine Learning. In section 2.3 a step is taken back which shows how Machine Learning is applied to fluid simulations. Graph Neural Networks are intro-

duced and proposed as an alternative in section 2.4. The knowledge gap is defined in section 2.5. The generation of the ground truth data, needed to train the graph neural network, is dissected in chapter 3. It covers pre-processing, tuning & running the solver, and post-processing. The construction of the graph and the training of the simulator are explained in chapter 4. The results are shown in chapter 5. This work is rounded off with a discussion in chapter 6, followed by the conclusions in chapter 7.

2

LITERATURE REVIEW

THIS literature review sets the foundation for identifying future research directions. Given the range of subjects covered, it is crucial to scrutinize each one to gain a good understanding. Research gaps need to be identified to motivate further investigations.

2.1. THE INJECTION MOLDING PROCESS

Injection molding is one of the most prominent polymer processing techniques, accounting for one-third of the production of polymeric products worldwide in 2021 [42, 43] and has been growing ever since [44]. It is widely used for producing plastic composite parts with good dimensional accuracy, complex geometry, and remarkable mechanical and physical qualities [10]. It is a cyclic process (operating in a continuous loop) that allows for close tolerances, little post-production work, reuse of plastic waste, and full automation at low cost [3, 45]. This motivates its application in the plastic, medical, electronics, automotive, and aerospace industries [10, 46, 47].

During the injection molding process, heating the thermoplastic allows for shaping the material, followed by a cooling and extraction process. The use of thermoplastics can contribute to environment-friendly technologies, offer short cycle times of < 1 minute, and ensure large numbers of production with consistent quality [10].

2.1.1. PROCESS

The injection molding process consists of different stages, as depicted in the schematic reported in Figure 3.2. The first part of the process (transport, melt generation, mixing, pressurization, and flow) is carried out in the injection unit of the machine, and the second one (product shaping) in the mold cavity [48]. Pristine or fiber-filled polymeric material (often in pellet or granular form) is fed into the single-screw extrusion machine, which is heated up under high pressure, usually reaching 100 MPa [49] and injected into a mold. Upon cooling, the mold opens, and the part is extracted [3, 41, 48].

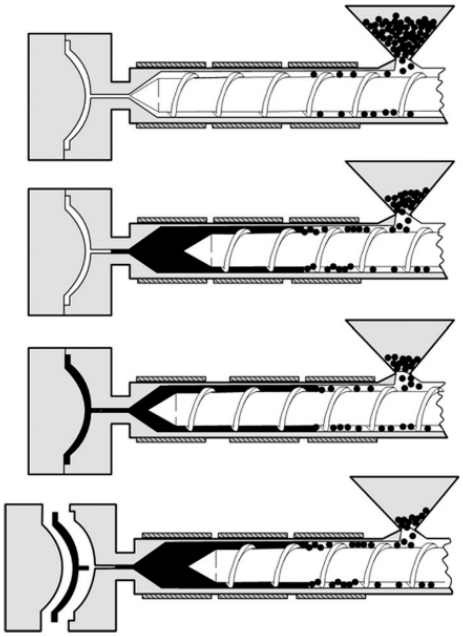


Figure 2.1: Schematic layout of the injection molding process. Reproduced from [3].

This process is commonly divided into the following phases [50, 51]:

1. **Mold closing**
   The first stage consists of feeding the granular material into the hopper, which directs it onto the screw. The mold is closed at this point.

2. **Injection**
   The reciprocating screw can move back and forth along its axis and rotates, causing the mixture to shear and dissipate heat. Together with the added heat through the barrel wall, the polymer melts, turns into a viscous liquid, and is directed towards the injection chamber in front of the screw. This is achieved by simultaneously turning and retracting the screw. It accumulates under an adjustable pressure until the desired volume is attained. At this point, the screw pushes forward as a piston and forces the polymer melt into the closed mold.

3. **Packing**
   The cavity is filled, and the pressure is maintained to counteract shrinkage upon cooling.

4. **Cooling**
   Next, the gate freezes, the cavity is isolated from the barrel, and the product can cool down by water running through the cooling channels.

5. **Plastication and screw back**
   The screw moves back along its axis, creating space for the next charge of polymer melt at its tip.

6. **Ejection**
   As the part gains stiffness and turns rigid due to the temperature drop, the mold is opened, and the part is extracted.

Typical temperature and pressure values for the material and machine components are listed in Table 2.1. From the mold's closing up until the final product's ejection, this process involves high flow and shear rates, several heat transfer mechanisms, time-varying boundary conditions, and complex geometry and material properties. The final quality is compromised if not appropriately controlled, as highlighted in subsection 2.1.3.

Table 2.1: Typical values for standard molding conditions. Adapted from [36]. These are based on the standard molding conditions recommended by Sumitomo DEMAG [36] for SUMIKAEXCEL PES and SUMIPLOY polymers and provide an estimate of typical values.

$T_{cylinder}$	T_{resin}	T_{mold}	$p_{injection}$	$p_{holding}$
300-420 °C	340-420 °C	120-180 °C	100-200 MPa	50-200 MPa

2.1.2. MATERIALS

Polymers are commonly classified into three groups: thermosets, thermoplastics, and elastomers [52]. The origin of their contrasting behavior can be traced back to the scheme of polymer chains at a macroscopic level, as depicted in Figure 2.2. Covalent bonds link the individual monomer units, forming chains that act independently. In the case of thermosets, additional cross-links form between the chains during the curing process. These cross-links are responsible for the often increased mechanical properties, higher melting points, and resistance to high temperatures compared to thermoplastics.

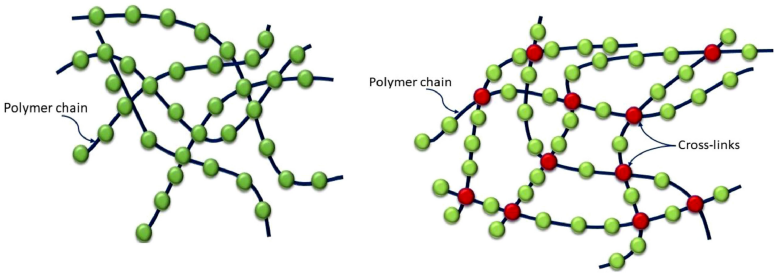


Figure 2.2: The schemes of thermoplastic and thermoset polymeric materials: (left) thermoplastics, with weak intermolecular forces between polymer chains; (right) thermosets, with strong covalent bonds. Reproduced from [4].

Consequently, thermoplastics can undergo multiple (however limited) heating and cooling cycles with softening or hardening, while thermosets char or burn when exposed to

high temperatures [53]. This property allows for recycling thermoplastics, for instance, by scrapping and reusing them in melt processing methods such as injection molding [6]. On the contrary, recycling is significantly challenging for thermosets.

Figure 2.3: A visual representation of the three types of molecular arrangements within a polymer. Reproduced from [5].

The injection molding process is suitable for an extensive range of materials. This work focuses on the use of thermoplastics. The thermoplastic molecular arrangement can achieve one of three possible structures: amorphous, semicrystalline, or crystalline configuration, as shown in Figure 2.3. These differences influence the physical and mechanical properties of the material. Amorphous polymers lack an ordered structure. Chains are randomly arranged in a random network. Crystalline polymers are quite the opposite: the molecular structure is ordered with regular repeating patterns (crystal lattice structures). Often, however, polymers do not exhibit complete crystallinity, as a significant proportion tends to be amorphous. The coexistence of these two phases is called a semicrystalline polymer.

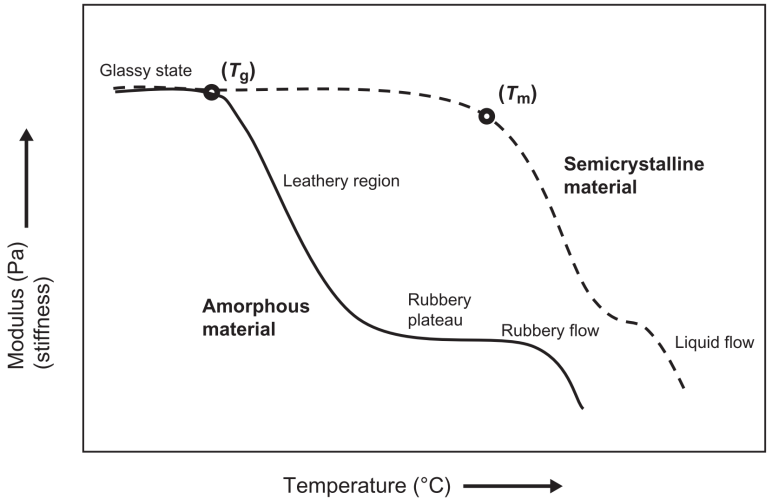


Figure 2.4: Melting and softening behavior of semicrystalline and amorphous materials. Reproduced from [6].

The transition between solid and liquid state is determined by the glass transition temperature T_g and the melting temperature T_m of the respective polymer. Amorphous polymers transition to a rubbery viscous fluid phase above T_g, gradually softening, while crystalline polymers transition to a fluid above a sharp melting point T_m. For semicrystalline polymers, T_m is generally speaking higher than T_g. This sudden increase in mobility, coupled with changes in temperature, pressure, and shear rate, affects the viscosity and strain response of the non-Newtonian polymer melt. Figure 2.4 illustrates that above T_g, the stiffness of amorphous materials rapidly decreases while semicrystalline materials retain their mechanical properties until T_m.

Table 2.2: Typical physical and mechanical properties of thermoplastic polymers. The values displayed are midpoint approximations. Adapted from [37–39].

Material	E $[GPa]$	ρ $[g/cm^3]$	T_g $[°C]$	T_m $[°C]$	Structure
ABS	2.20	1.10	120	*N/A*	Amorphous
PET	2.80	1.30	75	255	Semi-crystalline
PE	0.20	0.92	-125	115	Semi-crystalline
PEEK	5.10	1.35	145	335	Semi-crystalline
PP	1.65	0.90	-10	170	Semi-crystalline

Table 2.2 shows some examples of thermoplastics and their characteristics typically used for processing. ABS does not have a melting point due to its purely amorphous nature. The tensile strength of a polymer tends to scale with its Young's modulus and is therefore not included in the table [38].

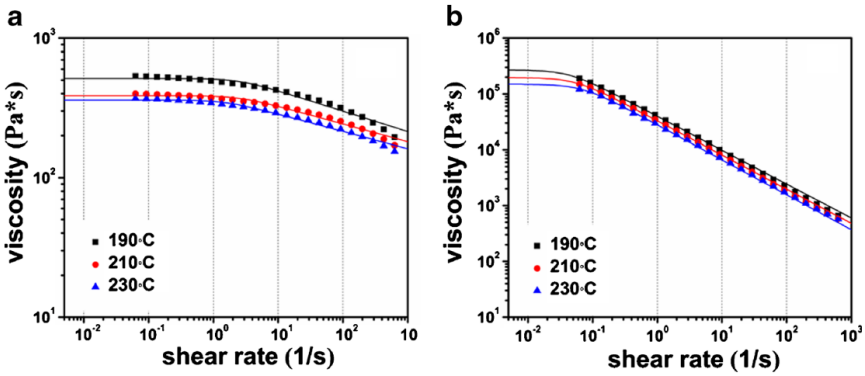


Figure 2.5: Plot of steady viscosity versus shear rate for samples at different temperatures, with the solid lines indicating the non-line fitting curves by Carreau-A model: (a) HDPE; (b) UHMWPE/ HDPE blends. Reproduced from [7].

The thermoplastic fed into the machine is either in its raw form or in combination with a second phase (e.g., particles, fibers). Polyethylene and polypropylene are two of the

most commonly used polymers due to their valuable properties such as high toughness, chemically inertness, and ease of manufacturing [54]. These are often combined with short glass or carbon fibers, increasing the mechanical properties of the composite, such as stiffness and strength [55].

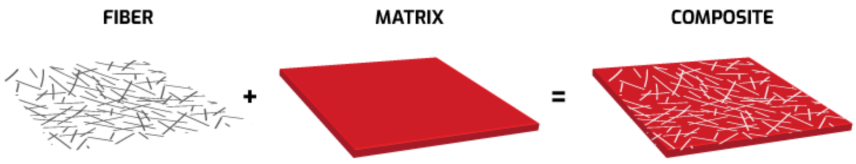


Figure 2.6: Composition of a composite. Reproduced from [8].

The complex rheological properties of polymers have been studied extensively in the past. Shear rate, temperature, and pressure affect the viscosity, and with that, the fluid behavior of the polymer melts [56]. An example is illustrated in Figure 2.5, which shows the relation between viscosity, shear rate, and temperature for high-density polyethylene (HDPE) and ultra-high molecular weight polyethylene (UHMWPE)-HDPE.

Table 2.3: The effect of different reinforcements on the properties of thermoplastic composites (↑ improve, ↓ deteriorate). Reproduced from [10].

Additive	Maximal content (wt%)	Modulus of elasticity	Elongation at break	Toughness	Warpage proneness	Fire resistance
Glass fiber	60	↑↑↑	↓↓	↑	↑	↑↑
Aramid fibers	20	↑↑	↓↓↓	↑↑	↑↑	↓
Carbon fibers	60	↑↑↑	↓↓↓	↑	↑	↓
Basalt fibers	60	↑↑↑	↓	↑↑	↑↑↑	↑↑
Natural fibers	60	↑	↓	↑	↓↓	↓↓
Antistatic agents	5	—	↑↑	—	↓↓↓	—
Elastomers	15	↓	↑↑↑	↑↑	↓↓	↓
Mineral fillers	40	↑↑	↓↓	↑	↑↑↑	↑
Wood fiber	80	↑	↓↓↓	↑	↓↓	↓↓

Furthermore, one must consult the PVT (Pressure-Volume-Temperature) behavior beforehand to avoid part relaxation problems. If not controlled properly, it may lead to dimensional inaccuracies and warping. This intricate nature of polymer blends introduces additional complexities next to the varying stresses, high temperatures, and high cooling rates exposed to the material during the injection molding process. Each of these factors influences the behavior of the melt, creating a challenging environment for tuning the processing conditions.

Fibers can be added to a polymer matrix to form a composite, as shown in Figure 2.6. Fibers in thermoplastics can be short (< 0.5mm), long (< 15mm), or continuous (∞) [57–59]. Some common examples of reinforcement materials are glass, carbon, basalt, ceramic, and natural fibers [10]. By adding this second phase to the polymer matrix, the

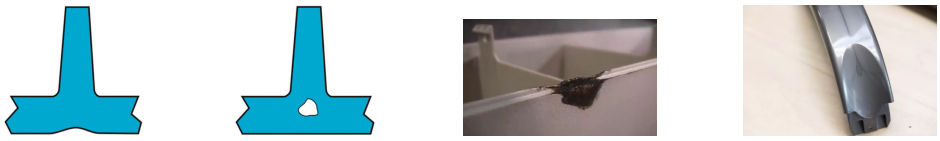
mechanical properties of the material (e.g., stiffness, creep resistance, toughness) are enhanced [10, 51]. Table 2.3 shows how specific properties of thermoplastic composites are affected based on the type of filler. The general trend of the mechanical properties reveals that the modulus of elasticity is inversely correlated to the elongation at break and toughness [60].

2.1.3. PROCESSING PARAMETERS & PRODUCT QUALITY

The complexity of the injection molding process gives rise to several factors that can affect the product's final quality. Preventing molding flaws is crucial, motivating its investigation to understand the whole process better. The four main factors influencing the final part quality are the raw materials, the injection machine, the mold structure, and the injection molding process parameters [61]. In this research, the focus lies on the latter, which covers mold temperature, melt temperature, injection speed, injection pressure, injection time, holding pressure, holding time, cooling time, and mold design [41, 43, 62]. Polymer matrix defects and fiber attrition are explained in the following two sections.

POLYMER MATRIX DEFECTS

Common defects that affect the part quality and mechanical performance together with their causes are listed in Table 2.4 [41]. Aesthetic requirements can also be imposed for the surface finish. Figure 2.7 illustrates four examples easily recognizable by visual inspection.



(a) Sink (b) Void (c) Burn mark (d) Flow line

Figure 2.7: Common plastic injection molding defects. Adapted from [9].

Among these processing parameters, the injection pressure and temperature control are the ones that are the more likely to cause defects when not adequately regulated [62]. All in all, it is a multi-factor optimization problem where preventing some defects may promote others. Table 2.4 shows that, for example, low injection pressure is a cause of weld lines, whereas high injection pressure is a cause of burn marks. Thus, a trade-off must be identified, which depends on the desired properties, part geometry, and material. The computational pipeline is critical to recognize this trade-off without lengthy and costly experimental campaigns.

FIBER ORIENTATION & FIBER BREAKAGE

Fiber orientation Fiber orientation significantly impacts the properties of a TPC since the longitudinal and transverse properties of a fiber differ. Table 2.5 shows how the tensile strength and modulus change depending on the alignment between the test direction and fiber orientation. During the filling process, the orientation of fibers varies. The

Table 2.4: Defects in injection molding. Adapted from [2].

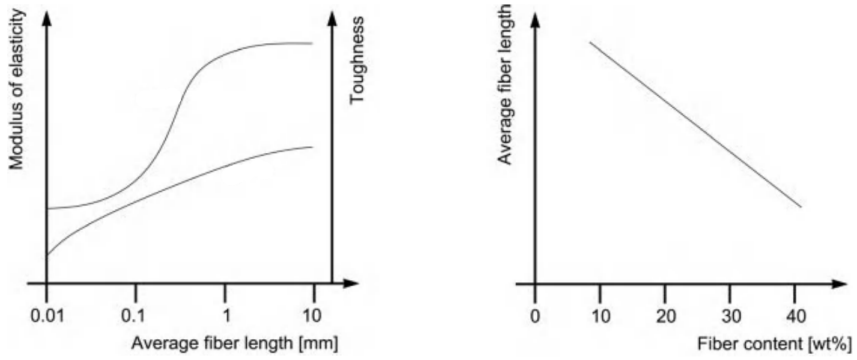
Defect	Causes
Warping	Inadequate cooling time
Flow lines	A varying flow rate of molten material Low injection pressure
Weld lines	Low injection pressure or speed Low mold temperature Gates and runners are not of adequate size
Flash	Low clamping force High injection pressure
Surface delamination	Presence of contaminants or foreign particles
Burn marks	High injection pressure High screw speed Cooling time is inadequate
Sink marks	Small runners or gates High mold temperature

melt flow interacts with its surroundings (e.g., boundaries), which affects the flow and aligns the fibers accordingly. Different regions within the mold exhibit variations in fiber alignment, with some areas showing random orientations while others have fibers predominantly parallel or perpendicularly to the flow. The average roughly determines the average fiber orientation. This property is influenced by processing parameters such as the injection rate, melt -, and mold temperature.

Table 2.5: Effect of fiber content and orientation on the mechanical properties of polyester/jute composites. Longitudinal - along the fiber; Transverse - across the fiber. Adapted from [40].

Fiber content (% w/w)	Test direction	Tensile strength (MPa)	Tensile modulus (GPa)
10	L	55.6 ± 4.7	3.5 ± 0.2
	T	5.3 ± 1.4	2.0 ± 0.2
30	L	139.7 ± 12.9	4.8 ± 0.1
	T	3.9 ± 0.7	1.7 ± 0.1
50	L	161.8 ± 3.3	5.6 ± 0.5
	T	0.4 ± 0.1	1.0 ± 0.2

Fiber breakage Injection molding is widely applied to thermoplastic matrix composites, consisting of two phases: the polymer matrix and fibers. The most common reinforcement is glass fiber due to its low cost and mechanical properties [10]. During the process, fiber attrition occurs due to the shear stresses induced by the polymer flow, fiber-fiber interaction, and contact with the mold wall. As shown in Figure 2.8, Long-fiber thermoplastic composites (LFTs) grant improved mechanical properties compared to their short-fiber counterparts [63], to which fiber breakage poses a significant problem. The fiber length is decreased along the way (often of the magnitude of a few tenths of millimeters), resulting in a significantly lower fiber length distribution in the final product [11, 51, 64, 65]. As a consequence, the mechanical properties and overall quality of the final product are adversely affected by this phenomenon.



(a) (b)

Figure 2.8: Based on injection-molded thermoplastic composites. (a) The effect of average fiber length on the elastic modulus and toughness. (b) The effect of fiber content on average fiber length. Reproduced from [10].

2.2. RESEARCH & OPTIMISATION

This section deals with the methods commonly used to observe, analyze, quantify, and optimize processing parameters and conditions to achieve the desired mechanical properties and product quality. Two important goals are set for this part of the research:

1. Analyse the influence of each (and combinations of) processing parameters on the outcome.
2. Optimization of the processing parameters.

The current methods can be classified into experimental, computational, and statistical techniques [55]. This optimization process can be tedious and challenging since each parameter affects the outcome, and their effects may be interrelated. subsection 2.2.1 explains experimental set-ups, which are mainly based on trial and error by physically conducting the experiments and analyzing the dependence of certain final part qualities on the input parameters. subsection 2.2.3 highlights the use of numerical simulations in

this context and its current limitations. Artificial Intelligence (Machine Learning) applications for this purpose have drawn significant attention over the years, summarised in subsection 2.2.4.

2.2.1. EXPERIMENTAL STUDIES

Determining the impact of processing parameters on the quality of molded parts through experiments used to be largely based on trial and error [55]. Molding personnel often rely on their expertise and "know-how" based on their experience in the field [61].

Kurt *et al.* [66] investigated the effect of molding parameters (packing pressure, melt temperature, and cooling time) on shrinkage (reduction in volume as it solidifies) and roundness of parts. They noted that the injection pressure and melt temperature are the dominant factors that determine the part quality. Additionally, they established a relationship between the processing parameters and mold temperature. They reported that cooling significantly impacted the roundness error, whereas no effect was observed on the cavity pressure.

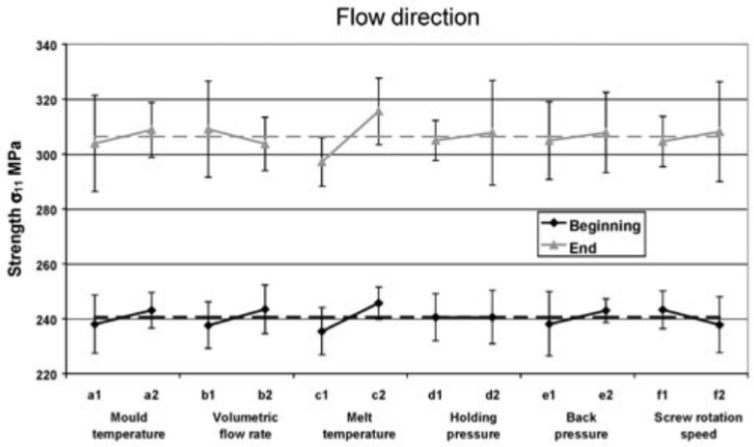


Figure 2.9: Influence of processing conditions for flexural strength in the flow direction, both at the beginning and end of the parts. Adapted from [11].

Lafranche *et al.* [11] identified the main parameters influencing the mechanical properties of 40 wt% 10 mm long glass fiber reinforced polyamide 66 injection molded parts. The change in the strength of specific geometries was recorded depending on temperature, pressure, flow rate, and screw rotation speed modifications. The outcome is shown in Figure 2.9.

While all parameters influence the strength to some extent, the melt temperature dominates the change in strength for the beginning of the parts. The same trend holds regarding the end of parts. Mold temperature, flow rate, back pressure, and screw rotation speed contribute to this change in strength, but to a lesser extent. However, it is difficult

to draw firm conclusions since the error bars are big and overlap. The trends, even if small, remain.

Simple experimental approaches lead to a waste of material and are time-consuming. The whole process also often turns out to be rather expensive. Additionally, the inability to tackle Multiple Inputs Multiple Outputs (MIMO) (in other words, the influence and interrelated effect of multiple parameters on the outcome) is a significant drawback. The tuning of this process is a problem involving higher dimensions. Experimental methods remain crucial with their ability to provide accurate results, which motivates re-occurring experimental studies that aim to quantify the impact of input settings on output quality. They also serve as a valuable ground truth to validate analytical and numerical simulations, described more in detail in subsection 2.2.3.

2.2.2. DESIGN OF EXPERIMENTS: THE TAGUCHI METHOD

The Taguchi method is an experimental method that provides a more structured and optimized approach to explore the parameter space. It is a statistical approach that aims to reduce the time, cost, and number of experiments needed to analyze and quantify the effect of parameters on the quality of the final product. This technique can better tackle this process's MIMO nature by considering the parameters' interrelated effects. It consists of two steps and determines:

1. The best combination of parameters for testing, given a certain set and range
2. The relative contribution of each parameter and the most significant ones affecting a certain defect

Factors	Level		Units
	Low (-)	High (+)	
A. Filling time (A)	2	3	sec.
B. Melt temperature (B)	215	230	°C
C. Mold temperature	35	60	°C
D. Maximum injection pressure	50	80	%
E. Packing time	4	6	sec.
F. Maximum packing pressure profile value	30	50	%
G. Cooling time	12	14	sec.
Uncontrollable Factors			
H. Air temperature	15	40	°C
J. Eject temperature	50	99	°C

(a)

Analysis of Variance for Means

Source	DF	Seq SS	Adj SS	Adj MS	F	P
B	1	6.6967	6.6967	6.6967	29.32	0.000
E	1	22.5192	22.5192	22.5192	98.60	0.000
G	1	24.9595	24.9595	24.9595	109.28	0.000
B*E	1	0.7042	0.7042	0.7042	3.08	0.113
B*G	1	0.8523	0.8523	0.8523	3.73	0.085
E*G	1	2.8344	2.8344	2.8344	12.41	0.006
Residual Err	9	2.0556	2.0556	0.2284		
Total	15	60.6219				

(b)

Figure 2.10: Parameters and their significance. (a) List of factors and levels. (b) ANOVA result of volume shrinkage. Adapted from [12].

A study by Kiatcharoenpol et al. [12] illustrates this approach. The authors used the Taguchi method to determine the effect of seven different processing parameters on the quality (volume shrinkage & total displacement) of parts. According to the first step, the parameters and their respective range are determined and shown in Figure 2.10a.

Next, Orthogonal Arrays (L16) are used to study the interaction of the parameters through 64 experiments. The Analysis of Variance (ANOVA) technique is then used to determine the most significant factors. The results are displayed in Figure 2.10b, where generally speaking, a higher number for the sequential or adjusted Sum of Squares (SS) indicates

a more significant contribution to the defect at hand. The melt temperature (B), packing time (E), and cooling time (G) are the significant factors driving the volume shrinkage. The significant factors differ from before (see subsection 2.2.1 because the interrelated effect was accounted for here. Ultimately, this processed information yields an optimized combination of factor values that minimizes the volume shrinkage.

The drawbacks of this method are, on the one hand, that time and expertise are still required for the large number of experiments involved, prone to wastage and human error. On the other hand, the final values often do not match the predetermined levels set for machines and equipment, causing final settings to deviate from those values.

2.2.3. NUMERICAL APPROACHES

The persisting downsides of experimental studies have motivated the need for techniques that can design and predict experiments based on a minimum number of trials, ensure sufficient accuracy, and the ability to efficiently relate input parameters to the output quality of a part. Computational methods provide a powerful alternative. Key elements are [50]:

1. Mathematical equations (describing the physical processes)
2. Numerical methods (implementing the equations)
3. Properties of the materials
4. Geometric complexity of the mold

The medium under investigation is viewed as a continuum, meaning that the property values are averaged over many molecules, avoiding discontinuities. Typically, a set of balance equations describing the fluid flow and heat transfer needs to be solved. They are classified as follows [48].

The conservation of mass, often referred to as the continuity equation, is described by

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2.1}$$

where ρ is the density of the molten polymer, v is the velocity vector and t is time. For incompressible flows, density is constant, simplifying the equation to $\nabla \cdot (\rho \mathbf{v}) = 0$.
The Navier-Stokes (NS) Equations are a well-established model for describing the momentum balance and fluid flow behavior. They account for the forces acting on the fluid, pressure gradients, viscous stresses, and external forces:

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \nabla \cdot \tau + \mathbf{f} \tag{2.2}$$

where p is the pressure, τ is the stress tensor (which includes viscous stresses), and f represents body forces (e.g., gravity).
The heat transfer equation governs the polymer's temperature distribution and heat transfer. It is described by

$$\rho c_p \left(\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T \right) = k \nabla^2 T + \Phi \tag{2.3}$$

where c_p is the specific heat capacity, T is the temperature, k is the thermal conductivity and Φ represents the viscous dissipation term. In the case of isothermal conditions, this equation can be neglected.
The viscosity characterizes the material properties and its behavior under different conditions. Constitutive equations describe this often shear-dependent behavior. The Carreau model (Equation 2.4) and the Cross model (Equation 2.5) are often used when dealing with polymer melts [67]

$$\eta(\dot{\gamma}) = \eta_0 \left[1 + (\lambda\dot{\gamma})^2\right]^{\frac{n-1}{2}} \tag{2.4}$$

$$\eta(\dot{\gamma}) = \frac{\eta_0}{1 + (K\dot{\gamma})^m} \tag{2.5}$$

where η is the viscosity, η_0 is the zero-shear viscosity, λ is the time constant related to the relaxation of the polymer, $\dot{\gamma}$ is the shear rate, and n and m are parameters.
Boundary conditions are crucial for determining the behavior of the domain's boundaries and the system's initial state. These include specifying velocities and pressures at the inlet and outlet, slip or no-slip conditions at the walls, and temperature or heat fluxes.

ANALYTICAL METHODS

For simple cases, Equation 2.1 and Equation 2.2 can be simplified with the help of assumptions. The Couette and the Poiseuille flow between a fixed and a moving plate are used as an example [13].

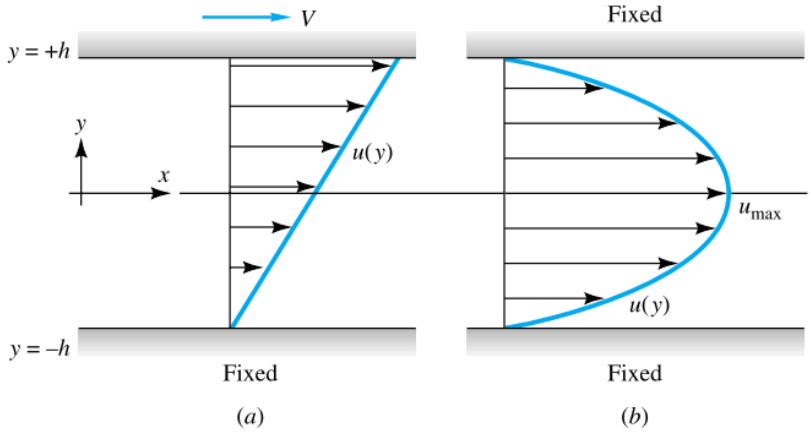


Figure 2.11: Incompressible viscous flow between parallel plates: (a) no pressure gradient, upper plate moving; (b) pressure gradient $\partial p/\partial x$ with both plates fixed. Reproduced from [13].

Figure 2.11 shows the two scenarios, where the aim is to find an expression for the velocity profile between the plates. Within the context of this work, the plates could represent the boundaries of a polymer melt mold-filling process in 2D. The flow is assumed to be fully developed, laminar, and incompressible. No pressure gradient is present, gravity

effects are neglected, and no-slip boundary conditions are applied at the plates. The continuity and NS equations yield

$$u = \frac{V}{2h}y + \frac{V}{2} \quad -h \leq y \leq +h \tag{2.6}$$

for the Couette flow, and

$$u = -\frac{dp}{dx}\frac{h^2}{2\mu}\left(1 - \frac{y^2}{h^2}\right) \tag{2.7}$$

for the Poiseuille flow. In practice, however, more complex cases are encountered where the flow dynamics are not straightforward and assumptions are limited. Consequently, numerical methods become essential for solving these cases.

COMPUTATIONAL METHODS

The core solving techniques of the coupled equations presented in the previous section are based on finite differences, finite elements, or finite volumes. The standard numerical approaches that employ these can be classified into the Eulerian and the Lagrangian framework. A (hybrid) combination of the two is also possible. An Eulerian description of a flow field assumes a fixed mesh and material is allowed to move through the domain. The material deforms on its own through space and time. The flow velocity [68] is represented by the function

$$u(x,t), \tag{2.8}$$

where x is the position and t is time. The concept is illustrated in Figure 2.12a.
The Lagrangian framework, on the other hand, discretizes the material (or fluid in this case) into a set of particles that move and interact with each other. The position of each particle is tracked over time, providing history-dependent information. This concept is shown in Figure 2.13a. The flow is denoted by the function:

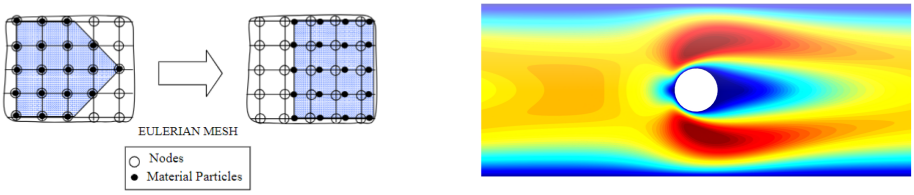
$$X(x_0,t), \tag{2.9}$$

where x_0 is the position of the respective particle at time t. The two frameworks are related as follows:

$$u(X(x_0,t),t) = \frac{\partial X}{\partial t}(x_0,t). \tag{2.10}$$

Both sides represent the velocity of a particle x_0 at time t.
The following paragraphs illustrate four standard techniques that rely on these frameworks: the Finite Element Method (FEM), Smoothed Particle Hydrodynamics (SPH), Discrete Element Method, and Material Point Method (MPM).

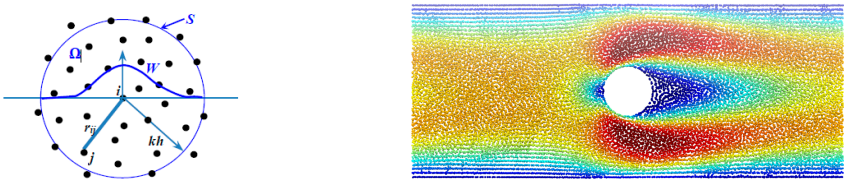
Finite Element Method (FEM) The Finite Element Method is based on the Eulerian framework. Figure 2.12 shows this principle and its application for simulating a velocity profile around a cylinder, as also often encountered in the CFD realm [15].



(a) (b)

Figure 2.12: (a) Description of motion for Eulerian formulation. Reproduced from [14]. (b) Steady-state velocity profile around a cylinder predicted with FEM. Reproduced from [15].

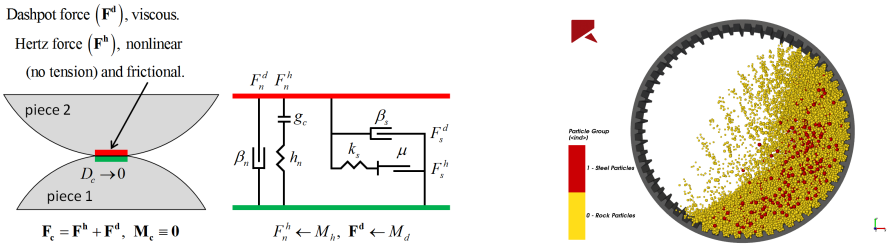
Smoothed Particle Hydrodynamics (SPH) Smoothed Particle Hydrodynamics is a meshless lagrangian-based technique. The material particles move and interact with each other through the interpolation of Kernel functions [14, 15, 69–72]. Due to the absence of a mesh, this technique allows for large deformations of possibly irregular geometries. The fact that each particle needs to be tracked, updated, and recorded makes it a more computationally demanding method.



(a) (b)

Figure 2.13: The SPH framework and profile. (a) SPH particle approximations for particle element *i* within the support domain *kh* of the kernel, including neighbor particle elements *j*. Reproduced from [16]. (b) Steady-state velocity profile predicted with SPH. Reproduced from [15].

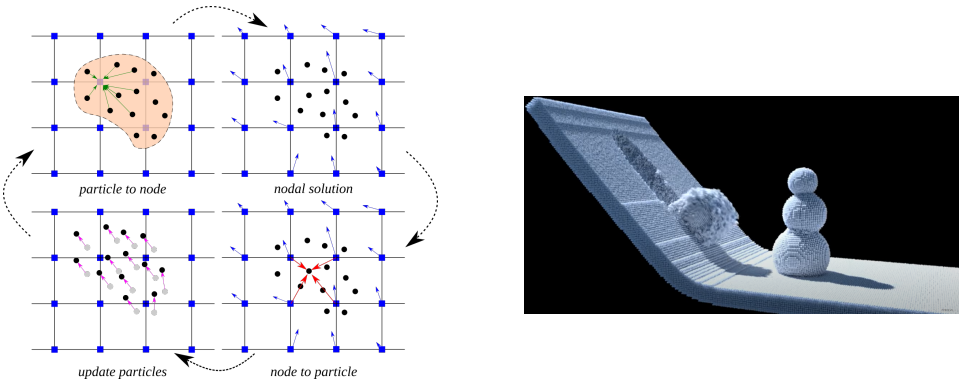
Disrete Element Method (DEM) The Discrete Element Method is similar to the SPH technique. Here, the interactions are not based on a kernel but instead on the contact and overlap of the boundaries of the discrete spherical elements. Various models can be used to model this contact behavior, including elasticity, plasticity, and failure [73]. Typical applications involve the simulation of granular materials. An example is provided in Figure 2.14a, where the forces and moments are modeled as a combination of spring and dashpot elements. Figure 2.14b shows the application of this method for simulating the mixing of steel and rock particles.



(a) (b)

Figure 2.14: The DEM model and an example of its application. (a) The linear contact model. Reproduced from [17]. (b) Mixing of steel and rock particles. Reproduced from [17].

Material Point Method (MPM) The Material Point Method is a hybrid Eulerian-Lagrangian technique that combines a mesh and particle representation of a material, which is very effective in large deformations. Figure 2.15a illustrates the algorithm loop, where material point properties are projected onto the mesh and solved. The updated kinematics is interpolated back onto the material points, completing the cycle. This technique is conventionally used for snow simulations, as shown in Figure 2.15b, where Disney applied this method for their new animated feature, Frozen [19].



(a) (b)

Figure 2.15: The MPM method and an example of its application. (a) Illustration of the MPM algorithm. Reproduced from [18]. (b) Snow simulation, used for the Disney movie "Frozen". Reproduced from [19]

.

Conclusion The present work addresses the flow of a polymer melt in a mold during the filling stage. The modeling framework must be chosen within this context for further research. The interaction of particles in the DEM framework is based on direct contact and its overlap. Longer-range interactions are not accounted for, which is essential for

modeling fluid-like behavior. While accurate, the implementation of MPM tends to be complex and, therefore, not ideal considering the timeframe and focus of this work. The SPH method is accurate, but computation time can be an issue. At last, FEM is a well-established method with extensive documentation and software support, making it easier to implement. It offers high accuracy for complex simulations, and with appropriate solver and parameter tuning, computation times can be optimized, making FEM a good trade-off for the research at hand.

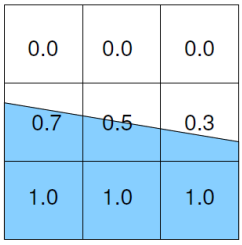


Figure 2.16: Volume-of-fluid. The alpha value for each cell, where $\alpha \in [0,1]$. Adapted from [20].

The simulation of melt entering an orifice involves a second phase: air. The volume of fluid (VOF) method is suitable in this context to simulate free-surface and two-phase flows [74]. Each cell contains a value between 0 and 1, indicating the presence of a fluid (0 = empty, 0 < 1 = interface, 1 = full), which refers to the volume fraction of a phase in a multiphase flow [20], as illustrated in Figure 2.16. Choosing the solver type and its convergence criteria is important to ensure stability and accuracy. For example, the PIMPLE algorithm is a combination of SIMPLE (Semi-Implicit Method for Pressure Linked Equations) and PISO (Pressure Implicit with Splitting of Operators) algorithms that offers robustness and stability for solving complex fluid flow problems [75]. It is designed to handle large timesteps while maintaining accuracy and can adjust corrector steps dynamically to balance computational effort and accuracy.
Conventional software packages can be used to implement all of the above. Some commercial options are: ANSYS Fluent [76], COMSOL Multiphysics [77], and Simulia Abaqus [78]. Moldex3D [79] and Moldflow [80] are designed to handle plastic injection molding simulations and are widely used in the industry. Popular Open Source Software options are OpenFOAM [81] (with e.g., the solids4foam toolbox [82]) and Elmer FEM [83].

FIBER SIMULATIONS

In the case of a thermoplastic composite melt, the polymer matrix is combined with fibers. This addition increases the complexity of simulating the mixture and raises the question of how to incorporate this second phase mathematically. Two frameworks are outlined as follows.

Average statistical methods In average statistical methods, individual fibers are not explicitly modeled. Instead, only the melt is simulated, and statistical methods account for the presence of fibers. Tuning is then based on phenomenological observations. In

other words, calculations are made at a scale that allows for reasonable computing time at the price of more or less accurate results. Hence, it deviates from perfection, but it is one of the better options to get insights about real-life parts whose size is too challenging for conventional microscale simulation.

Moldflow and Moldex3D use the Folgar-Tucker model to predict the orientation of fibers during melt flow. Initial assumptions (e.g., fiber volume fraction V_f, initial aspect ratio) and boundary conditions are imposed, while fluid rheology and dynamics are computed. This modeling approach statistically determines the fiber distribution and orientation throughout the mesh based on the melt velocity gradients.

A significant drawback of this model is the dependence of the fiber-fiber interaction on a single coefficient C_1, which is found by fitting it to experimental results [50]. There is no guarantee that this fitting will generalize to different geometries. It also depends on different closure approximation schemes [84, 85]. The Folgar-Tucker predictions may not match experimental data unless the needed fine-tuning and parametric studies are conducted based on the case at hand [86–90].

Next to orientation and distribution, the fiber attrition can also be accounted for. Long-fiber thermoplastics (LFTs) are prone to inaccuracies due to fiber breakage that occurs during the various stages of the process. Their length allows for deformation (bending, twisting) and failure. This is conventionally corrected by enabling post-processors, e.g., the phenomenological Phelps model [91], that predicts the fiber length distribution based on the previously computed rheological variables. This one, too, is based on a statistical method that requires a parametric study for each specific scenario that turns out to be time intensive [92].

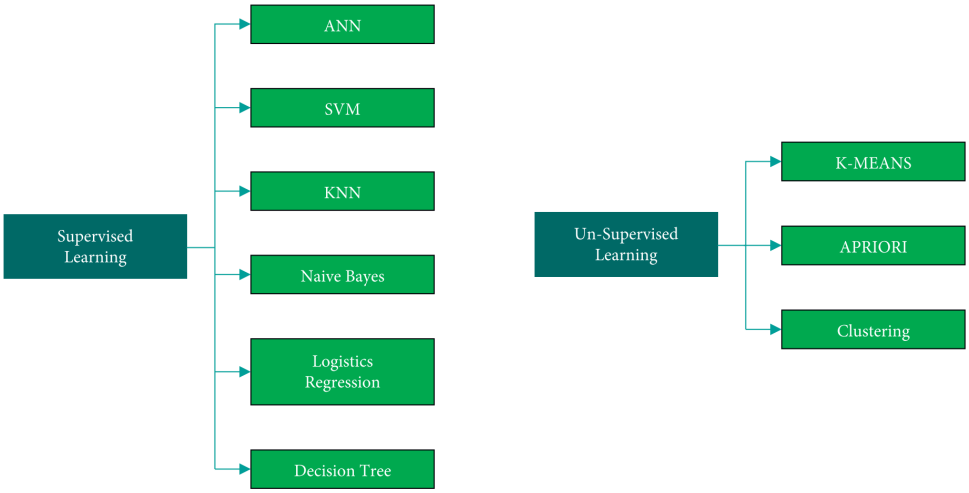
Software is constrained by the homogenization assumptions used for the fiber orientation distribution and the numerical capabilities of the software [93]. The predictions are often only valid for particular regions and are limited to a specific range of fiber concentrations of short fiber composites. The Folgar-Tucker model [94, 95] does not account for all fiber-fiber interactions and tends to overpredict the orientation evolution. Model parameters are often obtained from experimental results, requiring parametric studies for fitting values. This holds for both fiber orientation and breakage prediction. They also tend to be computationally demanding, which implies using simplifications to reduce computation time. Various experimental studies confirm these challenges [88, 89, 96, 97].

Direct microscale simulation In contrast to the statistical methods, direct simulation techniques simulate every fiber explicitly within the polymer melt. This allows for defining the physical behavior of the fiber and its mechanical response to applied loads and deformations. Interactions with the melt, other fibers, and the wall boundaries of the mold are also part of the modeling. For instance, fibers can be modeled as a series of spheres or rigid parts connected by tunable connectors. Various authors have contributed to this field of research [16, 98–106]. Although the direct methods grant a detailed insight about the mechanisms at play, they all require significant computational resources when applied to practical cases due to their microscale nature [50].

2.2.4. MACHINE LEARNING

Due to the complex nature of the parametric optimization process, efforts have been made to streamline existing methods, improve their efficiency, and enhance the accuracy and speed of optimization. The fact that conventional approaches are time-intensive justifies the need for faster monitoring [107, 108]. Using artificial intelligence for defect detection has shown to be an effective alternative [2]. Although the initial cost and effort to set it up are demanding, it is beneficial in the long term [109].

Machine Learning is a branch of artificial intelligence that focuses on developing algorithms to learn patterns from data and make predictions for unseen data, performing those tasks without explicit instructions. This process is often referred to as inferring the latent space [110]. Variables are derived through mathematical models based on directly observable and measurable variables. The learning and predictive process is possible without prior knowledge of the underlying physical nature. Their efficiency and versatility show their promising potential [2]. It can be subdivided into supervised and unsupervised learning, as depicted in Figure 2.17.



(a) Supervised learning.

(b) Unsupervised learning.

Figure 2.17: Machine Learning algorithms. Reproduced from [2].

UNSUPERVISED LEARNING

In unsupervised learning, predictions are derived from a function based on the statistical structure of unlabelled input data [111]. The algorithm is trained to analyze datasets and discover hidden patterns. Unsupervised learning is mainly implemented for the following three tasks:

1. Clustering: Unlabeled data is categorized into clusters based on their similarities or differences.

2. Association: Rules are used to identify relationships between variables within a

dataset.

3. Dimensionality reduction: When the dataset exhibits many features or dimensions, this method reduces it to a reasonable size while preserving its identity.

This project aims to train an algorithm that relies on labeled ground truth data to assess predictions' accuracy. Consequently, unsupervised learning is not of interest in this research.

SUPERVISED LEARNING

These algorithms can predict results based on patterns learned during the supervised learning phase. The main requirement is the presence of annotated (labeled) training data. This approach aims at optimizing a set of mathematical functions, such as given an input, it produces the desired outputs, focusing on minimizing a central cost function. Ultimately, it uses this to predict the output for a new set of unlabelled input data [112].

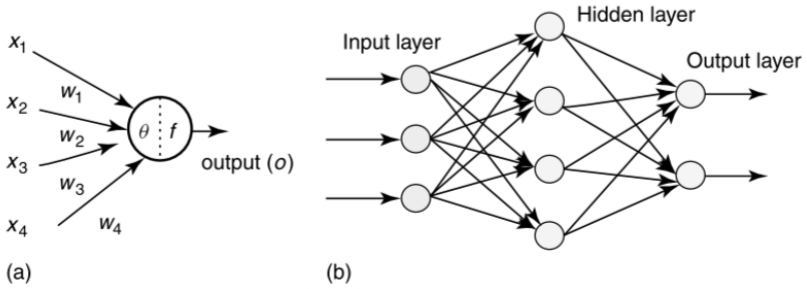


Figure 2.18: Reproduced from [21]. (a) Architecture of an artificial neuron. (b) A multilayered artificial neural network.

Artificial Neural Networks (ANNs) Artificial Neural Networks (ANNs) are the most commonly used algorithms in the parametric optimization of the injection molding process [2]. An artificial neural network (ANN) is a type of ML model that is biologically inspired. The initial idea, originating in the 1980s, is that it tries to mimic the human brain's structure, function, and learning procedure. However, due to its progression and increasing complexity over the years, these models have diverged significantly from this initial idea [113]. The core concept remains: analogously to neurons, ANNs consist of multiple layers of interconnected nodes that take input data, learn from it, and roll out the required output (see Figure 2.18). Numerous approaches and architectures have been developed up to date.

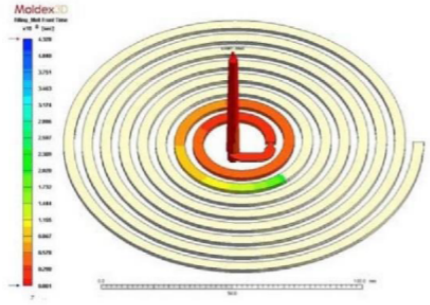
In a feed-forward network (a type of ANN without feedback loops), information flows from left to right through three layers: the input, the hidden layer(s) (latent space), and the output layer. Data processing occurs in the latent space, which can extend over multiple hidden layers, conventionally called a Multilayer Perceptron (MLP) [21]. The nodes are the basic building blocks of this approach. Each node receives inputs $x_1, ..., x_N$ with

respective weights $w_1, ..., w_N$ that determine the effect of the associated input signal. These weights represent the memory of the network. Learning is achieved by adjusting them according to a pre-defined learning algorithm. If the weighted sum of inputs exceeds the threshold value θ, the neuron activates and produces an output O [114]:

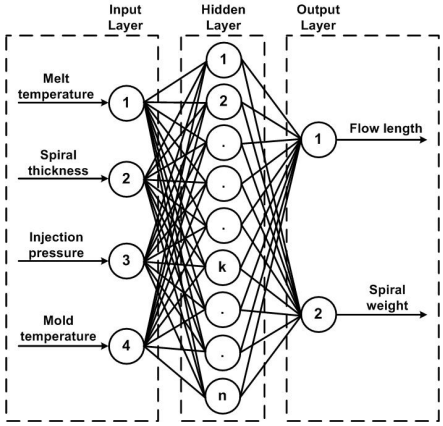
$$O = f(net) = f\left(\sum_{i=1}^{N} w_i x_i\right), \tag{2.11}$$

where $f(net)$ is the activation function.

A recent study by Selvaraj *et al.* [2] summarises Machine Learning Models (primarily ANNs) used to optimize injection molding machines. They found that various models have been employed over the past years, mainly through MATLAB and C-MOLD software programs. The authors concluded that ML algorithms are employable in many ways from start to finish of the injection molding process. DOEs can be used to generate the necessary training data, followed by its optimization with ML. This research confirms their potential within this framework. Two examples are presented as follows.



(a)



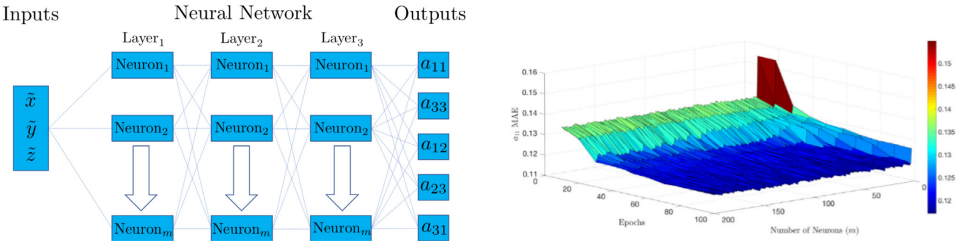
(b)

Figure 2.19: Adapted from [22]. (a) HDPE melt flow length, Moldex3D. (b) The architecture of the ANN for the prediction of flow length.

In a study by Stoica *et al.* [22], an ANN was designed and trained with simulation data from software packages and experimental data from DOEs. Polyethylene (PE) is injected at a constant flow rate into the spiral, as shown in Figure 2.19a. The melt flow length is monitored as a function of four variable processing parameters. The ANN is trained with these parameters as input, shown in Figure 2.19b, and its predictions are compared to the ground truth data. It was found that the ANN could predict the flow length with a mean relative error < 10%, where the model trained with Moldex3D data excelled in accuracy compared to the MoldFlow and DOE cases.

Similarly, the work of Sabiston *et al.* [23] shows the potential of combining ANNs with experimental results, using (tomography) fiber orientation training data to predict the fiber orientation within compression molded parts. The architecture is displayed in Figure 2.20a. Here, the inputs $(\tilde{x}, \tilde{y}, \tilde{z})$ are normalized positions of where the fiber orientation tensor is evaluated, and a_{ij} the predicted orientation tensor components. The effect of hyperparameters, such as the number of epochs and neurons, on the convergence of the model was investigated, as shown in Figure 2.20b.

The ANN was able to predict the orientation of tensor components within the variability orientation of neighboring microstructural units. The time needed to make these predictions is less than 1 second, which is insignificant compared to the approximately ten-hour training process.



(a) (b)

Figure 2.20: Adapted from [23]. (a) Schematic of the ANN. (b) Contour plots of the MAE prediction of the fiber orientation tensor a11 for the first 100 epochs.

Overall, using machine learning has its benefits, including reduced errors, time efficiency (reduced repetitive processes), requiring less computational power (training excluded), and occasionally being considered a profitable option in the long run. However, one must remember that the initial setup for each case tends to be time-consuming and computationally demanding. This should not be underestimated. However, once the model is trained, it has shown great potential, mainly accelerating optimization tasks.

2.3. MACHINE LEARNING & FLUID SIMULATIONS

Since modeling the injection molding process is essentially a fluid simulation, in this section, we take a step back from injection molding and look at the general implementation of Machine Learning for fluid simulations. The conducted research in the past shows that the automotive, combustion, acoustics, and aerodynamics industries are some examples that have profited from involving machine learning in the respective fluid simulations [115]. Its contribution in these areas is quite alike: the existing experimental and computational methods are combined with data-driven machine learning algorithms to reduce computational costs and increase the efficiency of the methods [116]. Various models that deal with complex nonlinear fluid dynamics and large data sets have been developed. Optimization problems, for instance, profit from this combination since they are otherwise expensive, time-intensive, and challenging to solve with traditional methods. The application of ML to fluid flows applies to domains like analysis, modeling,

dimensionality reduction, superresolution, sensing, and estimation.

The computational pipelines discussed in Equation 2.2.3 can profit from ML. Solving Navier-Stokes equations often requires a trade-off between computational speed and accuracy. The work of Wandel *et al.* [117] showed that both can be achieved with the help of a Neural Network surrogate that learns incompressible fluid dynamics in 3D. The authors introduce a physics-informed loss function that penalizes residuals of the Navier-Stokes equations. The model showed significant improvements compared to current NN-based fluid models. Furthermore, it can include fluid phenomena (Magnus effect, Kármán vortex streets) and generalize to new domains.

A similar approach was applied in the Finite element method-enhanced neural network study of Meethal *et al.* [118]. The algorithm combines the residuals from finite element methods with custom loss functions from neural networks to solve forward and backward PDE problems. Computational cost is reduced compared to conventional supervised learning, and this method opens up a way to use this hybrid approach to make inexpensive surrogate models.

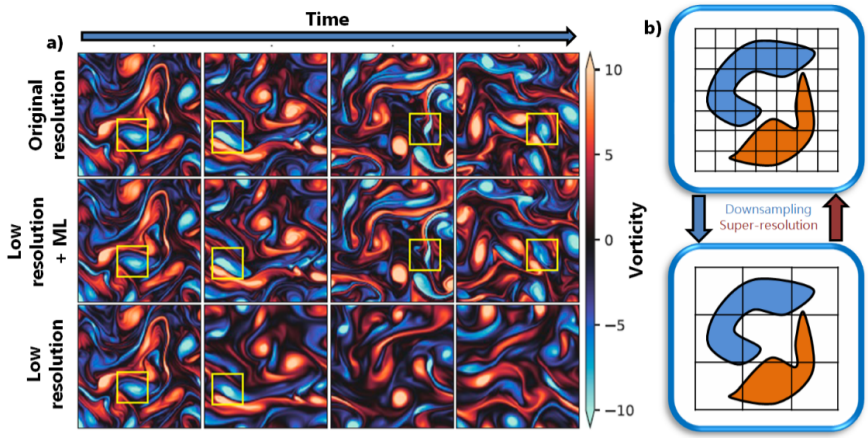
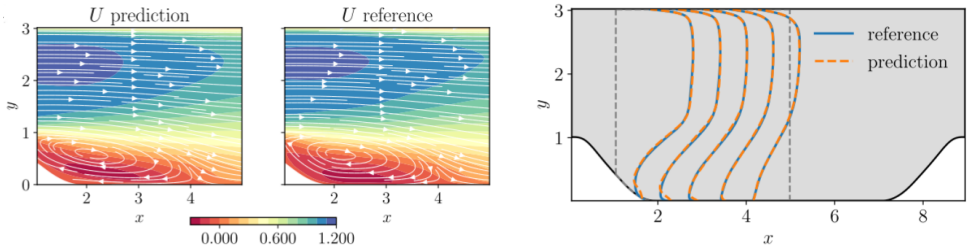


Figure 2.21: Adapted from [24]. a) Vorticity field for high-/low-resolutions, and the case with low resolution supplemented with ML. Key vortical structures are highlighted in yellow. b) ML accelerates simulation by recovering the details from downsampled flow data through super-resolution.

The application of ML for a Direct Numerical Simulation (DNS) acceleration super-resolution task is presented by Kochkov et al. [119]. The length scale of the smallest features in a turbulent fluid flow is $O(10^{-6})m$ [120] and classical methods (CFD) force disadvantageous trade-offs between computational effort and accuracy. This motivated the authors to replace the components of these solvers most affected by accuracy loss with deep learning. The results are illustrated in Figure 2.21. Two-dimensional flow data is generated using a coarse mesh (saving computation time), followed by a learned ML algorithm that interpolates local features. Compared to the original method, a 10x coarser resolution and an 80-fold improvement in computational time were achieved. This gain was obtained

without any degradation in terms of accuracy and generalization. However, this work is limited to CNNs combined with traditional solvers. The geometries (cylinders, channels) are also quite simple, relying on specifically turbulent inflow conditions.

Physics-informed neural networks (PINNs) can also be used for turbulence modeling. This framework was introduced by Raissi et al. [121] and is a type of supervised neural network that can be trained to solve partial differential equations (PDEs) enforced through a loss function while taking the given laws of physics into account. Eivazi et al. [25] used PINNs to solve the Reynolds-averaged Navier-Stokes (RANS) equations based on boundary data without any model or assumptions for turbulence. The model is first applied to laminar flow and then to four turbulent flow cases. One of these is the simulation of turbulent flow over a periodic hill, shown in Figure 2.22.



(a) (b)

Figure 2.22: Adapted from [25]. (a) Contour of U and flow streamlines for the PINN prediction and the reference data. (b) U profiles at five different streamwise locations.

Only the DNS data on the domain boundaries and the RANS equations were used to train the supervised part of the model. Figure 2.22b shows the agreement between the PINN predictions and the reference data. The authors demonstrate the effectiveness of PINNs with both laminar and turbulent conditions. However, the performance has not been assessed yet when dealing with non-Newtonian fluid dynamics (as with polymers), nor with more complex and irregular geometries (as encountered in injection molding). Additionally, the solver is limited to the Reynolds-Averaged Navier-Stokes (RANS) equations for incompressible turbulent flows.

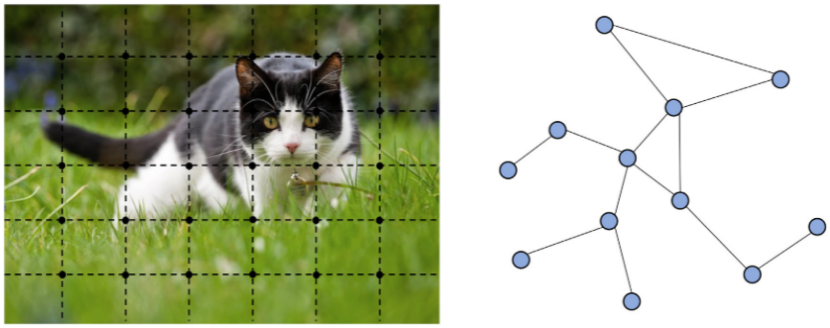
Other examples include, for instance, the development of Reduced Order Models (ROM) [122] and dealing with noisy data from experimental fluid mechanics [123].

As shown above and in subsection 2.2.4, combining ML with the existing methods into hybrid approaches can be a potent tool. Often, computational costs and convergence times are reduced, increasing the overall efficiency of the models. Although the involvement of ML seems to be primarily advantageous, one should also consider its costs. Large amounts of training data are often needed, and the training phase is usually computationally expensive. While it can successfully replace specific tasks within a process, it cannot single-handedly replace the entirety of a process. Additionally, ML does not provide any additional physical insights, so it is often classified as a black-box approach.

The question that remains at this point is the following: Is Machine Learning only capable of making predictions by inferring the data latent space, hence only recognizing patterns in data without capturing the inherent governing physical laws, or does it also have the capability of capturing the physical process on its own? Is there a data-driven alternative that can learn and simulate the underlying physics of the process? Regarding this matter, Graph Neural Networks (GNNs) have recently gained a lot of attention. Indeed, GNNs theoretically feature the required tools to learn physics. This makes them attractive to investigate within the scope of this work. However, as usual with any other ML architecture, its potential and limitations should be critically examined, as well as its interpretability, generalizability, and explainability. These topics are the backbone of this work and are presented in detail in the following section.

2.4. GRAPH NEURAL NETWORKS

Much attention has been devoted to Graph Neural Networks (GNNs) over the last few years. It is being applied to areas such as biology [124, 125], programming code [126], traffic prediction in Google Maps [127] and the most relevant for the present research, physics simulations [31]. In the next two sections, the basic principles of GNNs are explained, with examples of data that can be phrased as a graph and their difference compared to other types of data. Since a significant amount of research has recently been devoted to GNNs, the second section focuses on their latest developments and applications. The attention is directed toward the field of interest within this context, fluid rheology and dynamics prediction. Several research teams from various companies, universities, and research centers contribute to this exciting field of research. The relevant papers and advancements are described down below. Notably, this section concerning Graph Neural Networks is the main focus of this literature research. It aims to assess developing and implementing a GNN to simulate the injection molding process specifically as future work. For this reason, the theory and examples are critically examined to evaluate its potential within this framework.



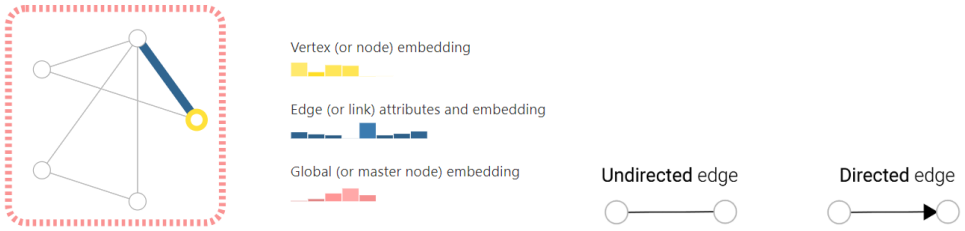
(a) (b)

Figure 2.23: Adapted from [26]. (a) Image in Euclidean space. (b) Graph in non-Euclidean space.

2.4.1. BASIC PRINCIPLES

The main idea of a graph-based approach is to discretize, e.g., a physical space into a set of nodes with relations (edges). The three basic types of attributes are nodes V (node identity, number of neighbors), edges E (e.g., edge identity, edge weight), and global U (e.g., number of nodes, longest path). Information can be stored and exchanged in and between each attribute. The type of data at hand and the objective functions determined by the user define the details of this stored information and how the attributes communicate with each other. Images, for example, are, in essence, symmetric, structured, grid-like graphs [27]. The resemblance is illustrated in Figure 2.23. They can be regarded as a simple graph representation, where the pixels are nodes with edges that represent their adjacency with surrounding pixels. This information is stored in the adjacency matrix.

In practice, these graph representations excel when dealing with heterogeneously structured data (without a structured grid), meaning that nodes have variable numbers of neighbors. This is classified as a non-Euclidian space (in contrast to the image example, in which case we talk about an Euclidian space) [26]. In some cases, the number of neighbors and type of edges vary over time. Social networks, molecules, citation networks, and knowledge graphs are some example domains with this kind of structure [128].



(a) (b)

Figure 2.24: Attributes and edge types of a graph-based architecture. Reproduced from [27]. (a) The three types of attributes: Nodes, Edges and Global. (b) Information flow, depending on the nature of the edge.

Edges can be directed or undirected, as shown in Figure 2.24b. If directed, information flows from a source node to the destination node. In the undirected case, information may flow both ways. Typical tasks that can be performed with graph-structured data are listed below.

1. **Graph-level**: Prediction of a property for an entire graph, e.g., determine the potential energy of a physical system [129] or a chemical property of a molecule based on its attributes [130].

2. **Node-level**: Prediction of the identity/role of each node. This is similar to image segmentation, where a label is designated to each pixel (node) consisting, for instance, of RGB values. In the context of a fluid, each particle (node) can be given attributes, e.g., a mass or velocity [31].

3. **Edge-level**: Prediction of the relationship between nodes. This determines if two nodes interact, and if yes, what the contact properties are [131].

The implementation of this architecture presents specific challenges. The adjacency matrix for a graph network that determines its connectivity can be relatively sparse. The nodes and number of edges per node can vary, leading to very few entries and making it space-inefficient. However, this issue can be tackled by replacing adjacency matrices with adjacency lists [27]. Additionally, one specific connectivity can be represented by more than one adjacency matrix [27]. It means that the representation of connections between nodes in an adjacency matrix changes when the order of nodes is rearranged, even though the actual connections stay the same.

2.4.2. GNN FUNDAMENTALS

Battaglia et al. [30] and Gilmer at al. [132] introduced the message passing concept and the Graph Nets architecture that are used here to outline the procedure of a GNN. Fundamentally, GNNs take a graph as input and output a modified graph. The graph carries information as node V, edge E, and global U features. This information is embedded and used for the learning process using, for instance, a multilayer perceptron (MLP) layer. Such a layer is also named a GNN layer and can be stacked together analogously to other conventional neural network models.

Predictions are made based on an exchange (*pooling*) of information between the various attributes, passed from one layer to the next (*message passing*):

1. Each attribute's embedding (properties of the surrounding attributes) is stored in a matrix.

2. Embeddings are combined via an aggregate function (e.g., max, mean, sum) operation.

3. Neighbouring nodes and/or edges update each other embeddings (every message is run through an update function).

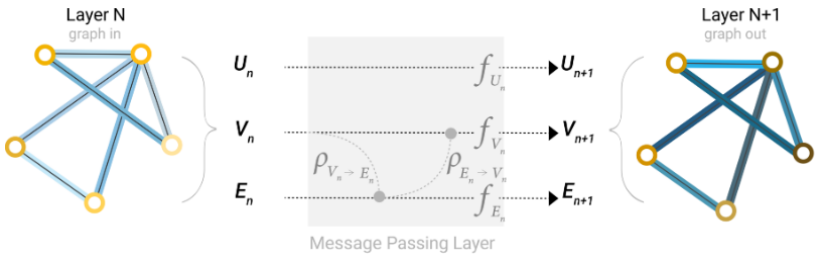


Figure 2.25: Architecture schematic for the Message Passing layer. The first step "prepares" a message composed of information from an edge and its connected nodes and then "passes" the message back to the node. f is the update function and ρ the pooling function. Adapted from [27].

Pooling consists of the first two steps; the third corresponds to the message passing. This process enables the exchange of information between nodes and edges, e.g., if the data are stored in edges and predictions are to be made for the nodes. Figure 2.25 illustrates a possible implementation of this concept. Here, the edge is updated with its embedding and then passed back to the node. The order of this update procedure and the features (nodes, edges, graph) may vary.

Information can be transmitted across several message-passing layers, also called the *depth*, (Graph Convolutional Network (GCN)), and is essential for training purposes of the GNN. The number of layers determines how far this message passing goes within the graph. If the network has k layers, it means that a node contains information of the nodes that are at most k steps away from it. This gives the user some control over establishing the maximum (cut-off) distance above which two nodes cannot transfer information.

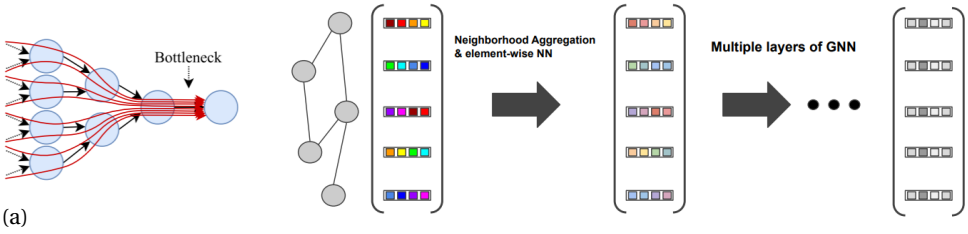
Changing the properties of an attribute or how they interact often leads to spin-offs of the original graph neural network [27]. Multigraphs, for instance, contain different types of edges, which means that depending on the type of nodes interacting, the properties of the connecting edge may differ [133]. Single nodes can also represent an entire (sub-)graph, classified as hypernode (or nested) graphs [134]. This makes it possible to store and train based on hierarchical information. Hypergraphs contain edges that can be connected to multiple nodes at once. These different GNN architectures are the subject of ongoing research [135, 136].

2.4.3. INCREASING THE COMPLEXITY AND CHALLENGES

Sanchez-Lengeling *et al.* [27] have investigated the effect of architecture on model performances, varying the number of layers, type of aggregation function, style of message passing, and the dimensionality of embeddings. Although the effects vary significantly depending on the case and the given data [137, 138], trends could be identified. A higher number of trainable variables results in a higher performance, and a higher dimensionality correlates with better mean and lower bound performance. This does not hold for the maximum bound. An interesting observation is made when looking at the performance of models with different numbers of layers. The trend shows that more layers do not correlate with better performance. Instead, the two-layer architectures are reported to be the best performing when compared to one-, three- and four-layer ones. Concerning the type of aggregation function, all three max, mean, and sum exhibit similar performances. A key architectural choice that stands out is the message passing. Generally speaking, the more the nodes, edges, and globals interact, the better the performance [27].

Two critical risks are acknowledged with graphs [28]: over-squashing and over-smoothing. When dealing with molecules, for instance, the interest often lies in long-range global predictions. With an increase in the number of layers and interactions among nodes, information coming from far away ends up being compressed. Consequently, this significant volume of information is aggregated and conveyed through a single node, posing a challenge for it to express such extensive information effectively. This phenomenon is described as over-squashing and illustrated in Figure 2.26a.

The opposite occurs when dealing with short-range interactions, shown in Figure 2.26b. If the number of message-passing steps is relatively small, information travels over short paths and cannot convey the system's underlying physical behavior. As a result, node representations start to resemble each other with increasing layers. As reported by Corso et al. [139], this is referred to as over-smoothing.



(a) (b)

Figure 2.26: (a) Over-squashing. The receptive field of each node grows exponentially. Adapted from [28]. (b) Over-smoothing. Node representations become indistinguishable from each other. Adapted from [29].

2.4.4. RECENT DEVELOPMENTS AND APPLICATIONS

GNNs are a new development compared to more established ML algorithms such as CNNs. Therefore, their application to physical problems is recent. While they represent a promising technique, they also come with drawbacks that have been discussed extensively but not fully addressed yet. For instance, generalization and error metrics. In particular, readers are referred to publications by the Google DeepMind [140] research-group and related work that pioneered this field [129, 131, 141–144].

RELATIONAL INDUCTIVE BIASES

Battaglia *et al.* [30] introduced the concept of relational inductive biases, emphasizing their importance in deep learning architectures for their potential with combinatorial generalization. This idea is a cornerstone of the GNN architecture used in this work. Inductive biases, defined as the ability of an algorithm to prioritize certain solutions irrespective of the considered data, enforce limitations to the relationships and interactions among entities during the learning process. These constraints can be regularization terms [145] or specific architecture choices. Table 2.6 illustrates four examples.

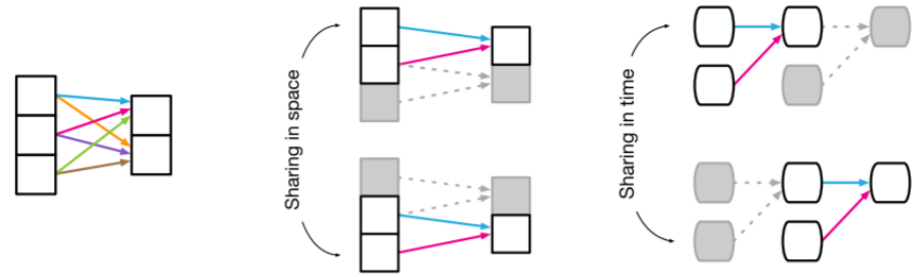
The fully connected case allows information to propagate from all units in a layer to all units in the next layer. There is no reuse, and information is not isolated. Therefore, the relational inductive bias is weak since all inputs can influence all output units' values.

In CNN and RNN's cases, translational and temporal invariance are introduced, respectively. Simply speaking, for a CNN, the absolute position of a feature within an image should not matter as long as the relative position is conserved (translation and rotation invariance); analogously, the outcome should not depend on the time for an RNN. These frameworks are visually illustrated by Figure 2.27.

Table 2.6: Various relational inductive biases in standard deep learning components. Reproduced from [30].

Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

The foundation of GNNs lies in understanding whether and how nodes exchange information. This approach makes it less responsive to variations in the number or composition of nodes and, instead, directs attention toward the interaction among physical entities. A graph structure, by its nature, inherently incorporates spatial locality and relational inductive biases. These can additionally be introduced through architectural design and training techniques, positively impacting performance, training time, and generalization of the model [27].



(a) Fully connected (b) Convolutional (c) Recurrent

Figure 2.27: Reproduced from [30]. (a) A fully connected layer, where the weights are all independent without information sharing. (b) A convolutional layer, in which a local kernel function is reused multiple times across the input. Arrows with the same color indicate shared weights. (c) A recurrent layer, in which the same function is reused across different processing steps.

ENCODER-PROCESSOR-DECODER FRAMEWORKS

GNNs have been used to simulate physical systems [30]. A common GNN architecture for this type of application is the Encoder-Processor-Decoder framework. The Encoder is responsible for constructing the latent graph from the input state. The Processor propagates information through the network, updating the graph via message passing. Finally, the Decoder extracts the dynamics information back into the physical realm. Multi-layer perceptrons (MLPs) act as the encoder and decoder, while layers of GN blocks take on the processor role. A GN block is a "graph-to-graph" module that accepts a graph as input, performs computations, and produces a graph as output [30]. Three notable works

that rely on this scheme have made substantial contributions to this field. Their details and specific architectures are presented below.

1. **Graph Network Simulator (GNS)**

In 2020, Sanchez-Gonzalez *et al.* [31] proposed the simulation of complex physical phenomena with GNN models. They introduced the Graph Network-based Simulator (GNS) framework, where the material (sand, goop, water, among others) is discretized into a set of particles (nodes), and the edges represent the interaction of those nodes with one another. During the training phase, the GNS model learns the material behavior and dynamics through message-passing. Different materials were used, ranging from fluid to deformable and rigid solids.

The Encoder-Processor-Decoder schematic is illustrated in Figure 2.28. The Encoder is responsible for constructing a latent graph from the input state with the help of a Multilayer perceptron (MLP). Node features (e.g., acceleration) and edge features (e.g., distance between two particles) are encoded into the latent space. Message-passing steps determine the connectivity radius of the nodes. Interactions are established by adding edges between the respective particles. The Processor stacks a number of GN blocks (graph input, graph output) that predict the system's future state. The Decoder transcribes the dynamics information back into the physical domain (e.g., predicted acceleration for velocity and position).

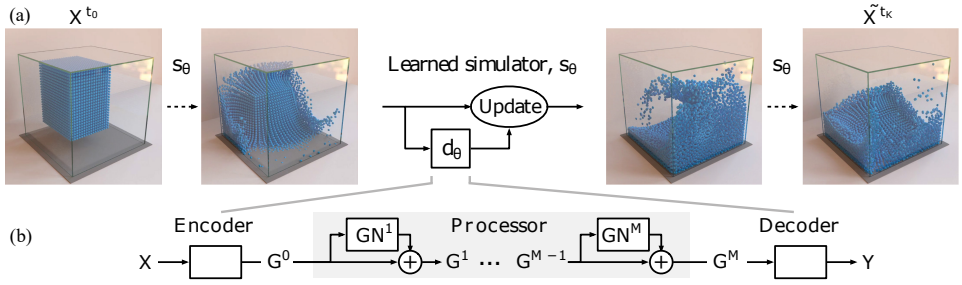


Figure 2.28: (a) The GNS predicts future states using its learned dynamics model d_θ. (b) The d_θ uses an "encode-process-decode" scheme, which computes dynamics information, Y, from the input state, X. Reproduced from [31].

Nodes and Edges are processed with the same MLP, which provides the GNS its global physical simulation power. The robustness of the model is ensured by adding noise to the input velocities. The model was exposed to different initial conditions, timesteps, and particles during testing. It was able to generalize to scenes containing an order of magnitude more particles and up to 10x more time steps than seen during training, ensuring visually similar dynamics between the prediction and ground truth.

2. **Constraint-based Graph Network Simulator (C-GNS).**

This framework shares similarities with the GNS but distinguishes itself in the Processor, where the simulator learns the constraint function landscape and uses its minimum to

define the implicit solution [32]. The Encoder and Decoder operate like the GNS model. The learned simulator was assessed for various physical domains: ropes, bouncing balls, rigid, and splashing fluids, as shown in Figure 2.29.

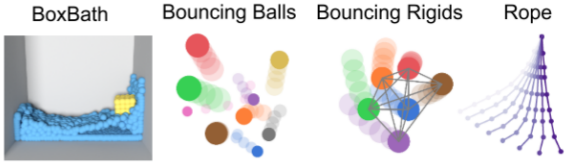


Figure 2.29: Renderings of the physical environments. Reproduced from [32].

The learned constraint coupled with the implicit approach outperformed previous explicit forward models [31, 33] in handling systems characterized by robust interactions and rigid, stiff dynamics.

3. **Meshgraphnets**

Pfaff *et al.* [33] proposed a similar GNS architecture applied to a mesh-based simulation.

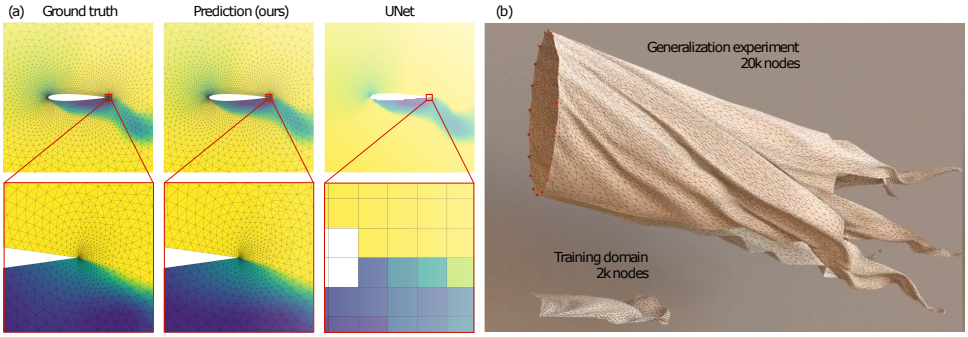


Figure 2.30: (a) Rollout of the GNS model versus ground truth versus the grid-based U-net baseline. (b) The GNS model can be scaled up to significantly larger and more complex setups than seen during training. Adapted from [33].

Here, the encoder is responsible for encoding mesh information into Eulerian systems (e.g., velocity over a fixed mesh) and Lagrangian systems (e.g., plate deformation). The Processor is similar to the GNS, and the Decoder predicts the next state via Euler integration. Figure 2.30 shows rollouts of the GNS for the Airfoil and FlagDynamic model. For the airfoil, the model can accurately predict dynamics via adaptive meshing. The dynamic flag model is trained with 2k nodes and can be generalized for significantly larger and more complex cases.

LOSS FUNCTION

Quantitative metrics are essential to evaluate the respective performances of GNNs for a given dataset. It determines how well the model learns and adjusts the weights through

backpropagation. The examples mentioned above make use of single scalar metrics, often resorting to the Mean Squared Error (MSE) or the Root Mean Squared Error (RMSE), as shown in Table 2.7.

Table 2.7: Frameworks and their corresponding error metrics. The minimum and maximum error values are for one rollout of the whole trajectory.

Framework	Error metric	Min. value	Max. value
GNS [31]	MSE	5×10^{-4}	2×10^{-2}
C-GNS [32]	MSE	6×10^{-3}	0.3
Meshnet [33]	RMSE	1.5×10^{-2}	11.5

Other studies that focus on simulating complex physics use similar approaches [146–149].

The question remains: how well does a single scalar quantity represent the accuracy of a model (that contains a vast amount of information)? Unlike computational methods, learned simulators rely on this quantity to direct the algorithm in the right direction. To confirm this, the rendered ground truth and prediction models can be compared, and indeed, there is an agreement between the two [31–33]. Still, this does not ensure an equally good representation of the accuracy when applying this metric type to other physics simulations. This allows for adapting, improving, or redefining the error metric concept.

2.5. KNOWLEDGE GAP

The previous section shows the potential of GNNs to simulate complex physics, ranging from water sloshing to rigid body interactions. The simulation of the injection molding process in this work deals with the injection stage, where the screw pushes forward as a piston and forces the polymer melt into the closed mold, as described in subsection 2.1.1. It is a (viscous) fluid simulation comprised of a pure polymer or combined with a second phase (fibers). In this literature review, it was found that the GNN framework has been used for similar cases (water sloshing, viscous fluid simulations), but its implementation for injection molding, in particular, is uncertain. This yields the 1st research question:

1. Can a GNN learn the underlying physics of the injection molding process?

In addition to simulating the process, its performance must be assessed by comparing it with conventional methods. As simply expressed by Kochkov *et al.* [119], also in this work, a utilitarian perspective is taken on model evaluation, where (learned) simulation methods are good insofar as they demonstrate accuracy, computational efficiency, and generalizability. This yields the 2nd and 3rd research question:

2. How does it compare to conventional methods?
3. How does the learned simulator perform in terms of generalizability?

The error metrics discussed in section 2.4 compress substantial information into one scalar quantity. This does not automatically ensure its understanding of the underlying physics. The loss value must be tailored accordingly. This yields the 4th research question:

4. To what extent does the error metric correctly represent the simulator's accuracy?

2.6. DISCUSSION

The conducted literature research, and future work is carried out through the collaboration of the TU Delft and TNO Brightlands Materials Center. From a scientific point of view, the same vision and interest are shared. From the industry's point of view, the economic side of this research is added as an additional factor. Therefore, the discussion is divided into two respective subsections.

2.6.1. SCIENTIFIC APPROACH

Graph Neural Networks have shown their potential within the framework of this research. Conventional methods to simulate the injection molding process and optimize processing parameters exhibit drawbacks such as computational costs or efficiency that can be mitigated by combining them with Machine Learning into hybrid approaches. The power of GNNs, however, lies in their ability to capture the whole physical process on its own. The off-line training phase (with a pre-collected dataset) tends to be computationally demanding, but the computational costs associated with running a learned simulator can be one to two orders of magnitude faster compared to its traditional counterpart [31–33].

A major challenge for applying a GNN to this problem is the specific choice for the hyperparameters and input settings. Depending on the specific scenario, the optimal parameter settings may vary. This setup is time-demanding, and so is the training phase, including the ground truth data generation. Researchers often use powerful CPU and GPU-based setups, which might also be needed in this case to accelerate the process.

As stated in the research questions, the model's generalizability is an important issue to address. In the long run, the learned simulator should be able to tackle different problems with increased complexity. Without this quality, its utilization would lack significance. The limited physical interpretability and uncertain generalizability are recognized drawbacks of data-driven approaches that must be tackled to allow this technique to compete against physics-based methods.

This work uses a GNN to learn the polymer melt behavior under certain processing conditions. Analogously to the mechanical behavior of materials, ideally, this model is to be trained on a small scale and learn the physics of a material, to then be upscaled and applied to different situations, bypassing the long computation times of conventional software. In the long term, the aim of this project is to establish a novel graph-based framework capable of efficiently simulating complex physics across various material models and processing conditions, where this study introduces the preliminary phase.

To the author's knowledge, applying a GNN specifically for injection molding simulation

has not been done before.

2.6.2. ECONOMIC PERSPECTIVE

The industry relies heavily on software specifically designed for injection molding (e.g., Moldex3D, Moldflow). It offers enough freedom to design and implement a model in an accessible way with an easy-to-use interface. Both software packages were developed in the late 1900s and have since been advancing with their capabilities to simulate these processes. Users pay for a yearly license that varies between $6,000 - 25,000$ euros [79, 150]. The impact on big tech companies may be modest compared to smaller entities, yet it persists.

The codes developed for the GNN are part of the Machine Learning computer science philosophy, complying with its open-source mentality. Often, codes are made readily available on public domains such as GitHub. Developing or adapting existing algorithms does not involve any economic burden. If a trained simulator succeeds, ensuring its accuracy and faster computation time compared to conventional software, the industry would have a motive to resort to this technique instead of using costly software packages.

2.7. CONCLUSION

The proposed method to simulate the injection molding process has great potential based on the literature findings, but its limitations are important to take into account. Hyperparameter settings, the specific architecture, interpretability, and generalization, among others, must be critically investigated. Data-driven approaches like these are often appealing due to their inherent ability to reduce computational effort. However, there are some hurdles to overcome to achieve a realistic and trustworthy model. If demonstrated to function effectively, it has the potential to redefine the way we model complex physics, not only in an academic and experimental setting but also within its application in the industry. This work is meant to serve as an important cornerstone, identifying key components and factors within this GNN framework that hereafter can be employed to guide future developments, enhance understanding, and drive progress in the field.

3

GROUND TRUTH

THIS chapter explains the pipeline created to generate the ground truth data[1]. Together with the training of the graph neural network, these are the two essential building blocks of this work. The writing style is designed to provide the reader with the necessary tools and accompanying codes[2] to directly replicate the process.

In this work, the focus lies on 2D mold filling simulations, where the following components are varied to generate the different cases:

- The geometry
- The inlet location
- The outlet location

The following steps are required to generate the ground truth data: build mold geometries, mesh and pre-process them, run the solver, and post-process the results. Ultimately, the goal of this chapter is to create a dataset for the training of the Graph Neural Network. Material properties, solver settings, and other specific parameters are tweaked along the way with the help of a mesh convergence study to ensure numerical stability and realistic material behavior. Both are crucial for the learning process of the neural network.

The philosophy behind the present research is inspired by Pfaff *et al.* [33], where the authors constrained the training data to a limited number of nodes and a fixed simulation time. Generalizability is investigated by testing the learned simulator with more complex and larger, unseen geometries or by predicting the model states at time steps after the time steps used as training data. A similar approach is taken in this work, where the training phase involves relatively simple geometries and a maximum simulation time of 1 sec, and the testing is done using bigger, more complex geometries.

[1] A detailed view can be found in Appendix B.
[2] The repository is currently held private. Access may be granted upon request.

As Gmsh [151] and OpenFOAM [81] offer a dimensionless environment, the International System of Units (SI) is used as listed in Table 3.1.

Table 3.1: SI units used for Gmsh and OpenFOAM.

Quantity	**SI**
Length	m
Force	N
Mass	kg
Time	s
Stress	Pa (N/m^2)
Energy	J
Density	kg/m^3

3.1. GENERATING GEOMETRIES AND MESHING

The first step is to create a randomized collection of mold shapes. The Application Programming Interface (API) of Gmsh (an open source 3D finite element mesh generator [151]), is consulted within a Python [152] environment. Six different convex polygons are generated with N sides, where $N \in [3,4,\ldots,8]$. The location of each vertice is chosen randomly over a circle centered on $C(3.5,3.5)$, as shown in Figure 3.1. It has a radius of $r = 3.5cm$ and lies in the 1st Quadrant of the Cartesian plane. For this example, $N = 3$ points are randomly generated on the circle in a clockwise sequence.

A minimum distance d_{min} between vertices is enforced, to ensure a relatively large and consistent surface area across all polygons of this order. The area range is about $15 - 35cm^2$ for the 3- to 8-sided polygon, respectively. d_{min} is calculated as in Equation 3.1 which was established empirically.

$$d_{min} = 4.5\left(\frac{r}{N}\right) \tag{3.1}$$

Next, the x and y-coordinates of the injection location are randomly generated in the area enclosed by the polygon boundaries. A clearance of $c = 0.3$ cm is set with regard to the nearest boundary to accommodate the future injection cut-out radius. The coordinates are saved to a CSV file. An example is shown in Figure 3.2.

The final step is the meshing of the shapes. First, the points are connected to create segments. The mesh size in the x,y- plane and the number of slices in the thickness (z-direction) is determined by the mesh convergence study, see section 3.3. Gmsh provides several meshing algorithms [151]. The hybrid Frontal-Delaunay algorithm, which generates quadrilateral meshes, is chosen. It is combined with a mesh smoothing parameter (1) to improve element quality (minimize distortion, improve shape) and reduce numerical errors during simulations. This combination is an effective method for generating

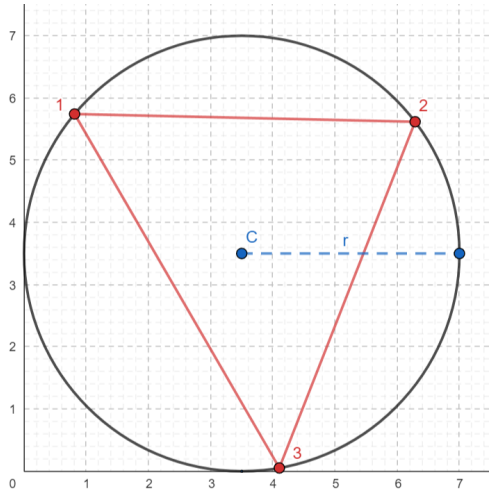


Figure 3.1: Method used to generate random polygons. Vertices are randomly defined on a circle and connected in a clockwise sequence. Here, it yields a 3-sided Polygon. This is repeated for N polygons. All dimensions are in centimeters.

complex geometries, which is suitable for this case. Doing this for the six types of polygons yields Figure 3.3, showing one example of each. The inlet is not shown since it is yet to be cut out based on the injection location; see subsection 3.2.1.

The training dataset is created by generating 200 random cases of each polygon. This returns $6 \times 200 = 1200$ meshes to work with.

3.2. PRE-PROCESSING

During pre-processing, the following steps are taken to set up the OpenFOAM v10 [81] solver:

1. Create an inlet and outlet
2. Define the initial and boundary conditions
3. Define the material properties
4. Tune the control and solver settings

3.2.1. INLET & OUTLET

The mesh files generated previously are exported in gmsh format (ASCII, version 2.2), which is subsequently converted for use with OpenFOAM. As a result, files with points, faces, owner, and neighbor data are created (according to OpenFOAM definition). At this stage, the outer edges of the model are defined as wall boundaries. OpenFOAM requires a fully defined model at each stage. Thus, for now, dummy values are used as the default for the inlet and outlet boundary conditions.

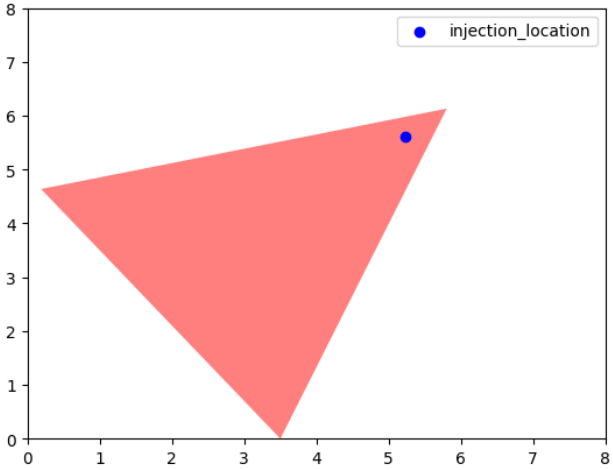


Figure 3.2: Example 3rd degree polygon and a randomly generated injection location. All dimensions are in centimeters.

The inlet and outlet boundary conditions are then properly defined. For each mesh file, the corresponding injection location is extracted from the CSV file (created previously, see section 3.1). All the cells whose center is located inside a trough-the-thickness cylindrical region of radius $r = 2\ mm$ and centered at the injection location are removed through query and boolean operations. The newly created faces are used to define the inlet patch. The result is shown in Figure 3.4.

A similar approach is taken for the outlet. FluidFoam, a Python package for processing OpenFOAM data [153], is used to extract the coordinates of all cell centers. The cell that is the furthest away from the injection location is defined as the outlet cell. Next, the boundary faces are extracted. From this set, the one corresponding to the outlet cell is isolated. At this point, the outlet set only contains one face. To cover the whole width of the geometry, all of the faces in the same column are added that belong to cells that share the same x and y-coordinates. The five nearest boundary cells are extracted to increase the outlet size, yielding a total of 75 faces that form the outlet. This specific final size is the smallest that ensures laminar flow, solver stability, and realistic running times, as reported in subsection 3.3.2. The corresponding faces are added to the set. This final set is used to overwrite the dummy patch for the outlet. Figure 3.4 shows the final result for an example polygon. At this stage, the geometry contains three boundary patches: an inlet, an outlet, and a wall.
Detailed views of the inlet and outlet are shown in Figure 3.19 and Figure 3.20.

3.2.2. BOUNDARY CONDITIONS

The boundary conditions are defined in the "0" directory of the OpenFOAM files. These must be attributed to the three types of patches. Each one has implications on the physi-

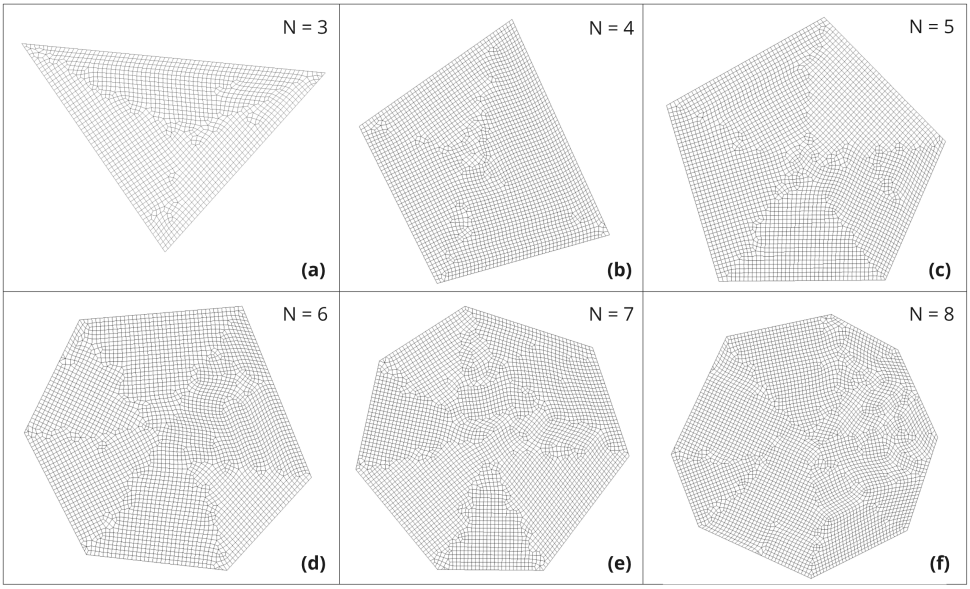


Figure 3.3: The six different types of polygons. Each one is a randomly generated example case for the respective polygon order. The number of nodes ranges from (a) ~2000 to (f) ~4000.

cal conditions for the fluid flow. Details are shown in Table 3.2. The walls exhibit a no-slip boundary condition. Alpha is defined as the volume fraction of the polymer melt in the respective cell(s); see Figure 2.16. It is set to a constant value of 1 at the inlet since this is where the melt flows in at a constant mass flow rate. The outlet may take on velocity and pressure values depending on the adjacent internal field. Here, the melt and air are allowed to exit freely into ambient pressure. The initial conditions set the velocity and pressure values of the internal field to 0 $m \cdot s^{-1}$ and 0 Pa for $t = 0$ s. These are allowed to change from the first time step onwards.

As the zeroGradient boundary condition is frequently used, Figure 3.5 shows the meaning in practice.

3.2.3. MATERIAL PROPERTIES

The multiphase flow involves two components: polymer melt and air. The material properties are listed in table Table 3.3. Although polymer melts usually exhibit non-Newtonian fluid behavior [56], this work uses a constant viscosity model as a first-order approximation. The viscosity is, therefore, independent of the temperature and shear rate. On the one hand, the goal of this work is to demonstrate the potential of GNS as a surrogate for injection modeling simulations. Training the surrogate to replace a first-order model is the logical first proof of concept before introducing more complexity in future research. On the other hand, the state-of-the-art shear-rate-dependent viscosity models developed for polymer melts (as described in subsection 2.2.3) are not natively

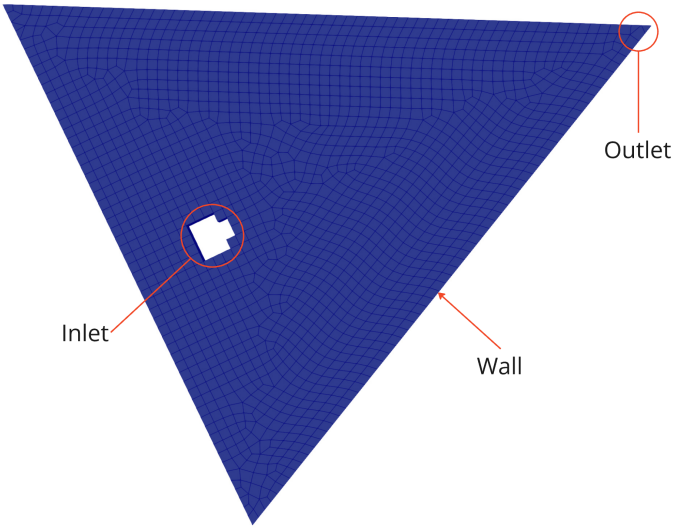


Figure 3.4: Example 3-sided polygon. The inlet is created based on the randomly generated location. The outlet location is chosen the furthest away from the inlet.

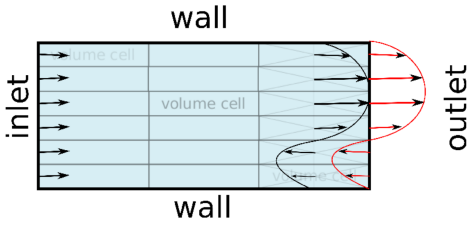


Figure 3.5: zeroGradient boundary condition at outlet patch. The quantity of the patch is extrapolated from the nearest cell value. Reproduced from [34].

implemented in OpenFOAM. Although custom implementations have been reported in the literature, this was not the aim of this work and the use of a first-order approximation guaranteed the generation of sufficient training data.

The model was validated (see subsection 3.3.1). The density and kinematic viscosity values are inspired by reference values of polypropylene (PP) at 160 °*C* [42].
The air is also modeled using a constant viscosity model. The density and viscosity values are based on the work of Baum *et al.* [35]. The surface tension ($\sigma = 0.02$ [N/m] [154]) accounts for the interaction between the two phases.

3.2.4. CONTROL AND SOLVER SETTINGS

The interFoam solver [155] within OpenFOAM is designed for simulating multiphase flow problems. The two-phase algorithm is based on the VOF method. A transport

Table 3.2: Boundary conditions for the walls, the inlet, and the outlet.

Patch	Boundary conditions	Comments
Wall	Alpha contact angle = 52 ° [154]	The contact angle between the fluid and the wall is limited to the gradient.
	Velocity = fixed value (0 0 0)	No-slip boundary condition.
	Pressure = fixed (0)	Constant pressure across the wall. No-penetration BC. Walls do not contribute to any pressure changes.
Inlet	Alpha = 1	-
	Mass flow rate = 0.008 kg/s	-
	Pressure = zeroGradient	Gradient normal to the boundary is zero. It may adjust freely based on conditions in the domain. Not controlled directly, but adjusts based on internal dynamics.
Outlet	Alpha = zeroGradient	No change of alpha across the boundary. The internal field value adjacent to the outlet determines the alpha value at the boundary.
	Velocity = zeroGradient	No change in velocity across the outlet. The flow can exit freely.
	Pressure = fixed value (0)	Fluid exits into a region with constant ambient pressure. It can flow out freely, driven by internal pressures.

equation determines the relative volume fraction of melt and air in each cell, which is denoted by the filling value, alpha (see subsection 2.2.3). Consequently, the interface between the two is not explicitly accounted for but reconstructed from the phase fraction field. Since the phase fraction alpha exhibits a value between 0 and 1, this interface cannot be sharply defined but occupies a volume in the region where the sharp interface could exist.

The momentum transport type is defined as laminar flow. The effect of gravity is neglected. The interFoam solver assumes incompressible, isothermal, and immiscible two-phase flow. The solver settings are defined in the "system" directory. The details are listed in the table Table 3.4. Additional solver settings are reported in Appendix A. The adjustTimeStep setting allows the simulation to adjust the time step size automatically, depending on convergence criteria. It tries to either reach the maximum timestep or the maximum Courant number (see subsection 3.3.2).

The PIMPLE algorithm is a combination of the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) and the PISO (Pressure Implicit with Splitting of Operators) algorithms that offers robustness and stability for solving complex fluid flow problems [75].

Table 3.3: Material properties. Adapted from [35].

Material	**Kinematic viscosity** $\mu \left[\frac{m^2}{s}\right]$	**Density** $\rho \left[\frac{kg}{m^3}\right]$
Polymer melt	1e-03	860
Air	1.831e-05	1.185

Table 3.4: Control and solver settings.

Setting	**Specification**
Solver	interFoam
Simulation time	1 sec
Δt	1e-06
adjustTimeStep	yes
writeInterval	0.01
maxCo, maxAlphaCo	0.5
max Δt	1e-03
timePrecision	6

It is designed to handle large timesteps while maintaining accuracy and can adjust corrector steps dynamically in order to balance computational effort and accuracy. It is used with 100 outer corrector iterations, while in practice, this number is never reached since the solver converges after a few iterations already. Details are listed in Table 3.5. The number of PISO corrector steps per time step is set to three. No additional correctors are used. Under-relaxation factors are defined for the velocity equation and pressure field to help stabilize the pressure-velocity coupling.

3.3. NUMERICAL MODEL

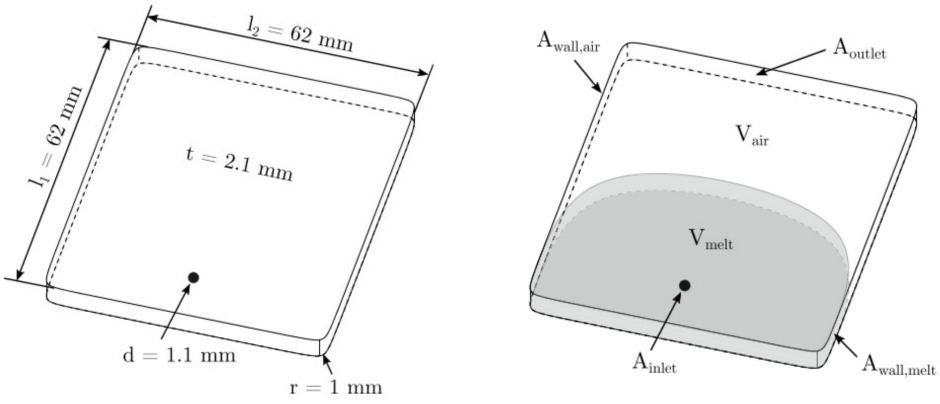
The model is subjected to a mesh convergence study, followed by a quantitative validation of the initial flow assumptions.

3.3.1. QUALITATIVE VALIDATION

A mesh convergence study is needed to ensure realistic material behavior. Polymer melts usually exhibit a complex non-Newtonian behavior (see subsection 3.2.3) not captured by the first-order model used in this work. Furthermore, the mold is initially filled with air, a compressible gas that would require a compressible solver, while interFoam is an incompressible solver. Consequently, the model and material behavior are validated qualitatively, along with a mesh convergence study, with a benchmark case. This benchmark case consists of filling a thin and square mold, based on the experimental and numerical work of Baum *et al.* [35]. The experimental setup is depicted in Figure 3.6.

Table 3.5: PIMPLE algorithm and relaxation factors settings.

Setting	Specification
PIMPLE	
consistent	yes
momentumPredictor	no
nOuterCorrectors	100
nCorrectors	3
nNonOrthogonalCorrectors	0
outerCorrectorResidualControl	
Tolerance (U)	1e-04
Relative tolerance (U)	1
Tolerance (p_rgh)	1e-04
Relative tolerance (p_rgh)	1
relaxationFactors	
equations (U)	0.6
fields (p_rgh)	0.6



(a) Description of the dimensions.

(b) Description of the boundary conditions.

Figure 3.6: Geometry dimensions and boundary conditions. Reproduced from [35].

The following investigations aim to analyze the effect of certain variables on material behavior. In the end, the final model is directly validated with experimental and numerical results [35].

1ST INVESTIGATION

The first investigation analyzes the effect of surface tension, mesh size, and inlet radius on melt flow behavior, as shown in Figure 3.7 and Figure 3.8. The settings and parameters used for this case are listed in Table 3.6. The variables used for each simulation are shown in Table 3.7. For 4a-4c, the mesh size (x,y- plane) was reduced to 0.5 to accommodate the smallest inlet radius option. In both figures, the color denotes the filling of each cell with either phase, where blue represents air and red polymer melt. The screenshots are made from the top view of the plate.

Table 3.6: Settings for the first investigation.

Parameter	Value
Moldwall contact angle theta0	52
BCs (walls)	No-slip
BC (inlet)	Mass flow rate = 0.016 kg/s
Gravity	0 m/s^2
Momentum transport	Laminar
Air	$\mu = 1.831e-05 Pa \cdot s$ $\rho = 1.185 kg/m^3$
Polymer	$\mu = 1e-03 Pa \cdot s$ $\rho = 860 kg/m^3$
Simulation time	0.5 sec
Δt	1e-05 sec
max Δt	1e-03 sec
Tolerance (p_rgh)	1e-06
Tolerance (U)	1e-06
PIMPLE: Outer corrector loops	1
PIMPLE: Inner corrector loops	1
Inlet location (x,y) [mm]	(31, 4)

The following conclusions are drawn:

- The effect of surface tension is minimal. Between simulation ID 1a and 1b the surface tension increased by 0.02 N/m (from 0) but the filling pattern is qualitatively not significantly different. The effect of surface tension is minimal. However, drawing a strong conclusion based on only two values is challenging. Additional values are required to reach a decision.
- A mesh size of 1mm suggests the right trade-off between accuracy and mitigation of numerical artifacts.

Table 3.7: 1st investigation: Variables for each simulation ID, corresponding to Figure 3.7 and Figure 3.8. The values in bold are varied for the corresponding simulations.

Simulation ID	Surface tension [N/m]	Mesh size [mm]	Number of layers in thickness [-]	Inlet radius [mm]
1a	**0**	1	3	1
1b	**0.02**	1	3	1
2a	0.02	**3**	3	1
2b	0.02	**1**	3	1
2c	0.02	**0.5**	3	1
3a	0.02	1	**1**	1
3b	0.02	1	**3**	1
3c	0.02	1	**5**	1
3d	0.02	1	**10**	1
4a	0.02	0.5	3	**0.5**
4b	0.02	0.5	3	**1**
4c	0.02	0.5	3	**3**

- The number of mesh slices significantly impacts the behavior. A slightly filled area visible for 5 and 10 slices shows the behavior towards the center of the thickness of the plate.
- A square inlet radius (0.5mm) affects the initial melt front shape. Little effect for t > 0.05 sec.

2ND INVESTIGATION

A few changes are made concerning the variables and the way that data is visualized:

- A mere two values for the surface tension did not give enough insight, so more are added. Investigate using a wider range of values.
- In the previous investigation, point values are displayed. However, OpenFOAM calculates the raw data at the cell centers. Point values are processed by interpolating the values from neighboring cells. This might give a filtered (unrealistic) representation of the actual behavior. For this reason, cell values are used and displayed from here onwards.
- The top view does not provide insight into what happens within the mold. The aim is to simulate a 2D slice, which should be free of boundary effects. Consequently, the middle slice (at z = d/2) is now extracted.

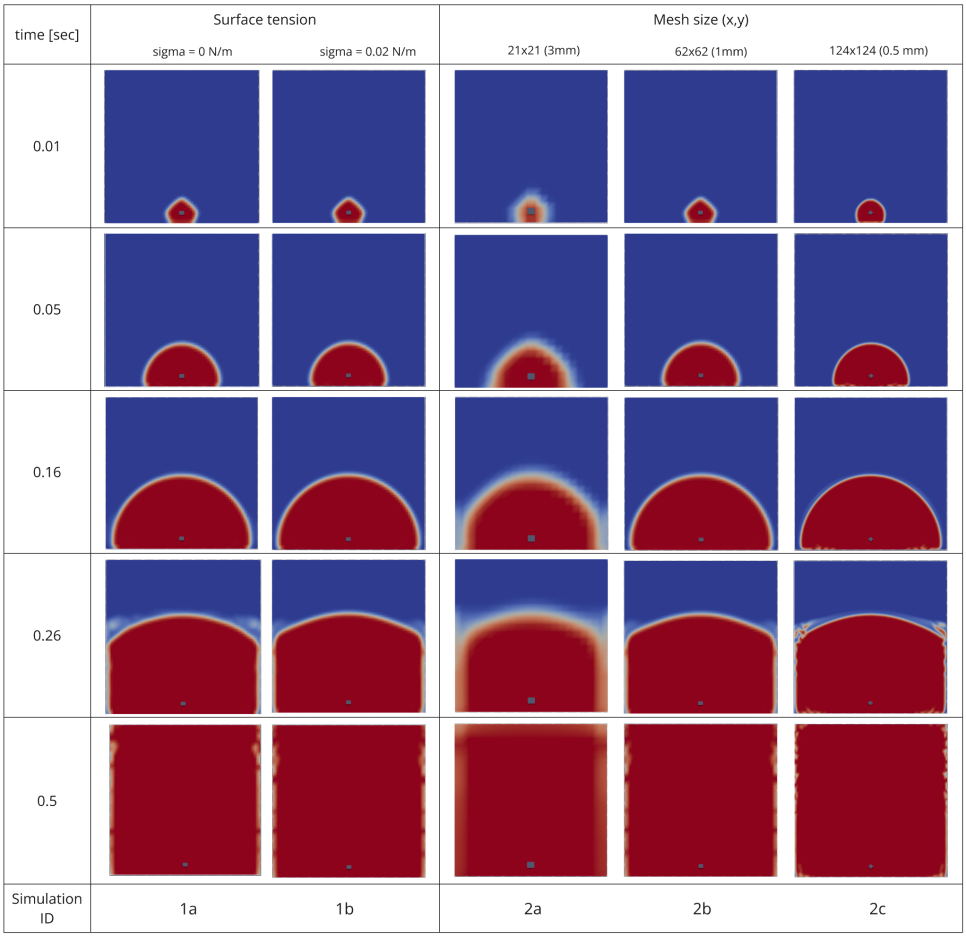


Figure 3.7: The effect of surface tension and mesh size on material behavior. (2a) 221 nodes, (2b) 3844 nodes, (2c) 15376 nodes.

- Using even values for the number of mesh slices in thickness, means that the middle slice lies in between two layers. Thus values are interpolated, leading to the same issue as the point and cell values case. Consequently, odd values are chosen such that the middle slice coincides with cell centers.

The effect of the number of slices and surface tension on the material behavior is investigated. The values are shown in Table 3.8.

Conclusions:

- 1-5 number of slices seem acceptable, while 11-21 contain some artifacts. The conservative convergence criteria for the solver could be the cause of this, which is examined in the next investigation.

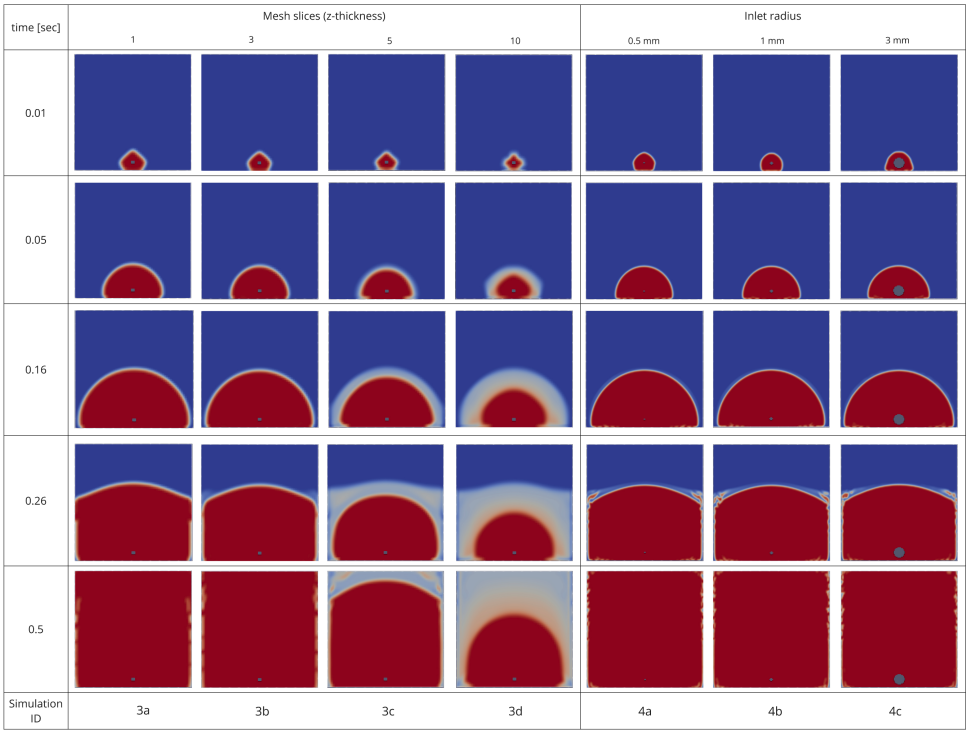


Figure 3.8: The effect of the number of slices in thickness and inlet radius on material behavior.

- The surface tension value has little to no influence on the material behavior. Case 2d shows a slightly different contact angle between the melt boundary and the wall, as well as more prominent bubble formations close to the side walls.

3RD INVESTIGATION

Until now, the convergence criteria for the solver were quite conservative. Allowing only one outer corrector loop (per time step) and one inner corrector loop (per outer loop) ensures fast computation, but does not necessarily ensure convergence. Both values are increased accordingly. Table 3.9 shows the new values with, in addition, tighter tolerances for the pressure and velocity.
Furthermore, the melt front behavior in the cross-section of the plate is investigated to analyze how the flow develops between the front and back plate. From literature [156], it is known that the shape of a fully developed laminar flow between two parallel plates tends to converge to a parabolic shape. With this in mind, five different cases are investigated and shown in Figure 3.10. An educated choice of the number of layers is to be made, to ensure realistic material behavior.

Execution times vary between 1 min (1 layer) and 44 min (20 layers). Although the number of outer and inner loops is set to 100 and 3, respectively, in practice, convergence is

Table 3.8: 2nd investigation: Variables for each simulation ID, for the second investigation, corresponding to Figure 3.9. The values in bold are varied for the corresponding simulations.

Simulation ID	Number of layers in thickness [-]	Surface tension [N/m]
1a	**1**	0.02
1b	**3**	0.02
1c	**5**	0.02
1d	**11**	0.02
1e	**21**	0.02
2a	1	**0**
2b	1	**0.02**
2c	1	**0.1**
2d	1	**1**

Table 3.9: Updated PIMPLE solver convergence criteria.

Parameter	Value
Nr. outer corrector loops per time step	100
Nr. inner corrector loops per outer loop	3
Tolerance (p_rghFinal)	1e-08
Tolerance (U)	1e-08

achieved after ~3 outer iterations and 1-90 inner iterations, respectively.
Conclusions:

- Using the PIMPLE instead of the PISO procedure (allowing the solver to converge) ensures solution convergence. As a consequence, little to no artifacts are observed compared to Investigation 2.
- The parabolic shape is visible for > 5 layers. Comparing 10 to 20 layers does not reveal a significant gain, at an increased computation time. A follow-up investigation is needed, to analyze the contour and include velocity profiles as well as computation time.

4TH INVESTIGATION

In this final investigation, an educated choice is made regarding the mesh size (x,y direction) and the number of layers in thickness. Previously, trends and intuition guided this investigation. Now, it is important to provide more substantial support.

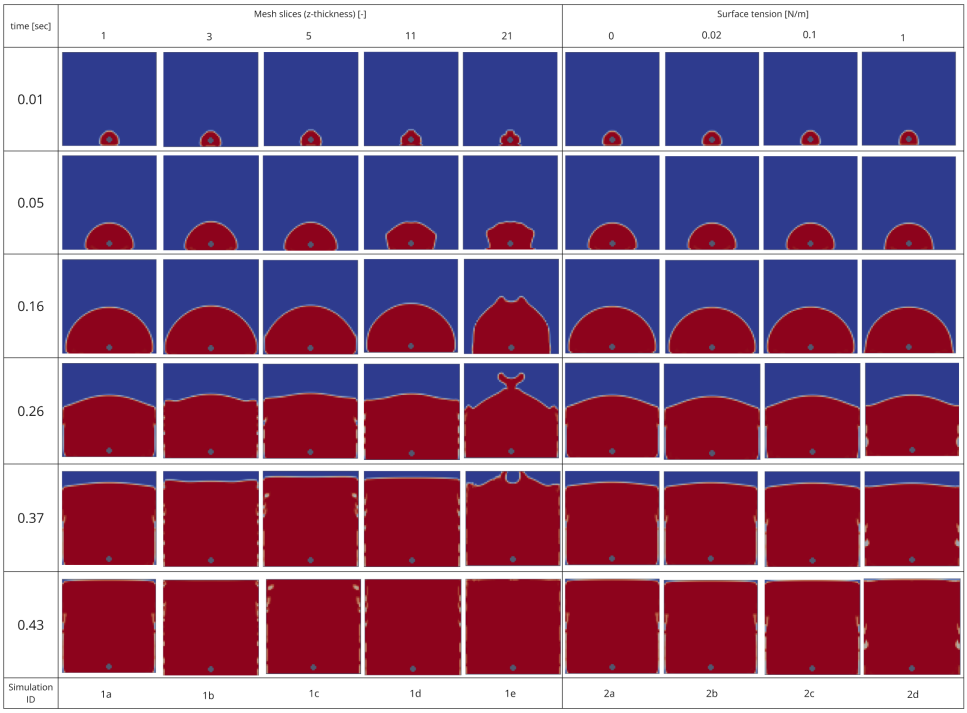


Figure 3.9: The effect of the number of mesh slices in thickness and surface tension on the material behavior.

A classic approach is taken, where two criteria need to be met to ensure the validity of this model:

1. Numerical convergence
2. Comparison to a known (e.g., analytical, experimental, numerical) result

Numerical convergence is explored by analyzing the influence of mesh size on saturation, contour, and velocity profile curves. Table 3.10 shows the variations used for the mesh size in this investigation.

Table 3.10: Mesh size values.

Parameter	Value
Mesh size (x,y-direction)	2 / 1 / 0.5
Nr. layers in thickness (z-direction)	1 / 11 / 15 / 21 / 25

Overall it is expected that with a finer mesh, the curves to numerically converge to the expected shape. However, computation time increases with the number of cells in the

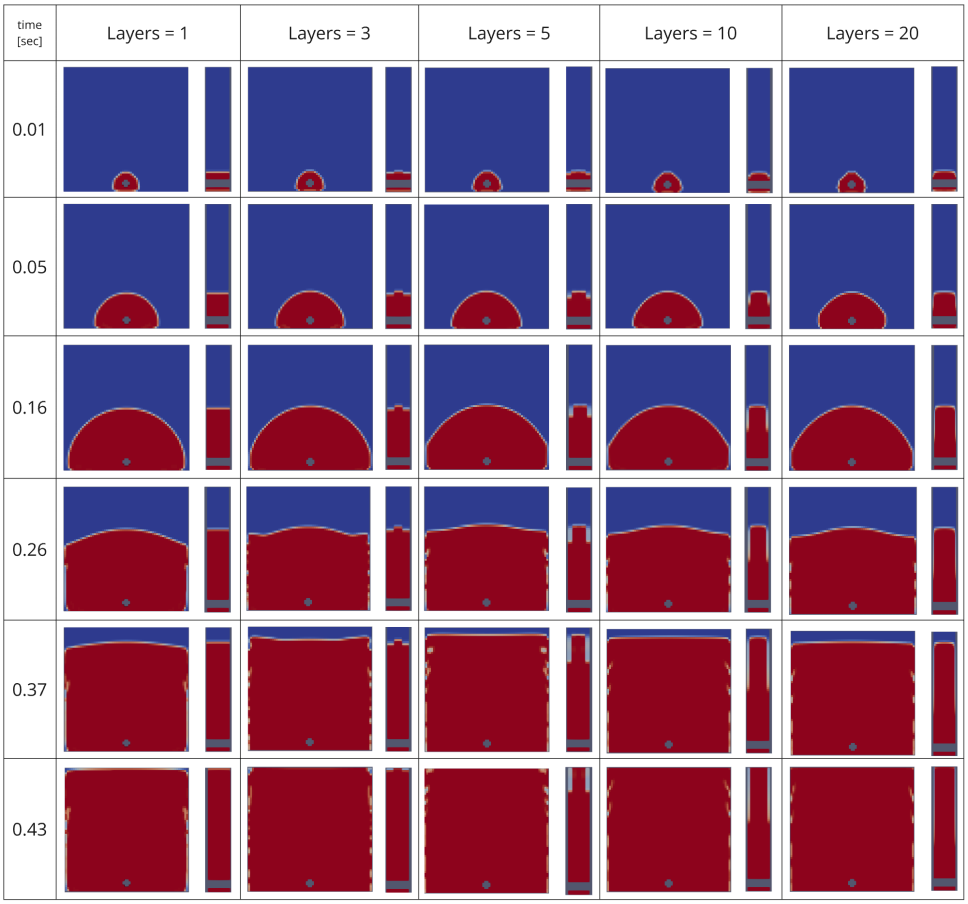


Figure 3.10: The effect of the number of mesh slices in thickness on the melt front behavior. The accompanying strokes show the melt front shape taken from the cross-section in the middle of the plate.

geometry. Therefore, an additional plot is included to examine the computation time versus the number of cells. The objective here is to find a good trade-off between accuracy and computation time.

In Figure 3.11, the shape is not realistic for the single-layer case, ruling these out. The gap between mesh size = 2 and the rest is relatively big. The more layers used, the smoother the parabolic shape.

The contour lines of the cross-section are displayed in Figure 3.12. Again, here it is clear that the single-layer case is not realistic. The roughest mesh size shows some irregular and unexpected shapes. Mesh sizes 1 and 0.5 have very similar shapes, besides the two small regions near the left and right boundaries.

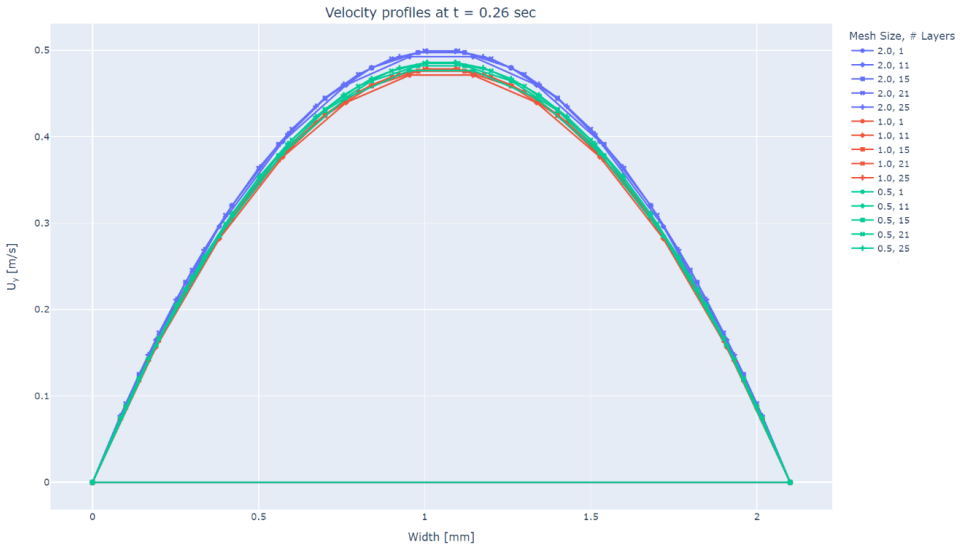


Figure 3.11: The effect of mesh size on the velocity profile at the cross-section in the middle of the plate. For the three 1-layer cases, the lines overlap. To be compared to Figure 3.10.

Figure 3.13 shows how the simulation time is affected by the total number of cells. The roughest mesh size (blue) is the fastest, clocking in at a range of a couple of seconds up to 5 minutes. The simulation time of mesh size = 1 (red) is between 15 minutes and 1 hour. The smallest mesh size (green) takes a minimum of 2.5 hours, up to 12 hours. It seems that mesh size = 1 is a good trade-off, as long as a sufficient number of layers ensures accurate results. In order to do this, the saturation curves are inspected, as shown in Figure 3.14.

In Figure 3.14, the single-layer cases diverge from the rest for t > 0.4s. For mesh size = 2, there is a relatively big gap compared to the other curves for 0.1 < t < 0.4s. The difference between mesh size 0.5 and 1 is much smaller, almost identical. Increasing the number of layers leads to a slight upward shift of the curve and a faster filling time.

Figure 3.15 shows the time it takes to fill up the whole geometry, depending on the number of cells. With a decreasing mesh size, the filling times seem to converge to 0.39 sec. Mesh size = 1 with 15, 21, and 25 layers all sit on this line. As there is no significant gain among these, while computation time increases significantly (see Figure 3.13), the final conclusions are made:

- The single-layer cases are not realistic.
- Using the roughest mesh size (=2) results in irregular contour shapes and does not converge enough.

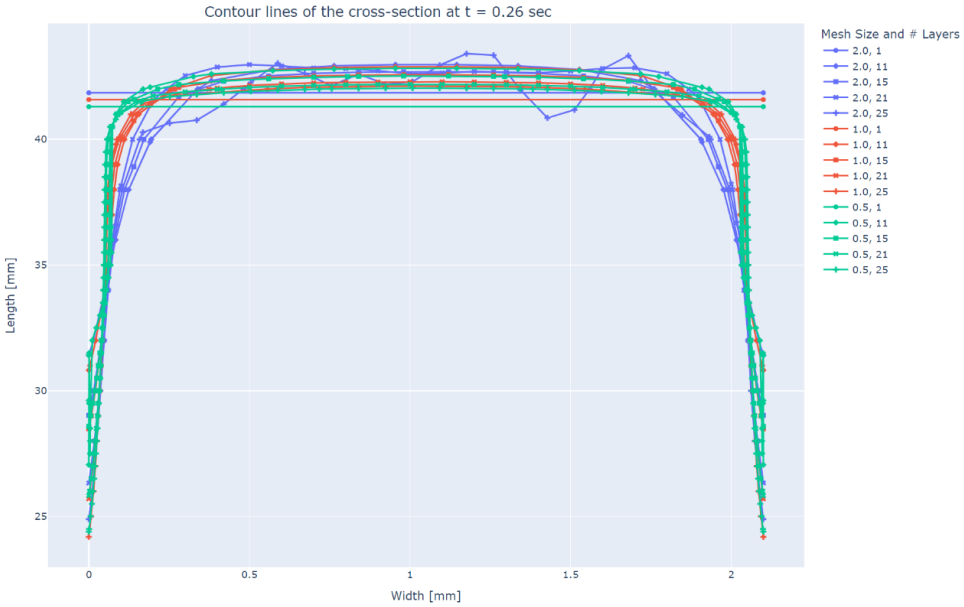


Figure 3.12: The influence of mesh size on the contour lines at the cross-section in the middle of the plate.

- Mesh size = 0.5 is computationally expensive and has no significant gain compared to mesh size = 1
- From the mesh size = 1 cases, 15 layers appears to be the right trade-off between convergence and simulation time. This model will be used from here onwards.

At this point, the first criterion (numerical convergence) has been shown. The second step is to validate the final model with a known result.

Baum *et al.* [35] investigated the filling level and melt front behavior experimentally and compared these to the numerical results. The outcome is shown in Figure 3.16a.

The previously validated model is used to simulate the same scenario. The result is shown in Figure 3.16b. The contour isolines are defined based on a criterion (filling = 0.5), as shown in Figure 3.18. The goal is not to replicate a perfect match, but to have something that qualitatively makes sense. A satisfactory agreement is met between the two.

The filling of the geometry with time, as depicted previously in Figure 3.16, is shown again fully in Figure 3.17.

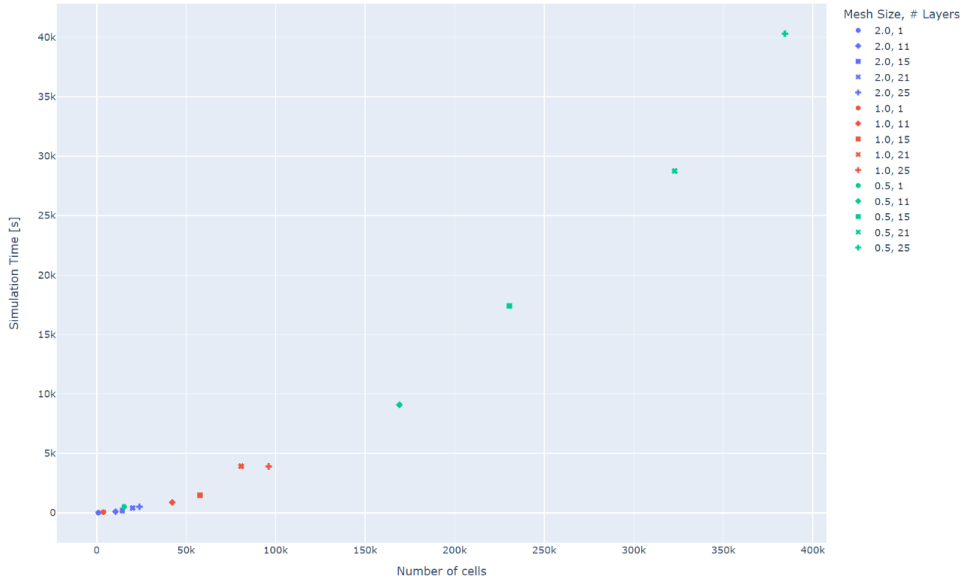


Figure 3.13: The influence of mesh size on the total number of cells and the computation time needed to run one simulation.

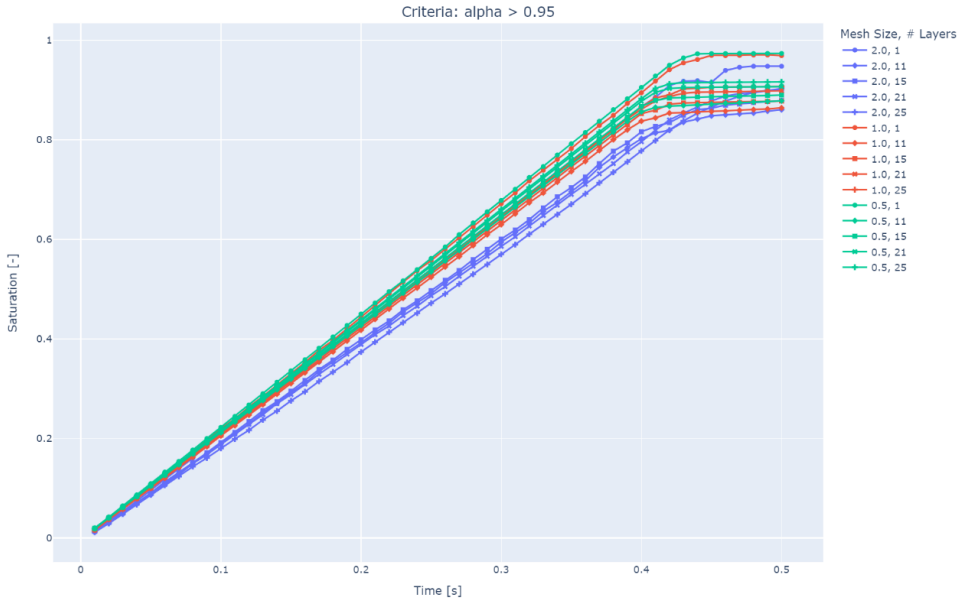


Figure 3.14: The influence of mesh size on the saturation (filling) of the entire geometry.

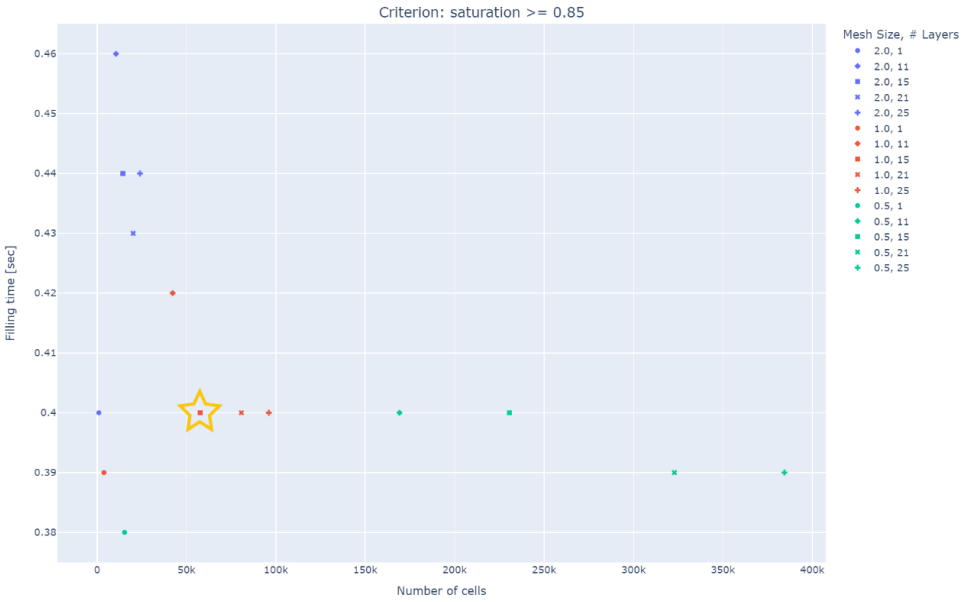


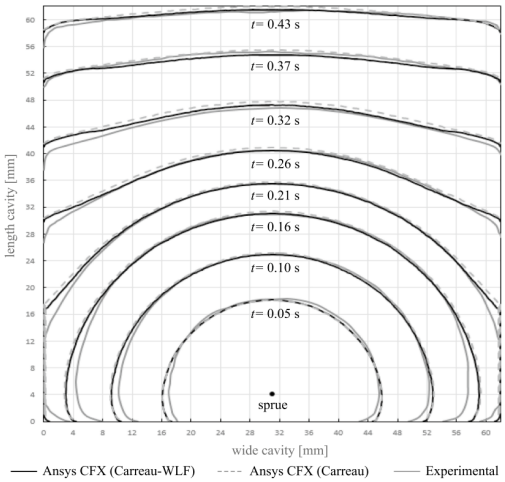
Figure 3.15: The influence of mesh size on the filling time of the entire geometry. The criterion for when the geometry is filled is set at when the mean filling of all cells reaches 0.85. See Figure 3.14. The yellow star denotes the final model choice.

3.3.2. QUANTITATIVE VALIDATIONS

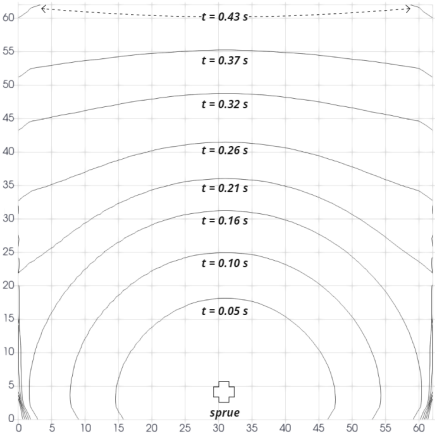
Some analytical validations are made to validate the initial assumptions for this model. The main one is the assumption that the flow is laminar. The inlet and outlet are investigated since, here, the cross-sectional area is the smallest and, according to Bernoulli's principle, the velocity is the highest. Consequently, if the Reynolds number ensures laminar flow in these regions, it should be laminar in the whole geometry. A randomized 3-sided polygon is used for this purpose. The inlet and outlet of interest are shown in Figure 3.4.

Figure 3.19 shows the inlet geometry and the velocity profile. Since the boundary condition is defined as a constant mass flow rate, as expected, the magnitude of the velocity (U) is the same for all faces and constant over the whole simulation. The constant maximum is $U_{max,inlet} = 0.286$ m/s.

Figure 3.20 shows the outlet geometry and the corresponding velocity profile. As expected, the no-slip boundary condition holds at the top and bottom edge (in blue). In between, the velocity profile takes roughly the expected parabolic shape. In the velocity plot, the average and maximum velocity are plotted, where $U_{max,outlet} = 2.9$ m/s. The "alpha.epoxy" value is also plotted, which denotes the filling of the cells with the melt. First, the air seeps out at a constant velocity. Around t = 0.3 sec, the melt arrives and causes the visible change in the velocity profile. The green line denotes the timeframe (t = 0.5sec) captured in the left image.



(a) Reference. Reproduced from [35].



(b) The final model.

Figure 3.16: (a) Comparison of the flow front (midplane) between simulation and experiments at different timesteps. Reproduced from [35]. (b) Flow front (midplane) of the final model at different timesteps.

The size of the outlet was initially smaller, but two issues had to be solved:

- Computation running time was too high. The solver's step size is limited by the Courant number. The Courant number, when set to, e.g., 0.5, means that the flow front cannot progress more than 50% in any cell, in any direction, between two timesteps. This is where the adaptive time stepping plays an important role, which checks at each instance whether the Courant number is exceeded, adapting the time step value on the go. Consequently, the time step decreases to satisfy this limiter (if the fluid travels too fast through the grid, the simulation can become unstable). If the outlet is very small, the velocity locally is very high. Consequently, the maximum Courant number is achieved rapidly, which in turn decreases the allowed time step, increasing the duration of the simulations significantly.

- A higher velocity and smaller surface area means that the Reynolds increases as well, putting the flow conditions locally in the transition or turbulent regime.

To counter both effects, the outlet size (the number of cells) was increased incrementally until the final configuration was achieved (see Figure 3.20, 75 faces total), satisfying both conditions. The simulation time was decreased from 10 hours to 1.5 hours. The initially turbulent flow is now ensured to be laminar, as shown as follows.

The maximum velocities are used to compute the Reynolds number for each case. Equation 3.2 shows the specific equation used,

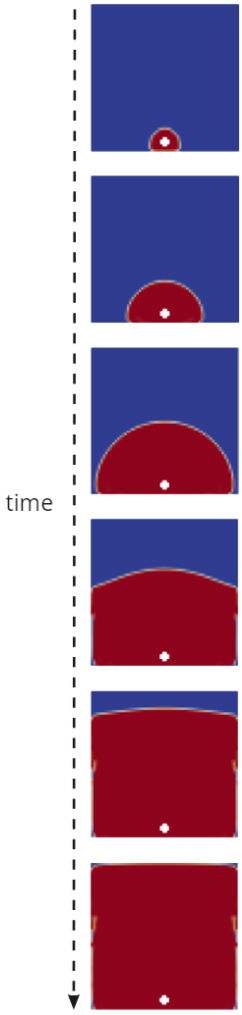


Figure 3.17: Filling of the plate using the final model and settings. Comparable with Figure 3.16b.

$$Re = \frac{\rho \cdot U \cdot D_h}{\mu} \tag{3.2}$$

where ρ is the density, U is the velocity, D_h is the hydraulic diameter, and μ is the dynamic viscosity.

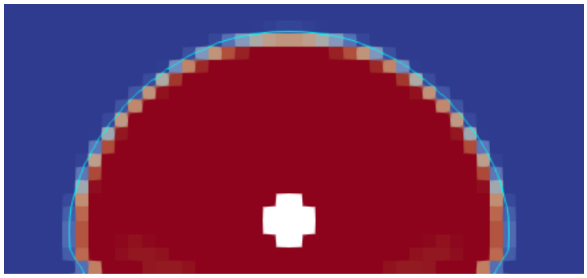


Figure 3.18: Contour isoline (in cyan). Criterion: filling (alpha value) = 0.5.

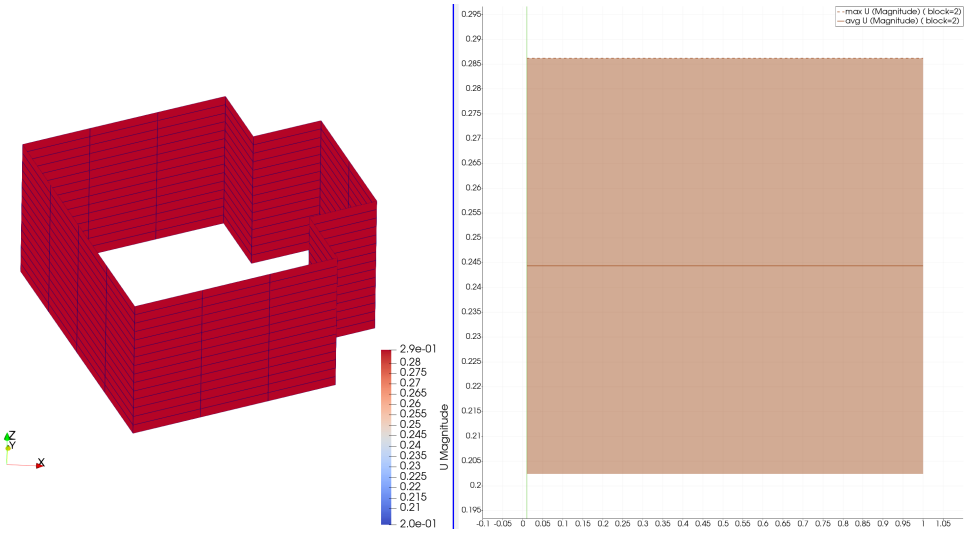


Figure 3.19: Inlet geometry and velocity profile over time. The blue line denotes the division between the two figures. (Left) The timeframe captured by the image is at t = 0 s. (Right) y-axis: U [m/s], x-axis: time [sec].

The hydraulic diameter (D_h) for a rectangular duct is given by

$$D_h = \frac{2 \cdot W \cdot H}{W + H} \tag{3.3}$$

where W is the width, and H is the height. It is calculated for both the inlet and the outlet. The values are listed in Table 3.11.

The Reynolds number is calculated for both cases:

$$Re_{inlet} = 86.09 \tag{3.4}$$

$$Re_{outlet} = 407.4 \tag{3.5}$$

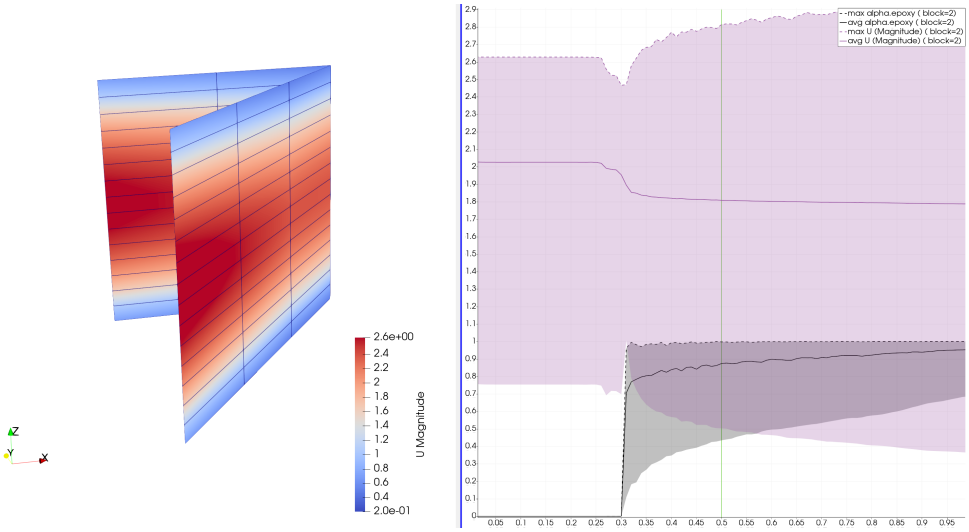


Figure 3.20: Outlet geometry and velocity profile over time. (Right) y-axis: U [m/s], x-axis: time [sec]. The green vertical line (at t = 0.5 s) indicates the timeframe captured by the left image. "alpha.epoxy" denotes the volume fraction of the polymer melt in the outlet cells.

Table 3.11: Values used to calculate the Reynolds for the inlet and the outlet.

Parameter	Value
Density ρ	$860\,kg/m^3$
U_{max_inlet}	$0.286\,m/s$
U_{max_outlet}	$2.9\,m/s$
μ	$1\times10^{-3}\,Pa\cdot s$
H	$2.1\,mm$
W_{inlet}	$15\,mm$
W_{outlet}	$5\,mm$

Re < 2000 holds for both, ensuring the assumption of a laminar flow regime.

3.3.3. RUNNING THE SOLVER

The ground truth dataset is now generated using the final model and variations of geometry and inlet & outlet locations. OpenFOAM performs CPU-based computations. The workstation provided by Baris from the Aerospace Structures and Materials Department at the TU Delft Aerospace faculty has 2 Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz with 2 × 24 = 48 total threads. A script is written that automatically runs the simulations

using parallelization with 15 threads at a time. The duration of the simulations lies in the range of 1.5 hrs - 4 hrs for 3^{rd} order up to 8^{th} order polygons, respectively. The whole process took about 220 hours ≈ 9 days. This returns 1200 simulations, that are now ready for post-processing.

3.4. POST-PROCESSING

The main goal of post-processing is to extract the needed information from the simulations and format it adequately for the graph construction and training of the network in chapter 4. The data is stored for every simulation and saved to a single JSON file. Other file formats (such as HDF5 and CSV) were tested and compared regarding data structure and compatibility. Although HDF5 is efficient due to its binary format, it does not handle well inhomogeneous data types and Python objects such as dictionaries. The CSV format presents similar issues. JSON instead offers wide compatibility and is easy to read and write. The content of each is shown in Table 3.12.

The first step is to extract the cell center coordinates (where cell values are computed) of each geometry. The fluidfoam [153] python package is used to read the mesh. The values of the middle slice are extracted, so the cells intersecting the middle plane are selected, and their x and y-coordinates are stored as nodal coordinates[3].

The next entry is the node types. Each geometry has four different types of nodes: inner -, wall -, inlet -, and outlet nodes. These are labeled 0,1,2,3, respectively. The values are stored in an array. Both position and node types are shown in Figure 3.21a.

The physical properties are now extracted: alpha (filling value), velocity, and pressure. The PyVista python package [157] is used for this purpose. For every timestep, the values are extracted from the nodes. To do this, the geometry is sliced through the middle plane and all node values at each timestep are stored. At the end, they are all stacked together in an array.

A filter is applied to the alpha values, as shown by Equation 3.6. This is done to mitigate the formation of air bubbles at the wall boundaries. They were deemed as numerical artifacts and it was found to be detrimental to the training of the network.

$$\alpha = \begin{cases} 1, & \text{if } \alpha > 0.1 \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

Additionally, since geometry size (the surface area) varies (as stated in section 3.1), smaller geometries (3^{rd} degree polygons) saturate faster than bigger ones (8^{th} degree polygons). The filling of the whole geometry takes about 0.3 sec for the smallest case, whereas the bigger ones take about 0.8 sec. The total simulation time of all simulations, however, is set to a constant value of 1 sec. The timesteps after complete filling will not add impor-

[3] The term "node" is conventionally used to denote a point of interest in a graph environment. Therefore, from here onwards, cell centers will be referred to as nodes (while in reality, they are cell centers and should not be mistaken for nodes as in a finite element mesh).

tant information to the network, so these are filtered out by defining a cut-off value of $\alpha_{\text{mean}} = 0.95$. Reaching this value marks the end of each respective simulation.

Lastly, the connectivity is defined. Inspired by Pfaff *et al.* [33], the mesh connectivity is used to establish the graph connectivity. The OpenFOAM Fluidfoam package [153] does not provide a straightforward method to extract cells and their neighbors, so the workaround goes as follows. OpenFOAM cells and faces are based on owners and neighbors [158]. These are extracted, yielding the owner and neighbor faces and corresponding cells. Next, these two lists are filtered to take out all cells and faces that are not part of the middle plane. The lists are combined into one list where each cell is combined with all its faces. With this in place, the connection of the inner domain is established.

Table 3.12: Mesh data saved for each simulation. `pos` contains the coordinates of all nodes, `velocity` contains the velocity components at each timestep, and `cells` contains each node number and its neighbors.

Mesh data	Size
`pos`	(nnodes, 2)
`node_type`	(nnodes, 1)
`alpha`	(ntimestep, nnodes, 1)
`pressure`	(ntimestep, nnodes, 1)
`velocity`	(ntimestep, nnodes, 2)
`cells`	(nnodes, 5)

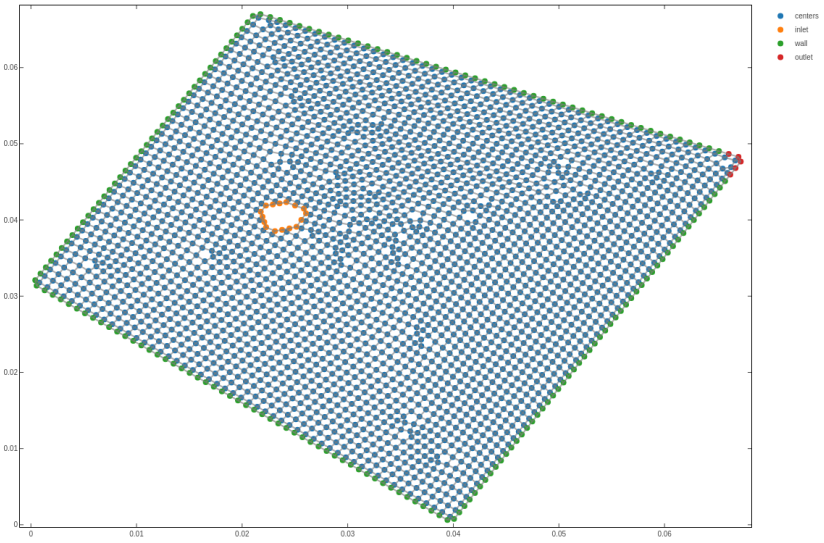
The boundary cells need to be added as well. The face numbers corresponding to the boundaries wall, inlet, and outlet are extracted. These are matched to the respective neighboring inner cell. This step adds the connectivity among inner cells, and the outer layer of inner cells with the nearest boundary face. The only thing missing is the connectivity between the boundary cells themselves. Pyvista [157] is used to extract the boundary node (the actual nodes, not cell centers) coordinates. For each node, the nearest two cell centers are selected, defining them as neighbors of each other. Now the complete mesh connectivity can be established and saved in a single list, where each cell is paired with its neighboring cells.

Table 3.13: Dataset parameters and their values.

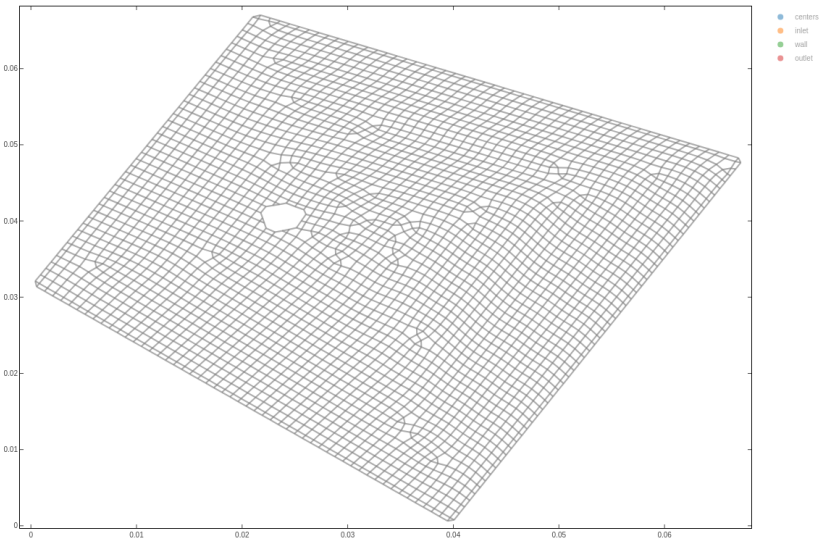
Dataset parameters	Value
Train / valid / test	1000 / 100 / 100
Time step dt	0.01 sec
Simulation length	30 - 90 steps
Number of nodes	2000 - 4000

The information is stored in one dictionary. Table 3.12 shows how this information is stored for each simulation and the corresponding array sizes. This is done for all 1200 simulations. The train, validation, and testing datasets are created: 1000 random (of the 1200) simulations are chosen, and the respective data is saved in a `train.json` file. The remaining 200 dictionaries are randomly divided into 100 (to create the `valid.json` file) and 100 (for the `test.json` file). The details of the dataset are listed in Table 3.13.

The train file contains 13.4 GB of data, and the valid & test are both 1.3 GB. The dataset is now ready to be used for training of the Graph Neural Network.



(a) Nodes and their node type.



(b) Mesh connectivity.

Figure 3.21: The mesh coordinates, node types, and connectivity of a randomly generated 4^{th} order polygon.

4

MeshGraphNet

The reader is referred to the original work of Pfaff et al. *[33] for an extensive description of the MeshGraphNet model.*

This chapter provides a detailed report of the steps required to encode the data as a graph, train the graph neural network surrogate, test it, and generalize it. The Python scripts implemented to achieve these steps are provided as a git repository [1]. The computational pipeline this code implements can be consulted. It features a detailed overview of the code logic, specifying each component's inputs and outputs and how their interactions[2].

4.1. Graph Encoding

This section details how the simulation data used to train the graph neural network is encoded into a graph. The node coordinates, the alpha values, and cell connectivity are extracted as shown previously in Table 3.12. The simulated duration ranges roughly between 0.3 and 0.8 seconds. This data is formatted and stored as a graph, complying with the PyTorch Geometric [159] format.

Node features are stacked as contiguous torch tensors. The `node_type` is stored as one-hot vectors. A one-hot vector is a binary vector used to represent categorical data, where only one element is set to 1 (indicating the presence of that node type), and all other elements are set to 0.

A `Data` object from the PyTorch Geometric [160] library is created and initialized with the node features, connectivity, target alpha values, and nodal position coordinates. The imported graph connectivity is converted to a sparse Coordinate (COO) format, where the elements are stored as tuples of element indices and the corresponding values, with

[1] This repository is the same as the one indicated in chapter 3 and is currently held private. Access may be granted upon request.
[2] See Appendix C.

Table 4.1: Node features for the graph encoding.

Node feature	Value	One-hot vector
`node type`	0 = inner	[1, 0, 0, 0]
	1 = wall	[0, 1, 0, 0]
	2 = inlet	[0, 0, 1, 0]
	3 = outlet	[0, 0, 0, 1]
`alpha`	Values of the cells, corresponding to the current time step	-

shape `[2, num_edges]`. Each column is a pair of node indices indicating an edge. To achieve spatial equivariance, a transformer object is applied to the graph data, which is composed of two transformations: *(i)* The Cartesian coordinate differences are added as edge attributes, *(ii)* the Euclidian distances between nodes are computed and added as additional edge attributes. This graph data can now be used for the actual training procedure.

4.2. Model & training

The `train.json` file containing the dictionaries with the data from each simulation (as described in section 3.4) is loaded in batches by a data loader. After shuffling the dataset, a model and a time step t are randomly drawn. The goal of the simulator is to learn a forward model of the dynamic quantities of the mesh at time $t+1$, based on the node features and connectivity at the time step t. Consequently, time is not explicitly defined, but implicitly accounted for. Including time as an explicit node feature and predicting the model state for random time step increments is discussed in chapter 6.

Generally, normalization is common practice, since parameters may differ by orders of magnitude, which also applies to the associated weights, making learning difficult. No normalization is used in this case since alpha values are already between 0 and 1. Table 4.2 lists some important hyperparameters and settings for the training procedure.

The diagram of MeshGraphNets is shown in Figure 4.1. The encoder constructs the latent graph from the input state. The system's state is described by $M^t = (V, E^M)$, where mesh nodes become graph nodes V, and mesh edges become bidirectional mesh edges E^M in the graph. It assigns features to nodes (alpha and node type, as shown in Table 4.1) and edges (set to 3: two relative displacements between each pair of nodes, and one norm (distance between two nodes)). The latter aims to achieve spatial equivariance, as explained in section 4.1. This information is required for the encoder to determine the initial size of the edge input for the MLP.

The processor consists of N message-passing blocks, where nodes send and receive messages to and from neighbors. All latent dimensions are size 128. The neighbors of each node are determined by the mesh connectivity. They are applied in sequence: *(i)* the residuals (original node and edge features) are saved, *(ii)* data is aggregated to construct messages, *(iii)* the respective edge and node embeddings are updated, where

Table 4.2: Hyperparameters and settings for meshnet. Most of these values are inspired by the work of Pfaff *et al.* [33].

Hyperparameters	Value
Batch size	10
Learning optimizer	Adam (exponential decay)
Learning rate	1e-04 to 1e-06
Number of message passing steps	15
NN architecture	ReLu-activated 2 hidden layers 128 layer neurons
Predicted value	alpha (filling) value
Loss function	RMSE[3]

the changes are added to the residuals. This is repeated for N message-passing steps. Information is aggregated locally to provide a global prediction later. The number of message-passing steps defines how far the information encoded at a given node influences the results at another node. For instance, if the message-passing steps update the node value by averaging the value of the first-neighbor nodes, then the information travels by one edge at each message-passing step.

After the final processing step, the decoder turns the newly updated states back into the physical realm. The simulation dimension informs the decoder about the dimension it should end up with. In this case, an MLP is used to transform the latent node features (size 128) into a single (size 1) node output feature, as alpha is a scalar value[4].

In conclusion, forward propagation is used to get the final representation of each node. Then, the loss is calculated via the loss function. A mask is used to filter out boundary nodes, so the loss is computed exclusively for the inner nodes. Backpropagation is performed to compute gradients and adjust weights. The model's parameters are updated using the optimizer. The learning rate is adjusted depending on the step count. This procedure is repeated for each training step. Training is performed using CUDA (version 12.3), an NVIDIA toolkit for high-performance GPU-accelerated applications [161], on a Quadro RTX 6000.

[3] Alternative loss functions may also be considered. See the loss investigation in Appendix D.
[4] To visualize the whole process, see Appendix C.

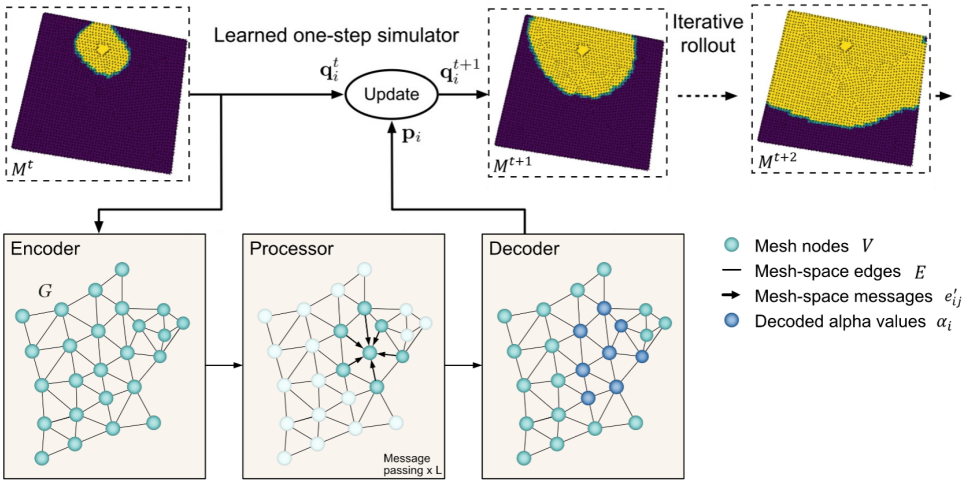


Figure 4.1: Diagram of MeshGraphNets operating on the mold filling domain. The model uses an Encode-Process-Decode architecture trained with one-step supervision, and can be applied iteratively to generate long trajectories at inference time. The encoder transforms the input mesh M^t into a graph. The processor performs several rounds of message passing along edges, updating all node and edge embeddings. The decoder extracts the alpha value for each node, which is used to update the mesh to produce M^{t+1}. Adapted from [33].

4.3. Rollout procedure

The rollout predicts a model state based on a previous state at a given time step, optionally rendering the results for visualization.

4.3.1. Prediction

During training, the model and its training state (including the optimizer state and the current step) are periodically saved after every epoch (one epoch means that all the training data have been seen by the network, i.e., 5577 training steps in this case). This allows the user to interrupt and resume training from a specific training state, and more importantly, evaluate the model after a specific number of Epochs. For a rollout prediction, the simulator and train state files are loaded at a given epoch number, and a prediction is made for one whole simulation (30-90 timesteps). The data loader loads a single simulation from the respective dataset in this case. The prediction takes the loss value of every single timestep prediction into account. The output dictionary contains the initial alpha values, the ground truth alpha values, the predicted alpha values, and the loss value. The node type and coordinates are added for rendering purposes. This data is stored in a pickle file in write-binary mode.

4.3.2. RENDERING

To visualize the prediction versus ground truth simulations, the data from the prediction file can be used to render a video showing the results. After loading the file, the initial alpha values are concatenated with the ground truth and predicted data to form continuous sequences for animation. Next, the `Triangulation` class of matplotlib [162] is used to create a triangular mesh from node coordinates. Each frame is rendered by iterating over the data to plot them as triangular meshes with color coding based on their values. The animation is rendered in MP4 format.

5

PREDICTION RESULTS

5.1. TRAINING & VALIDATION

The training loss values represent individual loss values recorded after each 10 training steps. A batch size of 10 is used. The training loss rolling mean is computed based on the previous and the next 50 values of each training value point. After each epoch (5577 training steps), the learned simulator is validated using the validation dataset. The respective loss values are computed and represented by the validation loss mean value and the validation standard deviation at each epoch. The training and validation curves are shown in Figure 5.1.

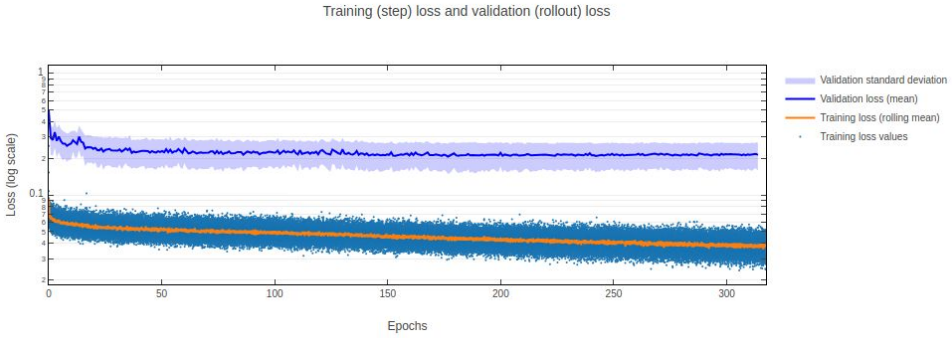


Figure 5.1: Train and validation loss curves.

The training graph decreases consistently with an increasing number of epochs. The validation also decreases for the first part of the graph, stabilizing from 150 epochs onwards. Hereafter it is clear that training does not add any substantial gain. For this reason, the learned simulator at 150 epochs is taken and used to generate validation and testing rollouts. Three cases are shown: *(i)* worst, *(ii)* medium, and *(iii)* best predictions paired with

their loss values. This is done using both training and validation data.

The learned simulator is loaded from its state at the 150 Epoch mark. For both train and validation rollouts, 20 samples were taken from the respective dataset. The best, the worst, and an average prediction are extracted, followed by rendering, and shown below.

Table 5.1: Loss values and elapsed time for training and validation rollouts.

Sample	Loss (RMSE)	RMSE Std Dev	Running time [sec]
Train			
best	0.096	0.053	0.52
average	0.19	0.053	0.5
worst	0.3	0.053	0.43
Validation			
best	0.11	0.059	0.9
average	0.21	0.059	0.89
worst	0.33	0.059	0.83

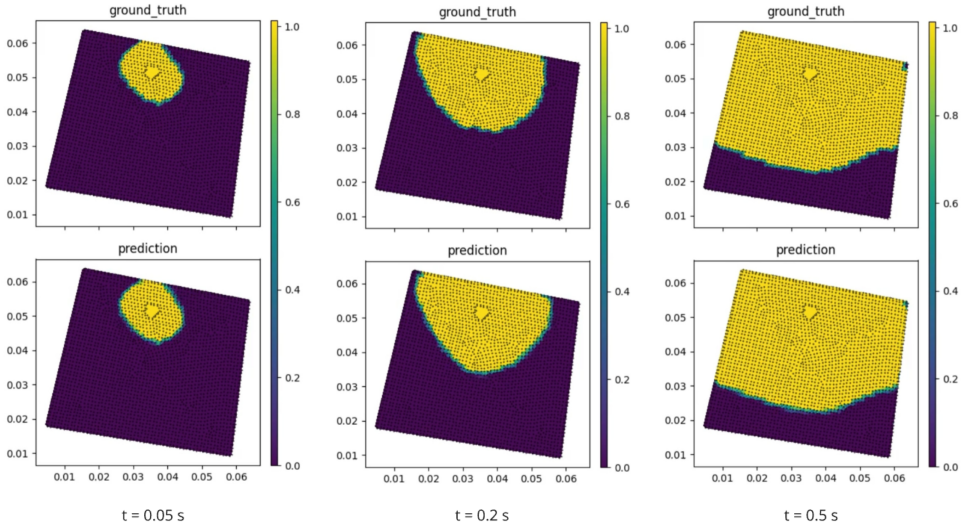


Figure 5.2: Training rollout: best case.

The following figures show the rollouts using the validation dataset:

Both training and validation rollouts' best cases show a good agreement between the ground truth and prediction. The average and worst cases show that the increased er-

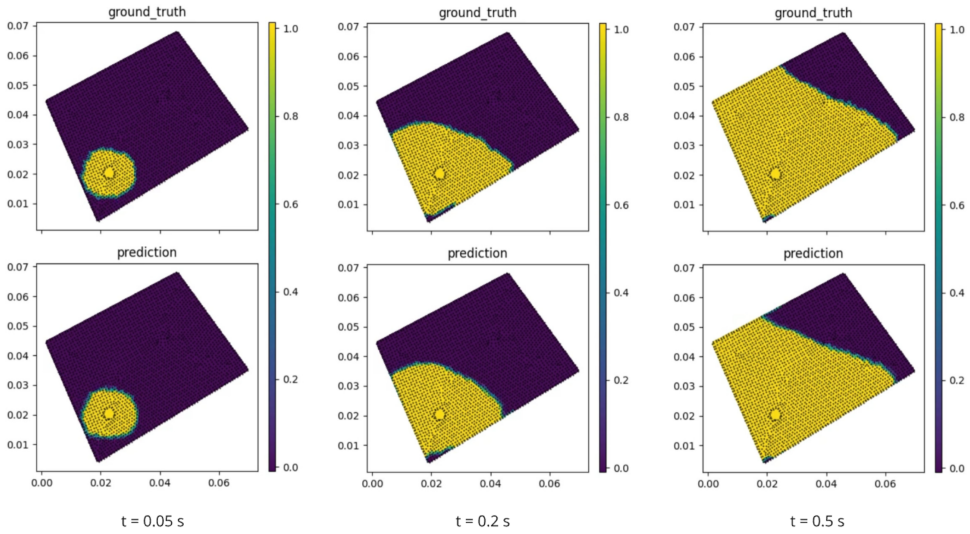


Figure 5.3: Training rollout: average case.

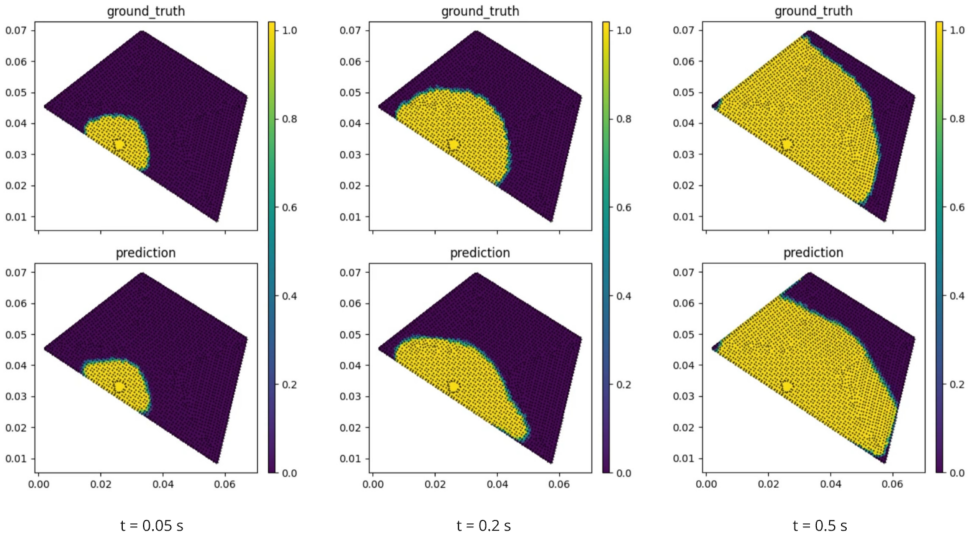


Figure 5.4: Training rollout: worst case.

ror originates from the difference in melt front shape (e.g., Figure 5.4), the prediction preceding the ground truth (e.g., Figure 5.7), or a combination of the two.

The simulation time versus the mesh size (number of cells) is shown in Figure 5.8. The ground truth running times are taken as the average of a few examples of the respective polygon order.

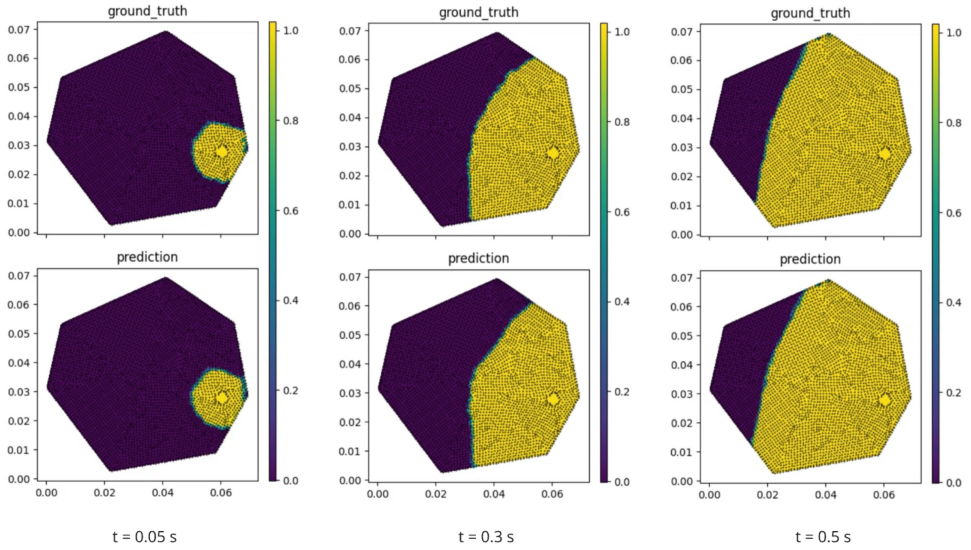


Figure 5.5: Validation rollout: best case.

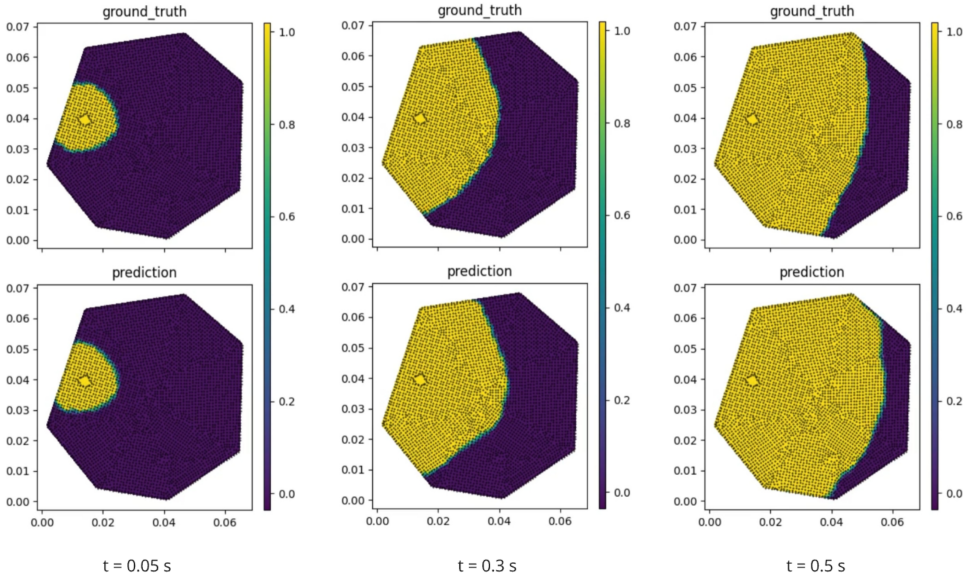


Figure 5.6: Validation rollout: average case.

It shows the notable difference in the running time of the ground truth and prediction. The simulator predicts the train and validation simulations in less than a second, while the ground truth takes several hours to run.

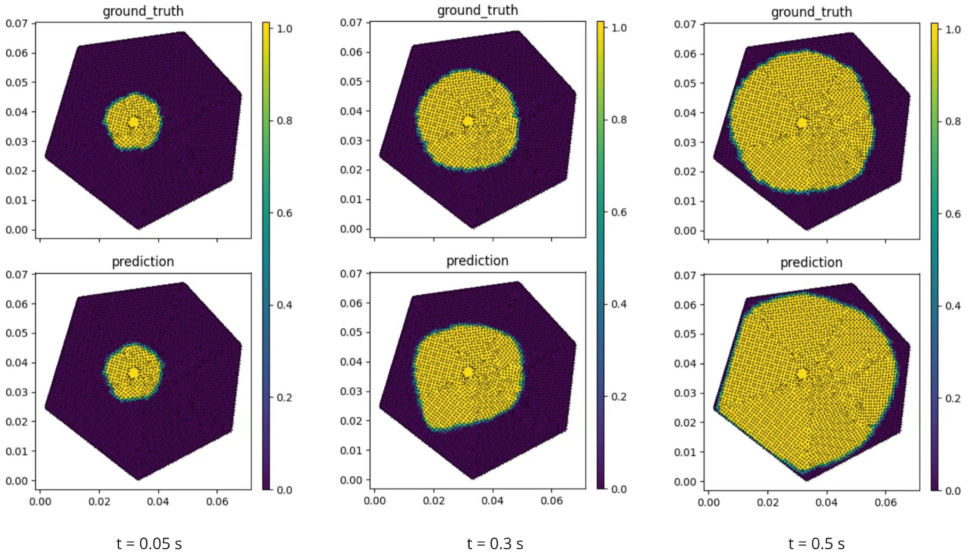


Figure 5.7: Validation rollout: worst case.

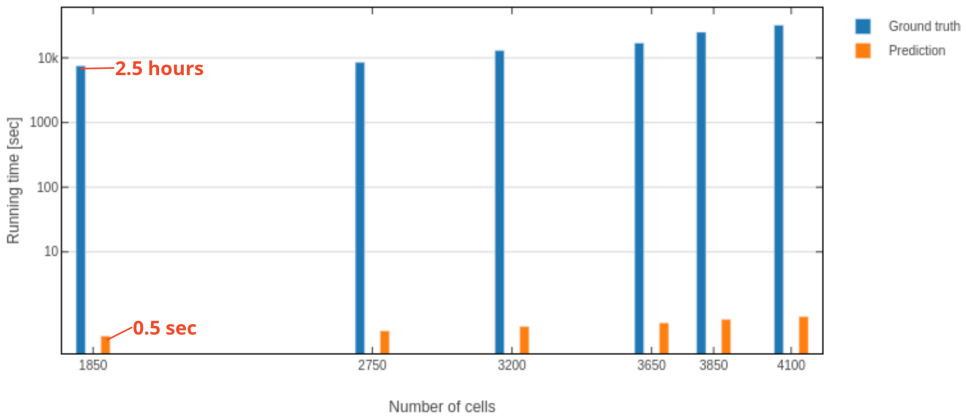


Figure 5.8: Running time comparison between the ground truth solver and GNN prediction.

5.2. GENERALIZATION

The model is now tested for its generalization potential. A new scenario is predicted: the filling of a U-shape. The inlet and outlet are chosen conveniently: the inlet in the upper left corner and the outlet in the upper right corner. Their sizes match those seen during training and validation.

It is clear that although the physical behavior is somewhat captured by the prediction, the melt front progresses slower than the ground truth. Consequently, the prediction is

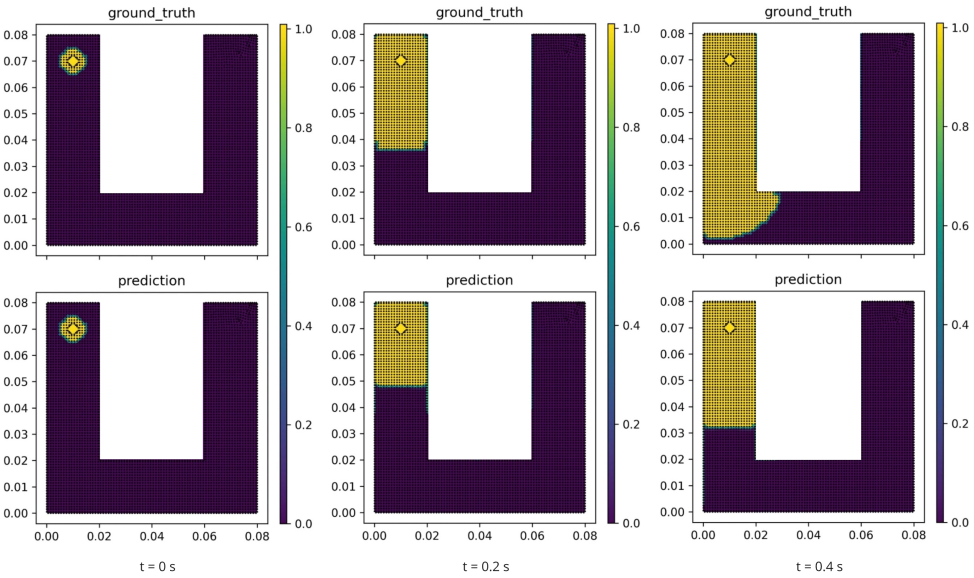


Figure 5.9: Filling of the U-shape: ground truth and prediction. Frames 1-3.

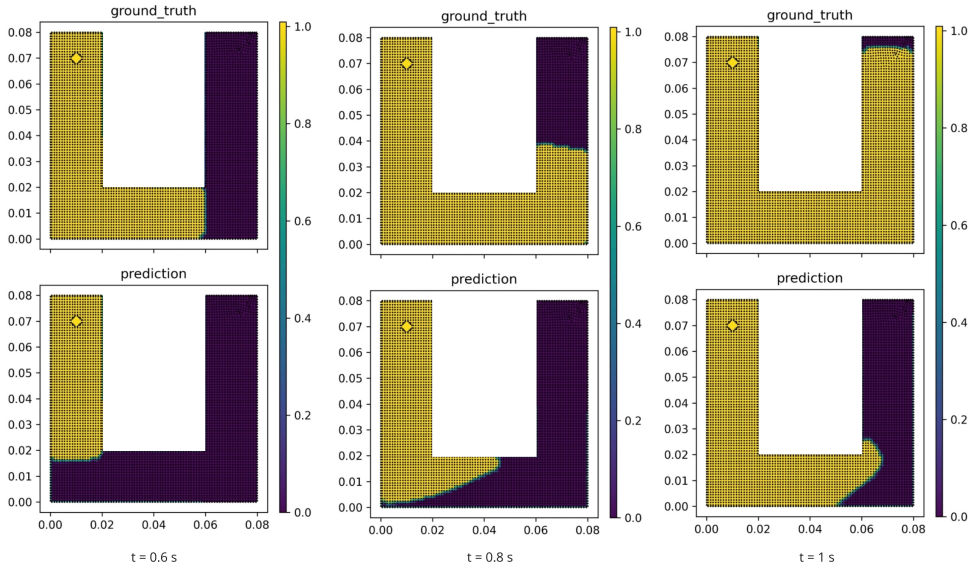


Figure 5.10: Filling of the U-shape: ground truth and prediction. Frames 4-6.

allowed to run for longer (about 2x as long) until the mold is filled. The melt front behavior is captured fairly accurately in the beginning but worsens as the simulation goes on. This may be caused by error accumulation, as discussed in chapter 6. Additionally,

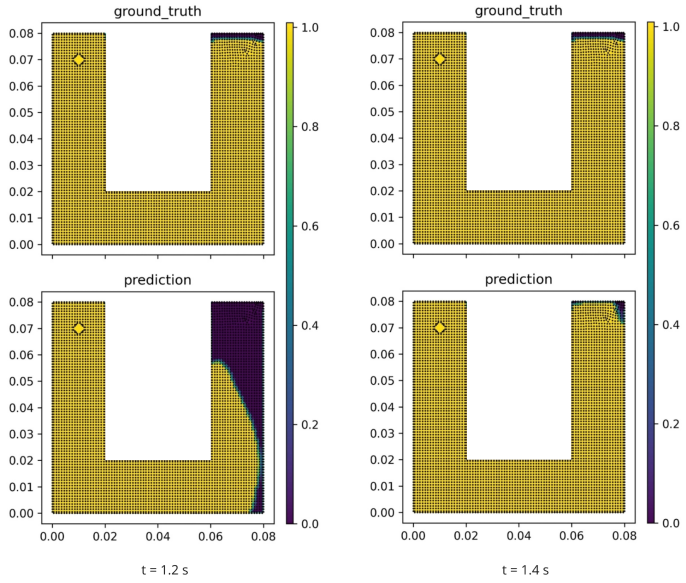


Figure 5.11: Filling of the U-shape: ground truth and prediction. Frames 7-8.

the prediction seems to accelerate the melt front speed when encountering a corner, specifically the inside of both corners in the U-case.

The next example case shows this cornering behavior by introducing a thinner cross-section overall and more corners compared to the previous case.

Here again, the cornering behavior is clearly visible. The prediction is also slower than the ground truth. The ground truth filling process takes 1 second, while the prediction aligns after twice the amount of time.

In the following third test rollout case, the prediction runs for about 3x-4x as long as seen during training and deals with a higher number of nodes. The geometry contains a simultaneously diverging and converging channel, followed by two asymmetric sections, two corners, and a larger square. The inlet is located in the lower left corner, as shown in Figure 5.16, and the outlet is situated in the upper right corner. The geometry is ~4x bigger, and simulation times ~3x longer than seen during training. The aim here is to reexamine the cornering behavior of the melt when facing corners, as well as to analyze the response of the prediction when encountering converging and diverging channel sections.

In Figure 5.16 at $t = 0.5$ sec, the prediction shows the melt front accelerating near the boundaries due to the converging section. This response is similar to the cornering behavior observed previously, where, in this case, it also does it in a symmetric fashion. Figure 5.17 shows similar results. Figure 5.18 illustrates the response to cornering. In some cases, the prediction tends to re-adjust after deviating from the ground truth during these specific sections. An example is shown by comparing the prediction at $t = 1.75$

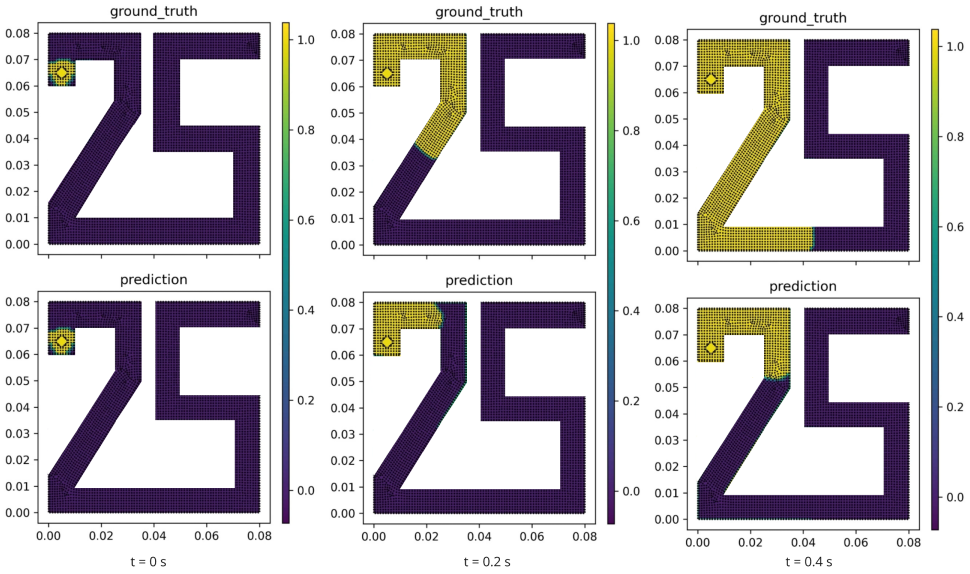


Figure 5.12: Filling of the 25-shape: ground truth and prediction. Frames 1-3.

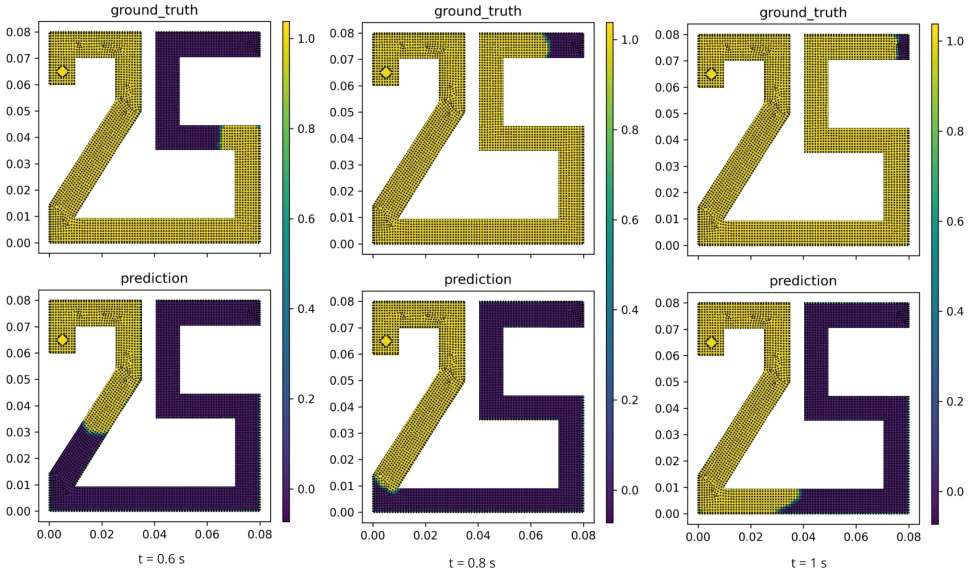


Figure 5.13: Filling of the 25-shape: ground truth and prediction. Frames 4-6.

sec in Figure 5.18 and the ground truth at $t = 1.25$ sec in Figure 5.17, where the filling pattern is relatively similar.

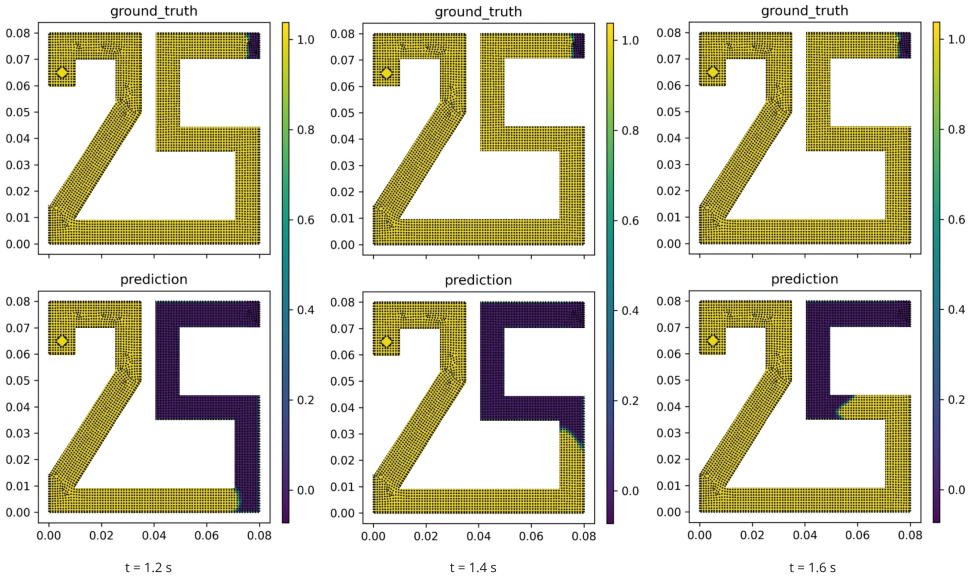


Figure 5.14: Filling of the 25-shape: ground truth and prediction. Frames 7-9.

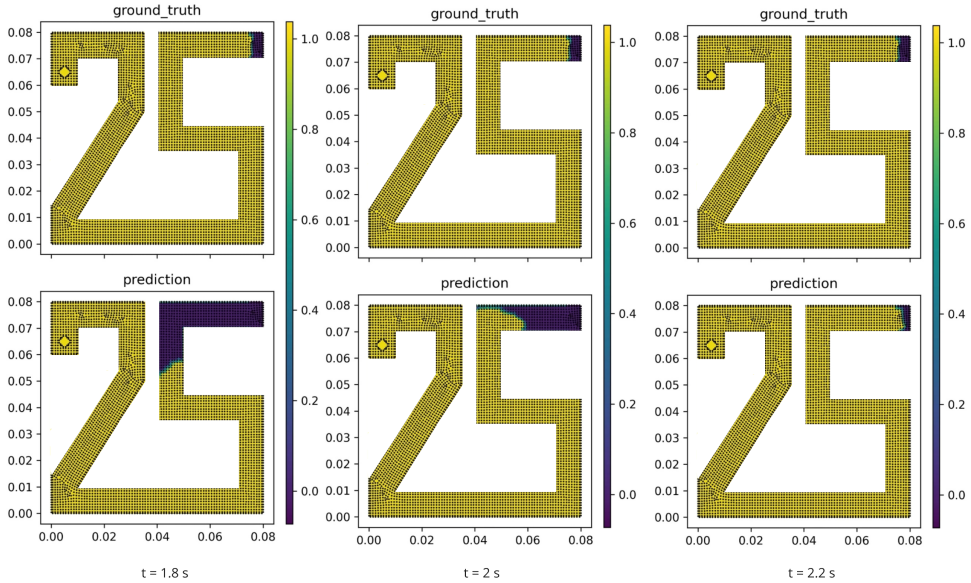


Figure 5.15: Filling of the 25-shape: ground truth and prediction. Frames 10-12.

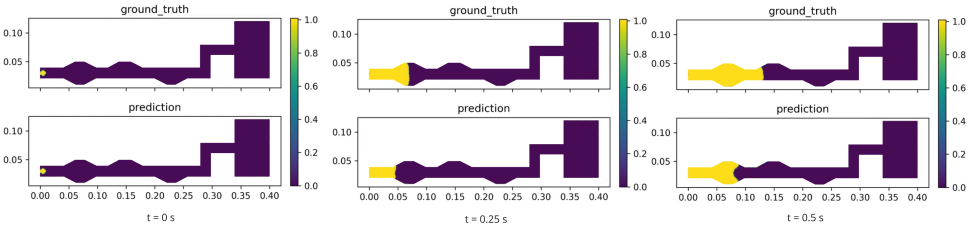


Figure 5.16: Filling of the channel: ground truth and prediction. Frames 1-3.

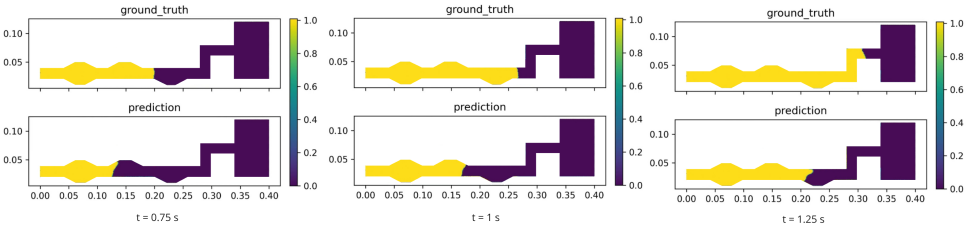


Figure 5.17: Filling of the channel: ground truth and prediction. Frames 4-6.

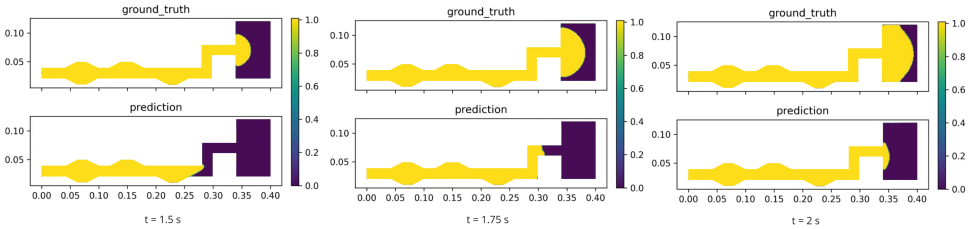


Figure 5.18: Filling of the channel: ground truth and prediction. Frames 7-9.

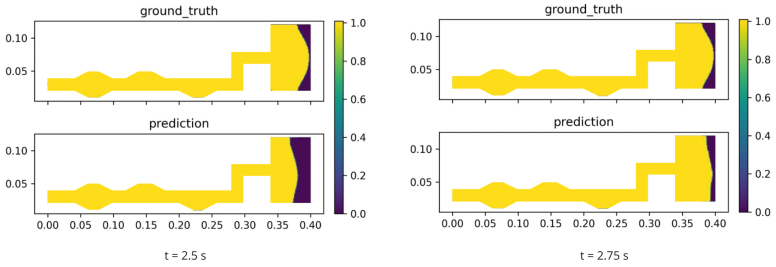


Figure 5.19: Filling of the channel: ground truth and prediction. Frames 10-11.

6

DISCUSSION

THE discussion is divided into four sections, debating the ground truth part, the graph neural network, answering the research questions posed in section 2.5, and potential directions for future research.

6.1. GROUND TRUTH DATA

There are a few elements concerning the ground truth data that can be adjusted or improved. One is using a solver that allows for a more detailed tuning of the viscosity, instead of the first-order model (described in subsection 3.2.3. Although the latter returned satisfactory agreements with a reference study as well as numerical convergence in subsection 3.3.1, the polymer melt presents complex behavior that might not be fully captured yet by the current method. The implementation of the compressibleInterFoam solver (allowing for other viscosity models) could be considered an option but would require a custom implementation since state-of-the-art shear-rate-dependent viscosity models are not natively implemented in OpenFOAM.

The randomly generated meshes as shown in Figure 3.3 contain unstructured mesh regions. These can be problematic for the numerical accuracy and solution stability of the simulations. During the qualitative validation in subsection 3.3.1, a structured mesh was utilized; however, no comparisons were made with unstructured meshes (as this was the case for the ground truth), nor was their effect analyzed. It is a challenge to find a meshing algorithm and geometry partitioning that would result in an equally structured mesh for all randomly generated geometries. Adjusting each single mesh is time-consuming and, therefore, not considered an option. It might be worth investigating parameters and geometry dependencies on the meshing process to promote overall more structured meshes.

The inlet is defined after the geometries are meshed. A region is cut out with a radius of 2 mm (see section 3.2), and the affected cells are removed. As a consequence, the

inlet shape is different for every geometry (see chapter 5). This was done to avoid affecting or distorting the original mesh and introducing unstructured regions. For the neural network, it represents an additional changing parameter. However, an idea could be to define the inlet circle at an early stage while creating the mesh, such that all geometries have the same inlet. This could be tackled simultaneously with the unstructured mesh issue (addressed in the previous paragraph) to ensure a good mesh quality for the whole geometry. For the outlet, the current method works fine and does not introduce additional distortions.

The backbone of the ground truth investigation in subsection 3.3.1 is proving numerical convergence and validation through comparison to a known experimental & numerical result. Although the agreements are regarded as sufficient, this statement could still be challenged, since there is no analytical verification. Regarding the velocity profiles in Figure 3.11, a comparison with the Poiseuille equation could be beneficial. This would provide an extra measure for the validity of the model.

Multiple slices in thickness of the ground truth model are pivotal in obtaining realistic behavior. At first, only one slice was considered (the respective results are not covered in this manuscript), which led to instabilities and unrealistic behavior.

To enhance its appeal to the industry, efforts should be focused on building a pipeline that leverages commercial software such as Moldex3D [79] and Moldflow [80], which are widely used for plastic injection molding simulations. These software packages are specifically designed for this purpose and are validated and accepted by the industry, eliminating the need for extensive validation procedures (in contrast to OpenFOAM, see subsection 3.3.1). Generating ground truth data can be done using one of these commercial tools, utilizing their respective APIs to extract mesh data (which, however, can still be a challenge, e.g., concerning automation). Material models can be accessed from pre-existing databases, and the simulation setup is straightforward. Additionally, by adopting industry-standard commercial software, the economic perspective (described in subsection 2.6.2) becomes relevant.

Throughout the ground truth investigation, the artifacts observed close to the melt front and boundaries (see, for instance, Figure 3.8) can be related to spurious currents in the air close to the interface. Their velocity can be a few orders of magnitudes above the polymer melt velocity, which has a huge impact on the Courant number. Unfortunately, they are very difficult to get rid of without using more advanced reconstruction schemes[1].

6.2. GRAPH NEURAL NETWORK

The rollouts of training, validation, and testing data show that the prediction is generally reliable at the start, but worsens with each successive timestep increment. An accumulation of errors is occurring. Pfaff *et al.* [33] reported that introducing random noise to the input can help mitigate this phenomenon, where the system has to first denoise the input and predict what happens after that. It can make the model more robust and im-

[1] See for instance: https://github.com/DLR-RY/TwoPhaseFlow

prove its generalizability.
A different idea stems from the fact that time is not explicitly accounted for in the network. It could be added as an additional explicit node- or global feature. Concerning this, instead of consistently predicting the next timestep, one could train the model to predict a model state for a random time step increment. These modifications could introduce a temporal awareness within the network.

A hyperparameters study is advised, perhaps inspired by Sanchez-Gonzalez *et al.* [31], Pfaff *et al.* [33]. The number of message-passing steps, the noise (if included) standard deviation, and history size are reported to impact the learning process the most.

The data extracted from the ground truth simulations in section 3.4 includes the velocity (x,y - direction) and pressure values. These are, however, not used as node features for the training of the network. To improve the predictions of the simulator, it could be beneficial to include the velocity and pressure values as additional node features. The learning process is likely to require more time, but could possibly improve the simulator's understanding of the flow behavior.

Using a triangular mesh for the ground truth meshing in section 3.1 could be advantageous because they are particularly easier to work with due to their inherent flexibility in representing complex geometries. This can be coupled with the unstructured mesh issues addressed in section 6.1. Furthermore, the PyTorch Geometric (PyG) library offers a convenient function that seamlessly integrates with triangular meshes, streamlining the pipeline. The render is also easily generated using a triangular mesh as described in subsection 4.3.2.

6.2.1. THE LOSS METRIC

The loss functions compared in Appendix D give a rough idea of their behavior and how they compare. For a steady-state flow or similar scenario where overall convergence is sought after, it performs well [33]. In this work, however, it is more of a dynamic setting. During the filling of the mold, most of the cells are either empty or filled. The melt front separates these two phases as it propagates through the mesh. Capturing this melt front behavior is essential, and the loss function should be tailored to prioritize it. This motivates the search for alternative loss functions. Some ideas are listed as follows:

- Weighted loss function: The idea is to assign different weights to different regions in the domain, e.g., give more weight to the loss of cells in and near the melt front. Focusing solely on the melt front is also an option. Since the melt moves mostly in one direction and bubbles are not present (aided by the implemented filter), saturated cells could be automatically held in this state, avoiding random decreases in the filling of cells that were previously filled.
- Gradient-based loss function: Use gradients/derivatives of alpha values and as a metric for the loss.
- Use shape functions to investigate the shape similarities between two melt front shapes. Python's `shapesimilarity` package [163] can serve this purpose. The first step could be extracting the nodes that make up the melt front at any given

time step. In the case of more complex shapes, a (polynomial) function could be fitted. This is followed by comparing the prediction with the target shape. A potential issue is that the same shape located at a different position within the geometry might yield a good score. So a sense of locality has to be introduced. This can be done by e.g., computing the centroids and distance between the target and the prediction melt front. These can then be added as a factor to the loss value.

- Similar to the shape comparison suggestion, one could also compare the melt front areas.
- Another approach would be focusing on the gradient between losses from a bad to a good case, instead of relying solely on the absolute values.
- The inlet BC entails a constant mass flow rate. This means that for each time step increment, the total filling of the mold increases by a constant amount. This could be added as an additional factor to the loss value.

Essentially, independently of the specific choice of loss metric, it should be invariant to the grid point density. Computing the mean by averaging the number of nodes of interest is a powerful tool embedded in the existing functions. A similar principle should be taken here.

A significant gap exists in physical understanding when evaluating GNN models using single scalar error metrics. Metrics such as the RMSE, MSE, and MAE (see Appendix D) quantify how well the approximation function matches the test data but do not provide insights into the learning process of underlying physics. A lower absolute loss value does not necessarily guarantee the physics is being captured successfully.

6.3. RESEARCH QUESTIONS

The research questions that emerged from the literature review in section 2.5 are addressed and answered.

1. Can a GNN learn the underlying physics of the injection molding process?

The graph neural network shows that it is able to learn the material behavior of the polymer melt. The melt front behavior is captured fairly accurately by the mid and best predictions of both training and validation rollouts. However, it does not always perform as well, and generalization is a challenge.

2. How does it compare to conventional methods?

The running time of the simulator's predictions is significantly faster than the ground truth data. It takes less than a second, compared to the several hours it takes to generate the ground truth simulation. This is a significant improvement.

3. How does the learned simulator perform in terms of generalizability?

The learned simulator manages to capture part of the material behavior but does not perform as well as during validation. Additionally, error accumulation is present, which leads to the prediction progressively diverging from the ground truth.

4. To what extent does the error metric correctly represent the simulator's accuracy?

This question touches on the fundamental conclusion: A small loss value does not affirm physical quantities of interest are met. It certainly gives an estimate but is inherently unreliable on its own. It is not expressive enough about the accuracy of the melt front behavior.

6.4. FUTURE RESEARCH

Short-term goals are suggested for the immediate follow-up research. Long-term goals are proposed with the idea in mind,

6.4.1. SHORT-TERM GOALS

The main suggestions addressed in the discussion are summarized as follows:

- Implement a shear-rate-dependent viscosity model for the polymer melt
- Increase the mesh quality
- Use software packages conventionally used in the industry for generating the training data
- Introduce noise at the input to tackle the accumulation of error
- Include time as an additional node or global feature in the network
- Conduct a hyperparameter study
- Include velocity and pressure values as additional node features
- Try different loss functions to improve the capturing of the melt front behavior

6.4.2. LONG-TERM GOALS

The current work presents the initial steps towards a much bigger objective. One where the approach to modeling the physical world is redefined. These are some of the envisioned long-term goals, fully acknowledging that they are ambitious and idealistic:

- 2D simulations serve as an important first step to assess the potential of the GNN to learn multiphase physics. It is however not realistic for actual injection molding. A transition to three-dimensional (3D) simulations could be a logical next step.
- The current loss metric is not expressive enough. Develop benchmarking tests to validate the models (with the same or similar principles as numerical models are validated):
  - Ensure generalizability, interpretability, and reliability of the models
  - Identify key architectural choices and develop code that is easy to use and reproducible
- Vary polymer type and processing conditions. Increase the complexity of the ground truth model.

- Explore rheological boundaries. How does a GNN train and learn general material models
- Transition from pure polymer to a composite consisting of polymer melt and fibers.
  - At first, use polymer melt and short, rigid, unbreakable elements immersed in the polymer matrix.
  - As fibers are introduced into the melt, it could be valuable to look into using a particle-based GNS approach [31] that accounts for the direct simulation of each individual melt and fiber particle.
  - Simulate long fiber-reinforced thermoplastics (LFTs), where the long fibers are allowed to bend and break. Fiber breakage is a known issue to tackle, as reported abundantly in literature [92]. Conventional simulations and predictions are time and computationally intensive. Training a GNN that can make the same predictions at a fraction of computation time would be a significant achievement.
- Start building a GNN polymer database, where GNNs are individually trained to understand the physics of a variety of polymeric materials and processing conditions.

The ultimate objective is to revolutionize the way physics is modeled. Analogously to the transition from experimental domains to numerical methods in the 1970s, the same vision can be applied here, where GNNs might replace conventional software, requiring a fraction of the computational cost compared to current traditional solvers.

7

CONCLUSION

While traditional FVM solvers are effective for injection molding simulations, their computational costs result in long processing times. Graph Neural Networks (GNNs) present a promising alternative as a data-driven approach, offering the potential for faster simulation times. It is evident that GNNs possess the ability to simulate complex physical systems. However, several challenges are faced that need to be addressed before they can fully replace traditional methods. The key insights drawn from the discussion are listed as follows.

- The Graph Neural Network is able to learn the physics of polymer melt behavior in 2D mold filling. It demonstrates significantly reduced computational time compared to the FVM solver, which is its most valuable advantage.
- Predictions do not always perform as well, mainly due to error accumulation during testing.
- A small loss value does not affirm physical quantities of interest are met. It gives an estimate but is inherently unreliable on its own.

In conclusion, the proposed future research focuses on enhancing the model's accuracy and robustness. Key improvements include introducing noise to reduce errors, incorporating time and additional physical features, conducting a hyperparameter study, and experimenting with different loss functions to better capture complex behaviors, such as melt front dynamics. Continued research and development are essential to overcome these hurdles and harness the full potential of GNNs. This work marks the first step towards realizing this objective.

REFERENCES

[1] Peggy Malnati, *Short carbon fiber compounds expand reach of thermoplastic composites,* (2022).

[2] S. K. Selvaraj, A. Raj, R. Rishikesh Mahadevan, U. Chadha, and V. Paramasivam, *A Review on Machine Learning Models in Injection Molding Machines,* Advances in Materials Science and Engineering **2022** (2022), 10.1155/2022/1949061.

[3] S.-J. Liu, *Injection molding in polymer matrix composites*, Tech. Rep. (Chang Gung University, 2012).

[4] M. Kazemi, S. Faisal Kabir, and E. H. Fini, *State of the art in recycling waste thermoplastics and thermosets and their applications in construction,* (2021).

[5] petrroudny43, *Vector graphic elements of amorphous, semi-crystalline, and crystalline polymer structure.* .

[6] A. Shrivastava, *Introduction to Plastics Engineering,* in *Introduction to Plastics Engineering* (Elsevier, 2018) pp. 1–16.

[7] J. Feng, L. Wang, R. Y. Zhang, J. J. Wu, C. Y. Wang, M. B. Yang, and X. R. Fu, *Formation of double skin-core orientated structure in injection-molded Polyethylene parts: Effects of ultra-high molecular weight Polyethylene and temperature field,* Journal of Polymer Research **21** (2014), 10.1007/s10965-014-0432-5.

[8] Romeo RIM, *What are composites?* (2023).

[9] A. HS, A. MF, M. A, and A. HA, *Minimization of Defects Percentage in Injection Molding Process using Design of Experiment and Taguchi Approach,* Industrial Engineering & Management **04** (2015), 10.4172/2169-0316.1000179.

[10] G. Kickelbick, *Polymer Composites* (Wiley - VCH, 2007) p. 498.

[11] E. Lafranche, P. Krawczak, J. P. Ciolczyk, and J. Maugey, *Injection moulding of long glass fiber reinforced polyamide 66: Processing conditions/microstructure/flexural properties relationship,* Advances in Polymer Technology **24**, 114 (2005).

[12] T. Kiatcharoenpol and T. Vichiraprasert, *Optimizing and Modeling for Plastic Injection Molding Process using Taguchi Method,* in *Journal of Physics: Conference Series*, Vol. 1026 (Institute of Physics Publishing, 2018).

[13] J. P. Holman and J. Lloyd, *McGraw-Hill Series in Mechanical Engineering CONSULTING EDITORS*, Tech. Rep.

[14] C. A. Huertas-Ortecho, *Robust Bird-Strike Modeling Using LS-DYNA*, Tech. Rep. (University of Puerto Rico, 2006).

[15] M. Rakhsha, C. E. Kees, and D. Negrut, *Lagrangian vs. Eulerian: An analysis of two solution methods for free-surface flows and fluid solid interaction problems*, Fluids **6** (2021), 10.3390/fluids6120460.

[16] D. Yang, K. Wu, L. Wan, and Y. Sheng, *A particle element approach for modelling the 3d printing process of fibre reinforced polymer composites*, Journal of Manufacturing and Materials Processing **1** (2017), 10.3390/jmmp1010010.

[17] H. Teufelsbauer, Y. Wang, M. C. Chiou, and W. Wu, *Flow-obstacle interaction in rapid granular avalanches: DEM simulation and comparison with experiment*, Granular Matter **11**, 209 (2009).

[18] Soga Research Group, *Material Point Method (MPM)*, (2023).

[19] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle, *A material point method for snow simulation*, Tech. Rep.

[20] R. Wemmenhove, *Numerical Simulation of Two-Phase Flow in Offshore Environments*, Tech. Rep. (2008).

[21] P. H. Sydenham and R. Thorn, *Handbook of measuring system design* (Wiley, 2005).

[22] C. R. Stoica, R. Maier, A. M. Istrate, S. G. Bucaciuc, and A. Despa, *Prediction of Polymer Flow Length by Coupling Finite Element Simulation with Artificial Neural Network*, Materiale Plastice **59**, 78 (2022).

[23] T. Sabiston, K. Inal, and P. Lee-Sullivan, *Application of Artificial Neural Networks to predict fibre orientation in long fibre compression moulded composite materials*, Composites Science and Technology **190** (2020), 10.1016/j.compscitech.2020.108034.

[24] R. Vinuesa and S. L. Brunton, *Enhancing Computational Fluid Dynamics with Machine Learning*, (2021), 10.1038/s43588-022-00264-7.

[25] H. Eivazi, M. Tahani, P. Schlatter, and R. Vinuesa, *Physics-informed neural networks for solving Reynolds-averaged Navier-Stokes equations*, (2021), 10.1063/5.0095270.

[26] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, *Graph neural networks: A review of methods and applications*, AI Open **1**, 57 (2020).

[27] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. Wiltschko, *A Gentle Introduction to Graph Neural Networks*, Distill **6** (2021), 10.23915/distill.00033.

[28] U. Alon and E. Yahav, *On the Bottleneck of Graph Neural Networks and its Practical Implications*, (2020).

[29] K. Ishikawa, *GNN: Over-smoothing*, (2021).

[30] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, *Relational inductive biases, deep learning, and graph networks,* (2018).

[31] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia, *Learning to Simulate Complex Physics with Graph Networks,* (2020).

[32] Y. Rubanova, A. Sanchez-Gonzalez, T. Pfaff, and P. Battaglia, *Constraint-based graph network simulator,* (2021).

[33] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, *Learning Mesh-Based Simulation with Graph Networks,* (2020).

[34] CFD Support, *zeroGradient boundary condition,* (2024).

[35] M. Baum, F. Jasser, M. Stricker, D. Anders, and S. Lake, *Numerical simulation of the mold filling process and its experimental validation,* International Journal of Advanced Manufacturing Technology **120**, 3065 (2022).

[36] Sumitomo Chemical Company, *Injection Molding Conditions,* (2022).

[37] Perkinelmer, *MP, Tg, and Structure of Common Polymers,* .

[38] U. K. Vaidya and K. K. Chawla, *Processing of fibre reinforced thermoplastic composites,* (2008).

[39] International association of plastics distribution, *IAPD Thermoplastics Rectangle,* (2015).

[40] E. Laranjeira, L. H. De Carvalho, S. M. Silva, and J. R. D'Almeida, *Influence of fiber orientation on the mechanical properties of polyester/jute composites,* Journal of Reinforced Plastics and Composites **25**, 1269 (2006).

[41] R. Farooque, M. Asjad, and S. J. Rizvi, *A current state of art applied to injection moulding manufacturing process - A review,* in *Materials Today: Proceedings*, Vol. 43 (Elsevier Ltd, 2020) pp. 441–446.

[42] T. A. Osswald and J. P. J. P. Ortiz, *Polymer processing : modeling and simulation* (Hanser Publishers, 2006) p. 606.

[43] J. Zhu, Z. Qiu, Y. Huang, and W. Huang, *Overview of injection molding process optimization technology,* in *Journal of Physics: Conference Series*, Vol. 1798 (IOP Publishing Ltd, 2021).

[44] K. Pulidindi and K. Ahuja, *Injection Molded Plastics Market Size,* (2023).

[45] A. Guevara-Morales and U. Figueroa-López, *Residual stresses in injection molded products,* (2014).

[46] S. Yashiro, T. Okabe, and K. Matsushima, *A numerical approach for injection molding of short-fiber-reinforced plastics using a particle method,* Advanced Composite Materials **20**, 503 (2011).

[47] V. M. Kryachek, *Powder Metallurgy and Metal Ceramics,* Tech. Rep. (Institute for Problems of Materials Science, Kiev, 2004).

[48] Z. Tadmor and C. G. Gogos, *An SPE*, 2nd ed. (John Wiley & Sons, Inc., Publication, 2006).

[49] L. Utracki and T. Vu Khanh, *Filled polymers,* Multicomponent Polymer Systems , 207 (2002).

[50] P. P. K. Kennedy and R. Zheng, *Flow analysis of injection molds* (Hanser Publishers, 2013) p. 349.

[51] S. K. Mazumdar, *Composites manufacturing : materials, product, and process engineering* (CRC Press, 2002) p. 392.

[52] J.-P. Pascault and R. J. J. Williams, *28 THERMOSETTING POLYMERS*, Tech. Rep. (2013).

[53] S. D. Anuar Sharuddin, F. Abnisa, W. M. A. Wan Daud, and M. K. Aroua, *A review on pyrolysis of plastic wastes,* (2016).

[54] G. Jogur, A. Nawaz Khan, A. Das, P. Mahajan, and R. Alagirusamy, *Impact properties of thermoplastic composites,* https://doi.org/10.1080/00405167.2018.1563369 **50**, 109 (2019).

[55] S. Kashyap and D. Datta, *Process parameter optimization of plastic injection molding: a review,* International Journal of Plastics Technology **19**, 1 (2015).

[56] N. D. Polychronopoulos and J. Vlachopoulos, *Polymer Processing and Rheology,* (2019) pp. 1–47.

[57] P. Hirsch, M. John, D. Leipold, A. Henkel, S. Gipser, R. Schlimper, M. Zscheyge, C. Bruno, P. Fernandes, A. Faroughi, L. L. Ferrás, and A. M. Afonso, *Numerical Simulation and Experimental Validation of Hybrid Injection Molded Short and Continuous Fiber-Reinforced Thermoplastic Composites,* Polymers 2021, Vol. 13, Page 3846 **13**, 3846 (2021).

[58] S. MohammadKarimi, B. Neitzel, M. Lang, and F. Puch, *Investigation of the Fiber Length and the Mechanical Properties of Waste Recycled from Continuous Glass Fiber-Reinforced Polypropylene,* Recycling **8** (2023), 10.3390/recycling8060082.

[59] A. B. Senior and T. Osswald, *Measuring fiber length in the core and shell regions of injection molded long fiber-reinforced thermoplastic plaques,* Journal of Composites Science **4** (2020), 10.3390/jcs4030104.

[60] R. O. Ritchie, *The conflicts between strength and toughness,* Nature Materials **10**, 817 (2011).

[61] S. MOK and C. Kwong, *Application of artificial neural network and fuzzy logic in a case-based system for initial process parameter setting of injection molding,* Journal of Intelligent Manufacturing **13** (2002).

[62] P. Zhao, J. Zhang, Z. Dong, J. Huang, H. Zhou, J. Fu, and L.-S. Turng, *Intelligent Injection Molding on Sensing, Optimization, and Control,* Advances in Polymer Technology **2020**, 1 (2020).

[63] H. Ning, N. Lu, A. A. Hassen, K. Chawla, M. Selim, and S. Pillay, *A review of Long fibre thermoplastic (LFT) composites,* International Materials Reviews **65**, 164 (2020).

[64] R. A. Malloy, *Plastic part design for injection molding : an introduction* (Hanser Publishers, 2010) p. 549.

[65] J. L. Thomason and M. A. Vlug, *Conlpo.viti˜ Purr A*, Tech. Rep. (Owens-Corning Science and Technology Center, 1996).

[66] M. Kurt, Y. Kaynak, O. S. Kamber, B. Mutlu, B. Bakir, and U. Koklu, *Influence of molding conditions on the shrinkage and roundness of injection molded parts,* International Journal of Advanced Manufacturing Technology **46**, 571 (2010).

[67] M. Ren, J. Gu, Z. Li, S. Ruan, and C. Shen, *Simulation of Polymer Melt Injection Molding Filling Flow Based on an Improved SPH Method with Modified Low-Dissipation Riemann Solver,* Macromolecular Theory and Simulations **31** (2022), 10.1002/mats.202100029.

[68] *Lagrangian and Eulerian specification of the flow field,* (2024).

[69] Z. Zhang and Q. Chen, *Comparison of the Eulerian and Lagrangian methods for predicting particle transport in enclosed spaces*, Tech. Rep. (Purdue University, 2007).

[70] W. Xie, Z. Liu, and Y. L. Young, *Application of a coupled Eulerian-Lagrangian method to simulate interactions between deformable composite structures and compressible multiphase flow,* International Journal for Numerical Methods in Engineering **80**, 1497 (2009).

[71] G. Qiu, S. Henke, and J. Grabe, *Application of a Coupled Eulerian-Lagrangian approach on geomechanical problems involving large deformations,* Computers and Geotechnics **38**, 30 (2011).

[72] M. Guida, F. Marulo, M. Meo, A. Grimaldi, and G. Olivares, *SPH - Lagrangian study of bird impact on leading edge wing,* Composite Structures **93**, 1060 (2011).

[73] itascacg, *Built-in Contact Models,* (2021).

[74] M. A. Benitz, M. A. Lackner, and D. P. Schmidt, *Hydrodynamics of offshore structures with specific focus on wind energy applications,* (2015).

[75] C. Greenshields and H. Weller, *The PIMPLE algorithm,* (2022).

[76] ANSYS, *Ansys Fluent,* (2024).

[77] COMSOL, *Comsol Multiphysics,* (2024).

[78] Dassault Systemes, *Simulia Abaqus,* (2024).

[79] Moldex3D, *Moldex3D,* (2024).

[80] Autodesk, *Moldflow,* (2020).

[81] OpenFOAM, *OpenFOAM,* (2024).

[82] solids4foam, *solids4foam,* (2024).

[83] Elmer FEM, *Elmer FEM,* (2024).

[84] J. S. Cintra and C. L. Tucker, *Orthotropic closure approximations for flow - induced fiber orientation,* Journal of Rheology **39**, 1095 (1995).

[85] N. Phan-Thien, X.-J. Fan, R. I. Tanner, and R. Zheng, *J. Non-Newtonian Fluid Mech,* Tech. Rep. (2002).

[86] R. J. Phillips, R. C. Armstrong, R. A. Brown, A. L. Graham, and J. R. Abbott, *A constitutive equation for concentrated suspensions that accounts for shear-induced particle migration,* Physics of Fluids A **4**, 30 (1992).

[87] C. L. Tucker, J. Wang, and J. F. O'gara, *Method and article of manufacture for determining a rate of change of orientation of a plurality of fibers disposed in a fluid,* Tech. Rep. (2007).

[88] J. Wang, J. F. O'Gara, and C. L. Tucker, *An objective model for slow orientation kinetics in concentrated fiber suspensions: Theory and rheological evidence,* Journal of Rheology **52**, 1179 (2008).

[89] M. Sepehr, G. Ausias, and P. J. Carreau, *Rheological properties of short fiber filled polypropylene in transient shear flow,* Journal of Non-Newtonian Fluid Mechanics **123**, 19 (2004).

[90] J. Férec, G. Ausias, M. C. Heuzey, and P. J. Carreau, *Modeling fiber interactions in semiconcentrated fiber suspensions,* Journal of Rheology **53**, 49 (2009).

[91] J. H. Phelps, A. I. Abd El-Rahman, V. Kunc, and C. L. Tucker, *A model for fiber length attrition in injection-molded long-fiber composites,* Composites Part A: Applied Science and Manufacturing **51**, 11 (2013).

[92] A. S. Chauhan, *Simulation and analytical study of fiber breakage of long fiber reinforced thermoplastic composites in injection molding,* Tech. Rep. (TU Delft, Delft, 2022).

[93] G. Manuel Vélez-García, *Experimental Evaluation and Simulations of Fiber Orientation in Injection Molding of Polymers Containing Short Glass Fibers*, Tech. Rep. (2012).

[94] F. Folgar and C. L. Tucker, *Orientation Behavior of Fibers in Concentrated Suspensions,* Journal of Reinforced Plastics and Composites **3**, 98 (1984).

[95] S. G. Advani and C. L. Tucker, *The Use of Tensors to Describe and Predict Fiber Orientation in Short Fiber Composites,* Journal of Rheology **31**, 751 (1987).

[96] G. M. Vélez-García, S. M. Mazahir, P. Wapperom, and D. G. Baird, *Simulation of Injection Molding Using a Model with Delayed Fiber Orientation*, Tech. Rep. (2011).

[97] A. P. Eberle, G. M. Vélez-García, D. G. Baird, and P. Wapperom, *Fiber orientation kinetics of a concentrated short glass fiber suspension in startup of simple shear flow,* Journal of Non-Newtonian Fluid Mechanics **165**, 110 (2010).

[98] X. Fan, N. Phan-Thien, and R. Zheng, *J. Non-Newtonian Fluid Mech*, Tech. Rep. (1998).

[99] S. A. Simon, A. Bechara Senior, and T. Osswald, *Experimental validation of a direct fiber model for orientation prediction,* Journal of Composites Science **4** (2020), 10.3390/jcs4020059.

[100] T. C. Chang, A. B. Senior, H. Celik, D. Brands, A. Yanev, and T. Osswald, *Validation of fiber breakage in simple shear flow with direct fiber simulation,* Journal of Composites Science **4** (2020), 10.3390/jcs4030134.

[101] K. Wu, L. Wan, H. Zhang, and D. Yang, *Numerical simulation of the injection molding process of short fiber composites by an integrated particle approach,* International Journal of Advanced Manufacturing Technology **97**, 3479 (2018).

[102] S. Yamamoto and T. Matsuoka, *A method for dynamic simulation of rigid and flexible fibers in a flow field,* The Journal of Chemical Physics **98**, 644 (1993).

[103] S. Yamamoto and T. Matsuoka, *Dynamic simulation of fiber suspensions in shear flow,* The Journal of Chemical Physics **102**, 2254 (1995).

[104] S. Yamamoto and T. Matsuoka, *Viscosity of dilute suspensions of rodlike particles: A numerical simulation method,* The Journal of Chemical Physics **100**, 3317 (1994).

[105] C. G. Joung, N. Phan-Thien, and X. J. Fan, *J. Non-Newtonian Fluid Mech*, Tech. Rep. (2001).

[106] C. G. Joung, N. Phan-Thien, and X. J. Fan, *J. Non-Newtonian Fluid Mech*, Tech. Rep. (2002).

[107] P. K. D. V. Yarlagadda, C. Ang, and T. Khong, *Development of a hybrid neural network system for prediction of process parameters in injection moulding*, Tech. Rep. (2001).

[108] T. Wuest, D. Weimer, C. Irgens, and K. D. Thoben, *Machine learning in manufacturing: Advantages, challenges, and applications,* Production and Manufacturing Research **4**, 23 (2016).

[109] I. Meekers, P. Refalo, and A. Rochman, *Analysis of Process Parameters affecting Energy Consumption in Plastic Injection Moulding,* in *Procedia CIRP*, Vol. 69 (Elsevier B.V., 2018) pp. 342–347.

[110] Wikipedia contributors, *Latent and observable variables,* (2024).

[111] P. Dayan, *Unsupervised Learning*, Tech. Rep. (1999).

[112] M. Cord and P. Cunningham, *Machine Learning Techniques for Multimedia* (Springer, 2008).

[113] Y. Adan, *Do neural networks really work like neurons?* (2018).

[114] D. J. Livingstone, *Artificial Neural Networks*, Tech. Rep.

[115] S. L. Clainche, E. Ferrer, S. Gibson, E. Cross, A. Parente, and R. Vinuesa, *Improving aircraft performance using machine learning: a review,* (2022), 10.1016/j.ast.2023.108354.

[116] S. L. Brunton, M. S. Hemati, and K. Taira, *Special issue on machine learning and data-driven methods in fluid dynamics,* (2020).

[117] N. Wandel, M. Weinmann, and R. Klein, *Teaching the Incompressible Navier-Stokes Equations to Fast Neural Surrogate Models in 3D,* (2020), 10.1063/5.0047428.

[118] R. E. Meethal, A. Kodakkal, M. Khalil, A. Ghantasala, B. Obst, K. U. Bletzinger, and R. Wüchner, *Finite element method-enhanced neural network for forward and inverse problems,* Advanced Modeling and Simulation in Engineering Sciences **10** (2023), 10.1186/s40323-023-00243-1.

[119] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer, *Machine learning accelerated computational fluid dynamics,* (2021), 10.1073/pnas.2101784118.

[120] U. Frisch, *Turbulence: the legacy of AN Kolmogorov* (Cambridge University Press, 1995).

[121] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,* Journal of Computational Physics **378**, 686 (2019).

[122] Y. Kim, Y. Choi, D. Widemann, and T. Zohdi, *A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder,* (2020).

[123] H. Eivazi and R. Vinuesa, *Physics-informed deep-learning applications to experimental fluid mechanics,* (2022).

[124] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackerman, V. M. Tran, A. Chiappino-Pepe, A. H. Badran, I. W. Andrews, E. J. Chory, G. M. Church, E. D. Brown, T. S. Jaakkola, R. Barzilay, and J. J. Collins, *A Deep Learning Approach to Antibiotic Discovery,* Cell **180**, 688 (2020).

[125] B. Sanchez-Lengeling, J. N. Wei, B. K. Lee, R. C. Gerkin, A. Aspuru-Guzik, and A. B. Wiltschko, *Machine Learning for Scent: Learning Generalizable Perceptual Representations of Small Molecules,* (2019).

[126] M. Allamanis, M. Brockschmidt, and M. Khademi, *Learning to Represent Programs with Graphs,* (2017).

[127] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, P. W. Battaglia, V. Gupta, A. Li, Z. Xu, A. Sanchez-Gonzalez, Y. Li, and P. Velickovic, *ETA Prediction with Graph Neural Networks in Google Maps,* in *International Conference on Information and Knowledge Management, Proceedings* (Association for Computing Machinery, 2021) pp. 3767–3776.

[128] J. Kunegis, *Proceedings of the 22nd International Conference on World Wide Web - WWW '13 Companion,* (Association for Computing Machinery, New York, NY, USA, 2013).

[129] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, *Interaction Networks for Learning about Objects, Relations and Physics,* (2016).

[130] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, *Molecular Graph Convolutions: Moving Beyond Fingerprints,* (2016), 10.1007/s10822-016-9938-8.

[131] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, *Neural Relational Inference for Interacting Systems*, Tech. Rep. (2018).

[132] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, *Neural Message Passing for Quantum Chemistry,* (2017).

[133] F. Harary, *Graph Theory* (Addison-Wesley, 1971).

[134] A. Poulovassilis, *ACM Transactions on Information Systems*, Tech. Rep. 1 (1994).

[135] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar, *HyperGCN: A New Method of Training Graph Convolutional Networks on Hypergraphs,* (2018).

[136] Z. Zhong, C.-T. Li, and J. Pang, *Hierarchical Message-Passing Graph Neural Networks,* (2020).

[137] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, *Benchmarking Graph Neural Networks,* (2020).

[138] J. You, R. Ying, and J. Leskovec, *Design Space for Graph Neural Networks,* (2020).

[139] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, *Principal Neighbourhood Aggregation for Graph Nets,* (2020).

[140] Google, *Google DeepMind,* (2024).

[141] S. Sukhbaatar, A. Szlam, and R. Fergus, *Learning Multiagent Communication with Backpropagation*, Tech. Rep.

[142] N. Watters, A. Tacchetti, T. Weber, R. Pascanu, P. Battaglia, and D. Zoran, *Visual Interaction Networks: Learning a Physics Simulator from Video*, Tech. Rep.

[143] Y. Hoshen, *VAIN: Attentional Multi-agent Predictive Modeling*, Tech. Rep.

[144] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, *Graph Networks as Learnable Physics Engines for Inference and Control,* (2018).

[145] J. L. McClelland, *International Perspectives on Psychological Scienc˜*, Tech. Rep. (1994).

[146] M. Fortunato, T. Pfaff, P. Wirnsberger, A. Pritzel, and P. Battaglia, *MultiScale MeshGraphNets,* (2022).

[147] V. G. Satorras, E. Hoogeboom, and M. Welling, *E(n) Equivariant Graph Neural Networks,* (2021).

[148] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, *Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids,* (2018).

[149] M. Z. Naser and A. H. Alavi, *Error Metrics and Performance Fitness Indicators for Artificial Intelligence and Machine Learning in Engineering and Sciences,* Architecture, Structures and Construction **3**, 499 (2023).

[150] Autodesk, *Moldflow Subscription,* (2024).

[151] C. Geuzaine and J.-F. Remacle, *A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities,* (2024).

[152] Python Software Foundation, *Python,* (2024).

[153] C. Bonamy, *FluidFoam,* (2020).

[154] G. Zitzenbacher, H. Dirnberger, M. Längauer, and C. Holzer, *Calculation of the contact angle of polymer melts on tool surfaces from viscosity parameters,* Polymers **10** (2018), 10.3390/polym10010038.

[155] OpenFOAMWiki, *interFoam,* (2023).

[156] F. White, *Fluid Mechanics* (2011).

[157] P. Developers, *PyVista,* (2019).

[158] Open CFD Ltd, *OpenFOAM Documentation,* (2024).

[159] M. Fey and J. E. Lenssen, *Fast Graph Representation Learning with PyTorch Geometric,* (2019).

[160] The Linux Foundation, *PyTorch,* (2024).

[161] NVIDIA Corporation, *CUDA Toolkit,* (2024).

[162] Matplotlib development team, *matplotlib,* (2024).

[163] N. Wenner, *shapesimilarity,* (2024).

[164] S. Gupta, *The 7 Most Common Machine Learning Loss Functions,* (2024).

A

COMPLETE SOLVER SETTINGS

Setting	Specification
timePrecision	6
ddtSchemes	Euler
gradSchemes	Gauss linear
divSchemes	div(rhoPhi,U): Gauss linearUpwindV grad(U) div(phi,alpha): Gauss interfaceCompression vanLeer 1 div(((rho*nuEff)*dev2(T(grad(U))))): Gauss linear
laplacianSchemes	Gauss linear orthogonal
interpolationSchemes	linear
snGradSchemes	orthogonal

Table A.1: Additional control and solver schemes settings.

Setting	Specification
alpha.epoxy.*	
nAlphaCorr	2
nAlphaSubCycles	1
MULESCorr	yes
nLimiterIter	5
alphaApplyPrevCorr	yes
solver	smoothSolver
smoother	symGaussSeidel
tolerance	1e-8
relTol	0
pcorr.*	
solver	PCG
preconditioner	DIC
tolerance	1e-6
relTol	0
minIter	1
p_rgh	
solver	PCG
preconditioner	DIC
tolerance	1e-05
relTol	0.05
p_rghFinal	
solver	PCG
preconditioner	DIC
tolerance	1e-08
relTol	0
U	
solver	PBiCGStab
preconditioner	DILU
tolerance	1e-08
relTol	0
minIter	1

Table A.2: Additional control and solver solution settings.

B

GROUND TRUTH PIPELINE

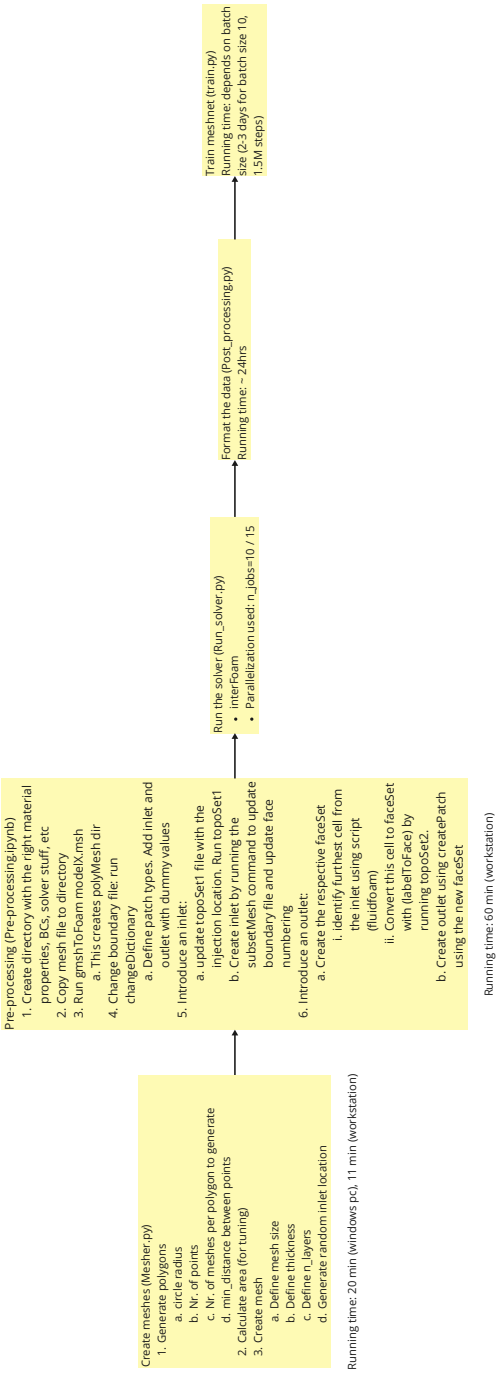


Figure B.1: Ground truth pipeline

C

TRAINING PIPELINE

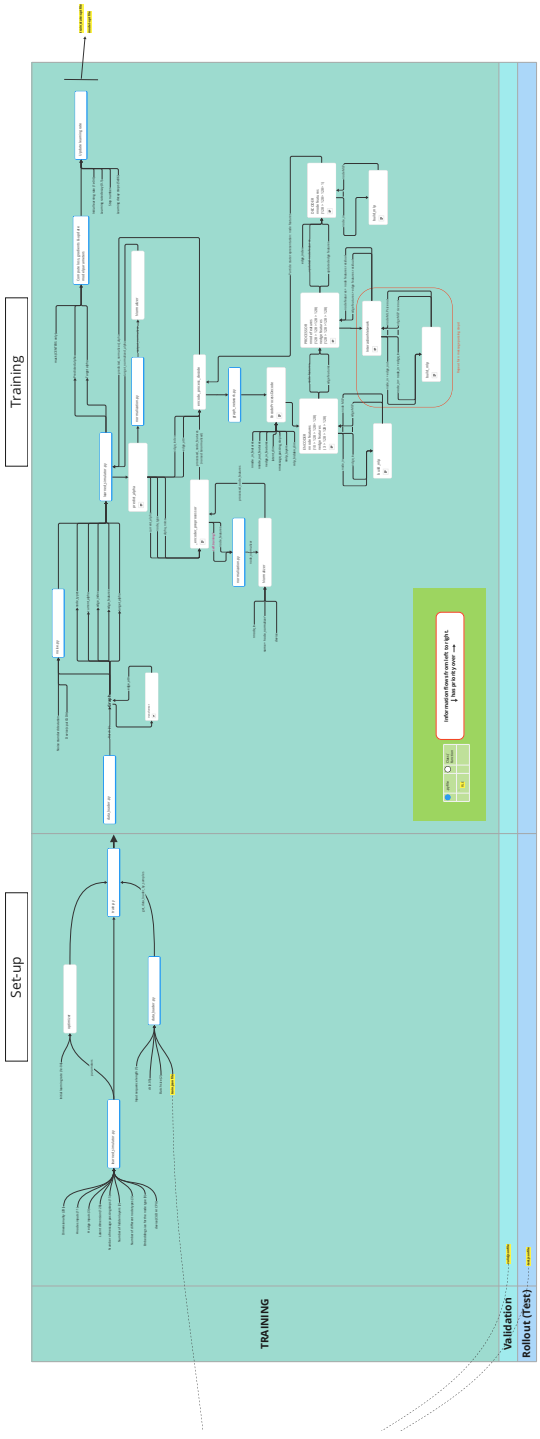


Figure C.1: Training procedure pipeline

D

LOSS INVESTIGATION

This investigation aims to choose different loss functions and see how well they perform in predicting the melt front during the filling process. A grid of 20x20 is constructed, where each of the 400 cells has a filling value (α), where $0 \leq \alpha \leq 1$. Three scenarios are constructed: a random case, a linear melt front, and one case with a circular melt front. These cases are depicted in Figure D.1.

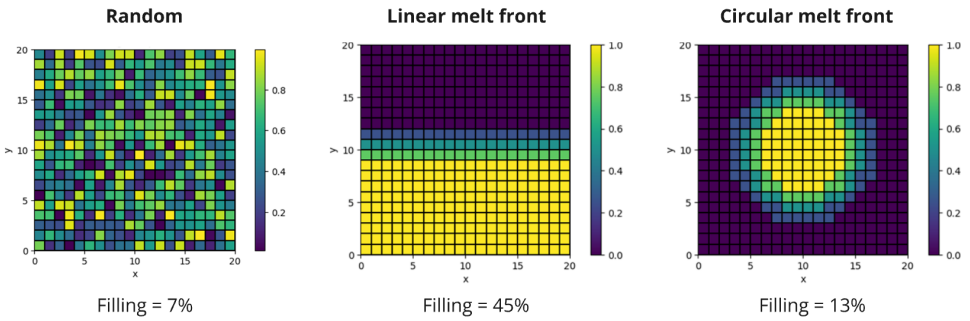


Figure D.1: Target melt front cases. These are to be regarded as the ground truth cases for this investigation.

The idea is now to generate the same grid but with different types of melt (front) shapes, compare them to the target scenarios in Figure D.1, and compute the loss function for each. Some commonly used loss functions for regression are included [164]: the Mean Squared Error (MSE), the Root Mean Squared Error (RMSE), the Mean Average Error (MAE), and the Huber Loss function (also called the Smooth Mean Absolute Error) with the tunable hyperparameter δ. The functions are shown below.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2 \tag{D.1}$$

The MSE penalizes the prediction for making large errors by squaring them. This function is highly sensitive to outliers.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2} \tag{D.2}$$

The RMSE also significantly penalizes outliers, but the square root scales it back to the unit of the original data, which can provide more interpretable results.

$$\text{MAE} = \frac{\sum_{i=1}^{N}|y_i - x_i|}{N} \tag{D.3}$$

The MAE computes the average magnitude of errors, which makes it more robust to outliers than the MSE, for instance.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \le \delta, \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases} \tag{D.4}$$

The Huber loss function can be regarded as a combination of MSE and MAE. It replicates the MAE approach that becomes quadratic when the error is small. This cut-off between the two behaviors is controlled by the tuneable hyperparameter δ. It determines the classification of an outlier. Two common values [164] are considered here: $\delta = 0.1$ and $\delta = 1$.

27 different melt front cases are generated. They are labeled as predicted scenarios, and compared to the three ground truth scenarios in Figure D.1. They range from a randomized melt front to similar and target cases. The aim is to investigate the value of each loss function and how it converges from other scenarios (high loss values) to similar (low loss values) and the respective target scenario (loss = 0).

Case 0 has randomized alpha values for all cells. The *(i)* shift (down/up), *(ii)* noise, *(iii)* smaller, *(iv)* bigger, means that this operation has been performed on the most similar target ground truth case. For instance, in case 10 a downward shift is introduced to the target linear melt front shown in Figure D.1. In case 20, the circular melt front in Figure D.1 is replicated, but with a slightly bigger radius. Combinations of these operations are also possible. The three target scenarios are also included in this set. Computing the loss function for each scenario compared to the three ground truth cases yields the following plots:

As expected, prediction 1 has a 0 loss value for all metrics. All other predictions have more or less the same value for each loss function respectively. The highest loss value is represented by RMSE, followed by MAE, MSE, and the two Huber loss functions.

Most of the loss values are quite intuitive for the linear and circular cases. For the linear case, predictions 10-14 stand out. The way that the functions converge to the ideal case differs. RMSE consistently penalizes the deviations the most, and its gradient descent towards the target case is also the steepest. This behavior is also present in the circular melt front graph, where the RMSE penalizes deviations the heaviest.

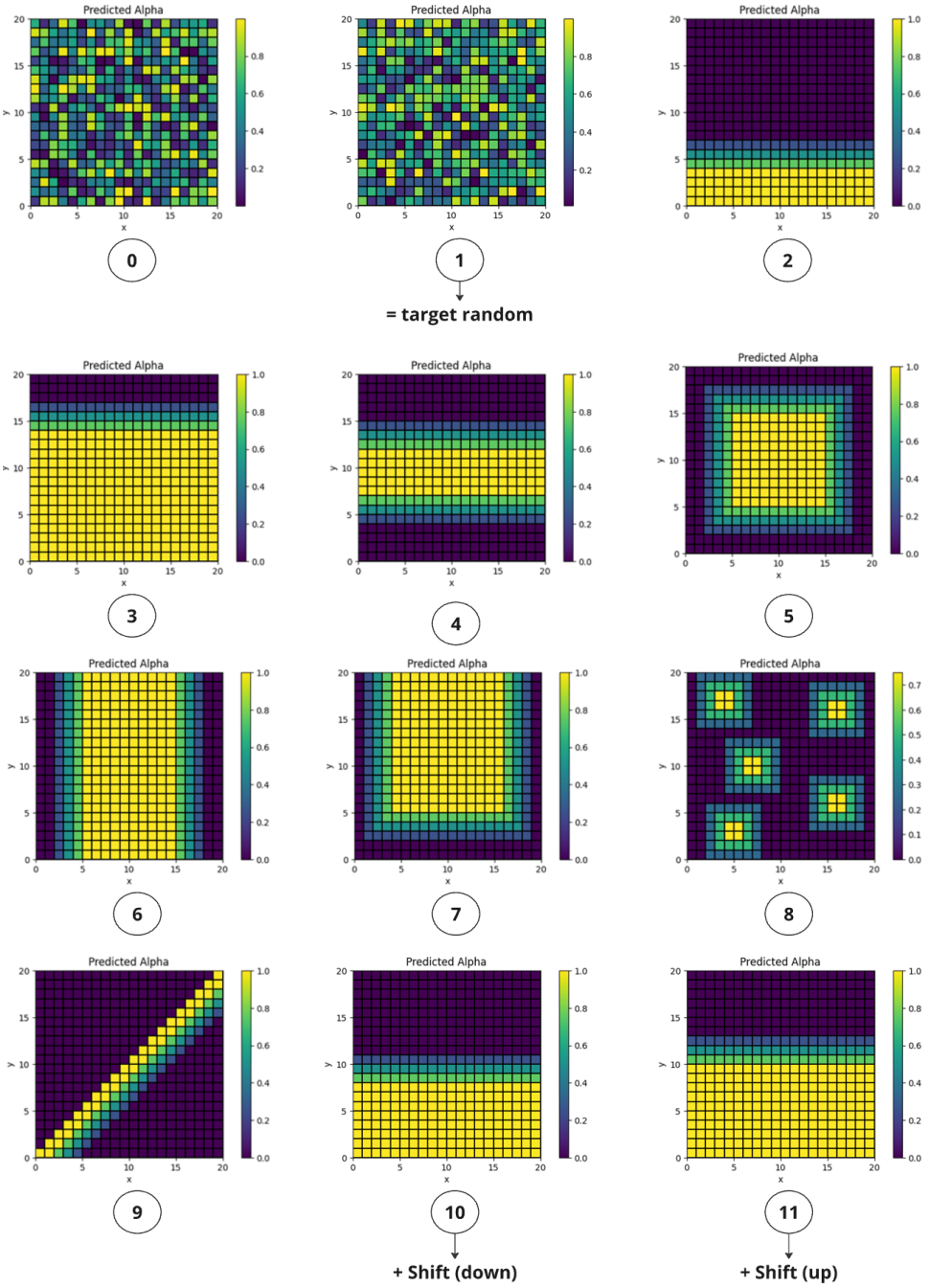


Figure D.2: Predicted scenarios 1.

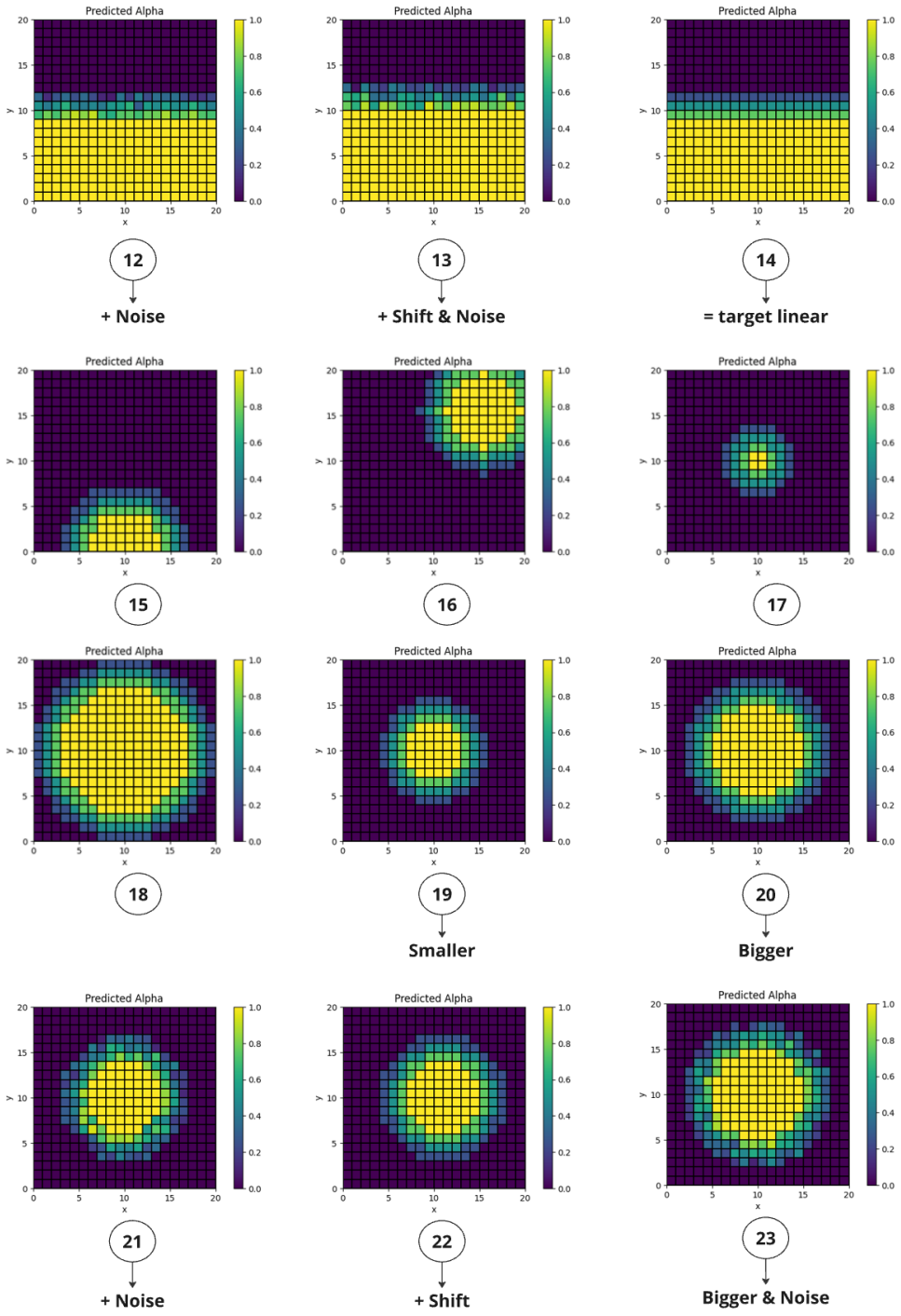


Figure D.3: Predicted scenarios 2.

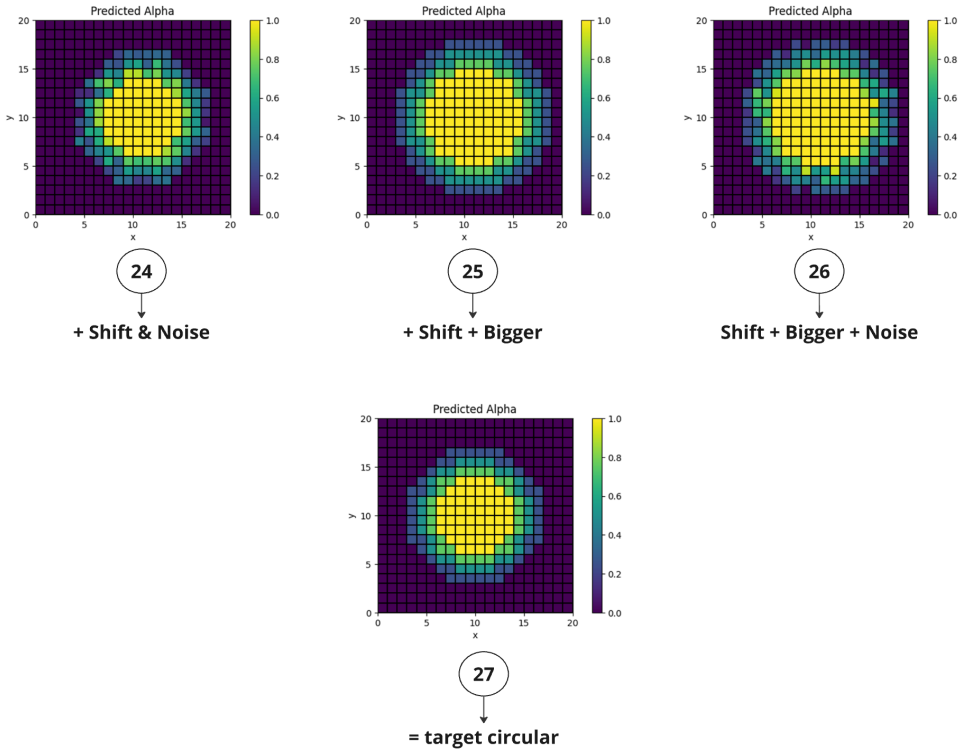


Figure D.4: Predicted scenarios 3.

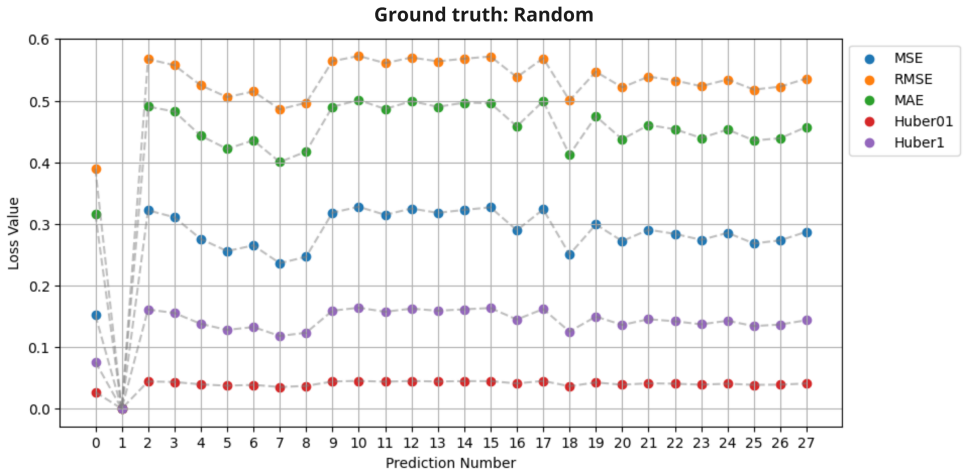


Figure D.5: Random melt front case. Loss values correspond to the predicted scenarios.

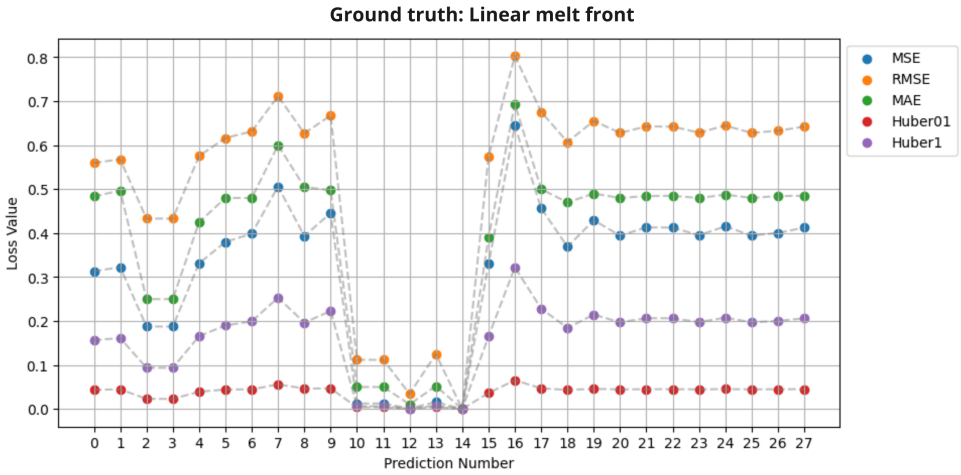


Figure D.6: Linear melt front case. Loss values correspond to the predicted scenarios.

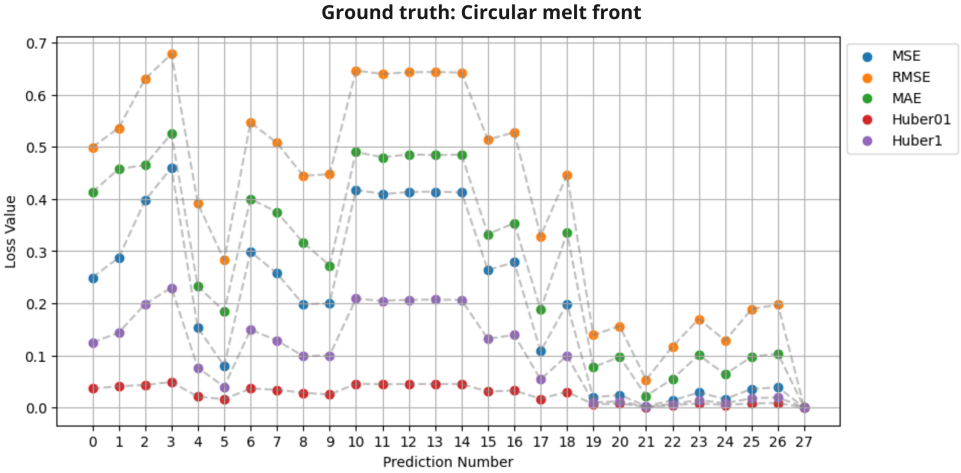


Figure D.7: Circular melt front case. Loss values correspond to the predicted scenarios.