MSc Artificial Intelligence
Master Thesis

# An Exploratory Study into using Machine-Learning for Fast Step-by-step Emulation of Numerical Mechanical Thrombectomy Simulations for Ischemic Stroke

by
Thijs Stessen
15159825

January 31, 2026

*Supervisor:*
MSc. Thijs Kuipers

*Examiner:*
Dr. Hoel Kervadec

Universiteit van Amsterdam

**Note to the reader**

This thesis contains terms from Physics, Medicine and AI. In some cases these terms overlap and mean different things depending on which field they are from, in which case we have split them up using an appropriate adjective (such as splitting 'mesh' into 'metal mesh' and 'mathematical mesh'). In other cases very similar concepts are used under different names, making it confusing what the differences are. When either of such cases occur we have written the words in *italics* and added an entry to the glossary.

# Glossary

**mathematical mesh** A set of points connected by edges, representing the surface of some object.

**metal mesh** Material like a net with spaces in it, made from metal wire.

**surrogate model** A computer program that can be used instead of something else (such as another computer program).

**phantom** A physical object mimicking a part of a body, that can be used instead of the real part (for training or experiments).

**physical outcome** What literally happens with all the objects involved in a medical procedure (such as how the blood clot moves when doing mechanical thrombectomy).

**clinical outcome** What happens to the medical condition of a patient in the hours, months or years after a medical procedure (such as death, or full recovery).

**artificial neural network** a machine learning model that uses parameters learned on data, linear algebra and non-linear functions to predict outputs given inputs.

**biological neural network** A set of connected neuron cells ex. the brain.

# Contents

**Abstract**

The treatment of ischemic stroke using mechanical thrombectomy involves difficult decisions under intense time constraints. Numerical physics simulations can in theory inform operators to make better decisions regarding treatment approaches and device selection, but are too slow to do so in practice. In this thesis, we investigate if current machine learning based surrogates can accurately emulate these simulations in a step-by-step manner while making them significantly faster. To do this we train three surrogate models on two simulations that involve a simplified aspiration procedure, with varying levels of geometric complexity. Our results show that two of our models accurately predict singular simulation steps and provide substantial speedups, especially when combined with specific data augmentations. However, the models showed a lack of stability when emulating simulations with complex geometries over longer time periods. Overall, this work provides a foundation for future studies to develop stable methods that scale to realistic numerical physics simulations of mechanical thrombectomy.

# Chapter 1

# Introduction

Acute Ischemic stroke (AIS), caused by a blood clot blocking an artery in the brain, results in the death of more than 3.2 million people per year globally as of 2019 and is projected to cause 4.9 million deaths by 2030 [1]. Treatment of AIS is often not successful; in the Netherlands, around 20 percent of patients undergoing endovascular treatment die within 90 days, 48 percent never fully recover neurologically and only 32 percent achieve a good *clinical outcome*, i.e. they recover most of their neurological capabilities [2]. In the United States, this 90 day mortality is around 14 percent [3] (although this is only based on privately insured individuals, which introduces a selection bias). Treatment of AIS broadly follows the following steps: imaging the brain using methods such as computed tomography (CT); administering a drug into the bloodstream that is meant to dissolve blood clots (intravenous thrombolysis with alteplase); if the patients health allows and if the blood clot is blocking a large vessel such as the middle cerebral artery or the intracranial carotid artery, physical removal of the blood clot via mechanical thrombectomy [4].

Mechanical thrombectomy involves bringing a device through the patients vascular system to the location of the blood clot and then deploying it. This device is either an aspiration device which sucks in the blood clot, a stent retriever which encapsulates the blood clot using a physical net-like *metal mesh* or a combination of both [5]. Many different designs exist but in practice, the decision which mechanical thrombectomy device to use (from the selection available at that hospital) is left to the discretion of the operator, as there is currently no guideline on which approved device to use.

When mechanical thrombectomy is performed, the amount of attempts taken to remove the blood clot must be minimized to achieve the best *clinical outcome*. In fact, when only one attempt is needed a so called first pass effect is observed and the chance of a good *clinical outcome* is much higher [6]. Moreover, each attempt comes at the risk of the defragmentation of the blood clot, possibly resulting in the fragments blocking other smaller blood vessels in the brain [7] which cannot be removed by mechanical thrombectomy. It is thus crucial to select a mechanical thrombectomy approach that offers a high chance of success while minimizing clot fragmentation.

How successful an attempt is and if fragmentation occurs, which we will call the *physical outcome*, is depended on the device used and the exact geometry of the blood vessel [8]. However these two aspects also impact the *physical outcome* interdependently: some devices perform better in some geometries. For example, a stent retriever that is too small can create shearing of the blood clot, possibly creating small fragments or not fully removing the clot [8]. This might seem like a trivial example: simply choose the stent with the diameter that is closest to that of the blood clot. However, the choice of the correct approach is more nuanced in the case that the blood clot is of an intermediate size. Is a slightly larger or a slightly smaller stent better in that case? Additionally, when mechanical thrombectomy attempts fail multiple times,

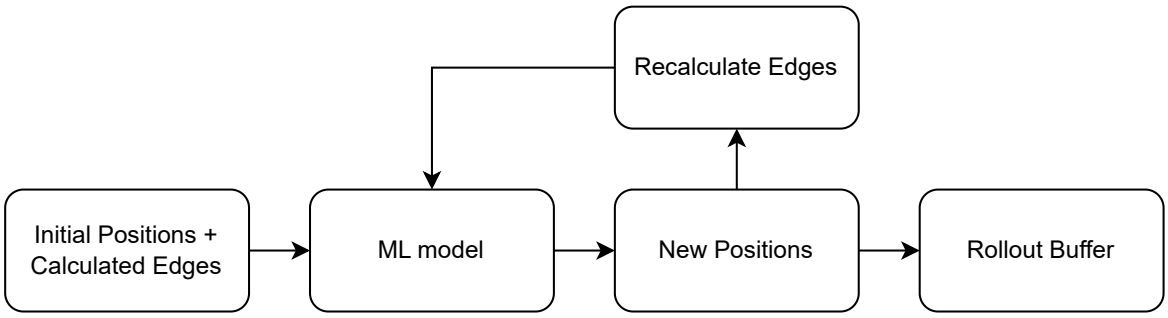



Figure 1.1: The step-by-step approach using a machine learning model as a *surrogate model*. The Rollout buffer can be made into a video or any other way to view a series of 3D points.

the operators commonly (around 80 percent of the time in one study [9]) switch to a different approach (rescue therapy). This rescue therapy is mostly done using mechanical thrombectomy with a different device, showcasing that operators believe that another device will work better for the same patient (i.e. they believe a different device works better even though the geometry of the patients vascular system is exactly the same). If they did not believe so, they would not try mechanical thrombectomy again, due to the associated risks. Thus we will work under the idea that choosing the right device depending on the patient is likely to improve the success rate of each attempt.

One of the most promising approaches that has been used to investigate the use of mechanical thrombectomy devices on different blood vessel geometries is the use of digital twins based on numerical physics simulations [10, 11, 12]. This approach involves creating a 3D model of a blood vessel, using the scan of a patient's vascular system (including the blood clot) and then using a numerical solver that incorporates known physical laws to simulate what would happen if a certain device was used to extract the blood clot. This simulation can then be used to analyze the likely *physical outcome* of the procedure. Importantly, this simulation is specific to the patient. However, there is a big disadvantage of using numerical physics simulations: they are very slow. Simulations with the accuracy required for the complex dynamics of blood clots and mechanical thrombectomy devices can take hours or even days on a computer cluster. In the clinical setting of acute ischemic stroke such time is not available since swift action (on the timescale of hours) is highly correlated with good *clinical outcome* [13]. This is exacerbated by the fact that the time in which the simulation could be performed, the time between diagnosis using imaging and the start of the thrombectomy procedure, is even shorter, around 20-60 minutes [14]. In this paper, we attempt to tackle this problem of slow numerical simulations by replacing them with a *surrogate model* that uses machine learning to predict the next timestep instead, which has the potential to be much faster and robust to noise, which has been done previously on different tasks [15, 16].

The current machine learning based surrogates that have been used in the setting of ischemic stroke (or very related settings) are end-to-end methods. They take in some starting position, such as a 3D model of a blood vessel, and directly predict some feature of the result. These models can still be seen as *surrogate models*, since they replace the numerical physics simulation. This has been done to directly predict the *clinical outcome*, with varying results [10][17]. Other approaches rather try to predict part of the *physical outcome*, such as the stresses on the blood clot or on the blood vessel [18]. While useful for assessing fragmentation risk, they give little insight into exactly what caused this risk. They show the user a likely outcome, but do not provide insight into how this outcome came to be. Additionally, they give little insight into how the procedure could be modified to achieve a better outcome. To overcome this, we intend to use a *surrogate model* of the numerical simulation in a step-by-step fashion, see figure 1.1.

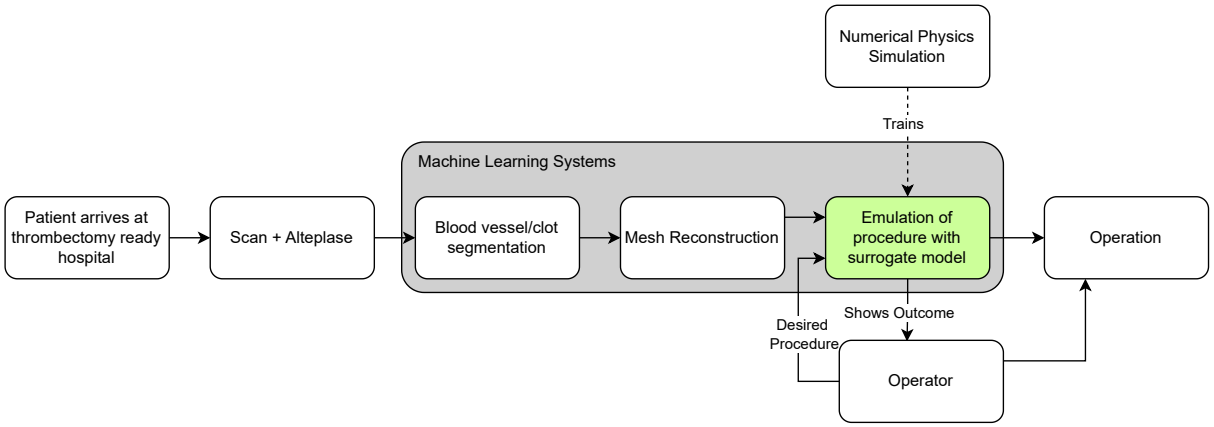



Figure 1.2: Overview of a possible pipeline of the use of digital twins in ischemic stroke treatment. Green represents the contributions of the current paper. Segmentation and reconstruction can currently be done by hand, but machine learning based models are being developed to automate this aspect.

This can be done in an iterative approach: the machine learning model is given the current state of the system and predicts the next one in a loop until some end time is reached. If our *surrogate models* accurately emulates the full numerical simulation then this allows us to view the simulation in the same manner as if it was fully made numerically, while benefiting from the speedup and the resistance to noise that machine learning methods bring.

Currently, several machine learning methods exist that are suited for a step-by-step approach, however they have to be adjusted to the specifics of mechanical thrombectomy simulations, which will be the main focus of this thesis. For an overview of how this research fits into the future of ischemic stroke treatment, see figure 1.2. The most promising machine learning methods, those that have been designed for a step-by-step approach and have been tested on datasets of physical systems, are either graph based or based on the attention mechanism. Graph-based methods include MeshGraphNet [19], a graph *artificial neural network* with residual connections. Attention based methods include the Transolver [20] and Erwin [21]. The main difference between these is how they handle inputs with large amounts of nodes and how their embeddings are calculated. We will investigate how these models compare in this setting, in terms of inference speed and in single-step and multi-step accuracy. Additionally, we will investigate methods for improving performance, such as generic data augmentations, but also data generation specific to 3D point cloud data. Namely, we will describe an algorithm for sampling new datapoints, by sampling on the triangles between three datapoints. Lastly, we will do two small ablation experiments, one to study the impact of data augmentation on generalizability, the other to study the impact of data-point size on speed and performance. This then gives us the following research questions:

- How well are MeshGraphNet, Transolver and Erwin suited as step-by-step surrogate models for numerical mechanical thrombectomy simulations?
  - How accurately do these models predict the next step of the numerical simulations?
  - How stable are the predictions of these models across longer timescales?
  - How fast is inference for these models compared to numerical solvers?
- What is the impact of data augmentations on single-step performance?
  - How does the addition of different amounts of generated datapoints to the training set impact performance?

  - How does rotating the input data affect performance?
  - How does increasing data-point size affect performance?
- How does rotating the input data affect generalizability to unseen geometries?

To investigate these claims we use 2 novel datasets provided by inSteps B.V. which both involve a blood clot moving through a blood vessel. These datasets are build using a traditional solver (Abaqus) for numerical physics and involve predicting the next timestep given the current one. The first dataset, called BendVessel, resembles a simplified aspiration procedure in which a force is applied to remove the blood clot from a simplified blood vessel. The second dataset, called ClotEntry, is similar to the first, but has a much higher resolution and uses realistic blood vessels. On these datasets we then train Meshgrapnet, Transolver and Erwin, adjusting the models and data as required. We investigate their performance on a single step and on a rollout and record their inference speed. Additionally, we investigate the use of various data augmentations and their effect on performance. Finally, we perform an experiment on generalizability to unseen geometries.

# Chapter 2

# Background

This chapter introduces the concepts from Medicine, Numerical Physics and AI required to understand the context of this thesis and its methods. First, we will cover the basics of Ischemic Stroke, focused on the aspects that are most relevant for modeling it in numerical simulations. Second, we will introduce numerical physics simulations, how they work and how they relate to Medicine and AI. Thirdly, we will describe the existing Machine Learning methods that are relevant to this thesis. We will cover both end-to-end methods previously used in the mechanical thrombectomy setting as well as methods suitable for a step-by-step approach. Lastly, we describe an algorithm for sampling new data from step-by-step pointcloud datapoints.

## 2.1 Ischemic stroke

Ischemic stroke results in the loss of blood flow to parts of the brain, resulting in damage to the affected tissue. In the following chapter we will describe in more detail the aspects of this process that are most relevant for our purposes. Namely, we will describe the blood vessels that commonly get blocked, the nature and composition of blood clots that block them and the types of devices used to remove blood clots.

### 2.1.1 Blood Vessels in the Brain

Blood Clots in Ischemic Stroke cases can be removed mechanically if they are located in large vessels, which will thus be the vessels we will focus on. These Large Vessel Occlusions (LVO) occur in around 33% of ischemic stroke cases and mostly (70%+ of LVO's) occur in either the middle cerebral artery (M1) and its branches or the intracranial carotid artery. [22]. These vessel are connected to the Circle of Willis, a circular system of blood vessels that can provide alternative routes for blood flow. Its shape is very variable [23], so much so that the amount of connections between blood vessels can differ from person to person. These variations can also influence the blood flow into the M1 and the intracranial carotid artery [24] and might thus be relevant for the interaction between blood clots in LVO's and the blood surrounding them. These blood vessels branch into many smaller vessels. Some branches, such as the M2 which branches off the M1, are large enough for mechanical thrombectomy, while many of these branches are too narrow for the current methods [25][26].

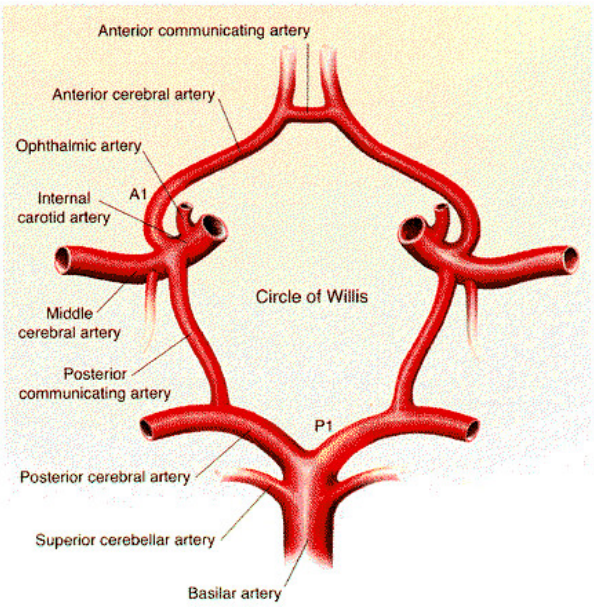



Figure 2.1: Schematic view of a typical example of the circle of Willis. The Middle Cerebral Artery (M1) and the Internal Carotid Artery (ICA), which is where most mechanical thrombectomy takes place, can be seen in the middle, on the left. Taken from [27]

For our purposes of modeling mechanical thrombectomy an important aspect of the blood vessels is their rigidity. During normal blood flow and during operations forces are applied to the blood vessel and change its shape. For example, when performing mechanical thrombectomy, the blood clot is pulled from its position while it is in contact with the blood vessel, creating friction. However, in all current 3D numerical physics simulations of mechanical thrombectomy it is assumed that these forces are insignificant compared to how rigid the blood vessels are [10][11]. This is based on the fact that the blood clots are much more elastic than the vessel walls (1-2 MPa vs. 10 kPa elastic modulus for the clot and vessel, respectively).

### 2.1.2 Blood Clots

Blood clots come in different sizes, shapes, and compositions which affect the their physical dynamics during treatment. We will discuss how these different factors impact the *physical outcome* of mechanical thrombectomy and what the limitations are of current simulations.

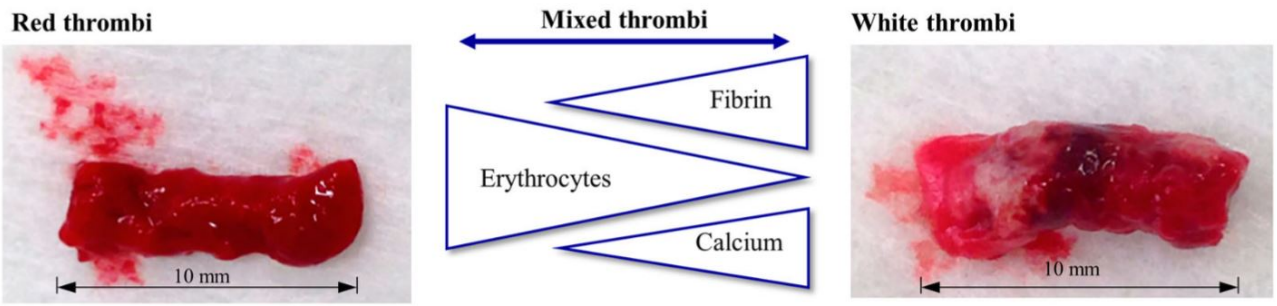



Figure 2.2: Different types of blood clots (thrombi). Calcium binds to the components of the blood clot and is more prevalent with increased platelets. The term Erythrocytes refers to red blood cells. Taken directly from: [28].

Blood clots mainly consist of fibrin, platelets, red blood cells, white blood cells, Von Willibrand Factor (a large protein present in the blood) and extracellular DNA [29]. All of these components occur in different ratios in different blood clots, causing them to be heterogeneous. Moreover, there is variation within singular blood clots, it is common that sections of a blood clot differ in their composition, for example a common distribution is a core high in red blood cell content and a shell with more fibrin, platelets and Von Willebrand Factor. [30]. The composition of a blood clot affects the likelihood of successful treatment using current mechanical thrombectomy procedures [29]. In general, blood clots with more fibrin are stiffer and more difficult to remove, in contrast with those with more red blood cells, which are the opposite. In addition, increased fibrin content increases the friction the blood clot experiences with the blood vessel walls and therefore how it behaves during mechanical thrombectomy. Lastly, increased red blood cell content increases the risk of fragmentation of the blood clot, which can result in other blood vessels getting blocked.

The size of blood clot certainly has some relation to the *physical outcome*, but the exact nature of this relation is still an area of active research [8]. Larger clots are easier to remove, but do not lead to better *clinical outcomes*. This indicates that larger clots have a bigger chance of fracturing.

### 2.1.3 Mechanical Thrombectomy Devices

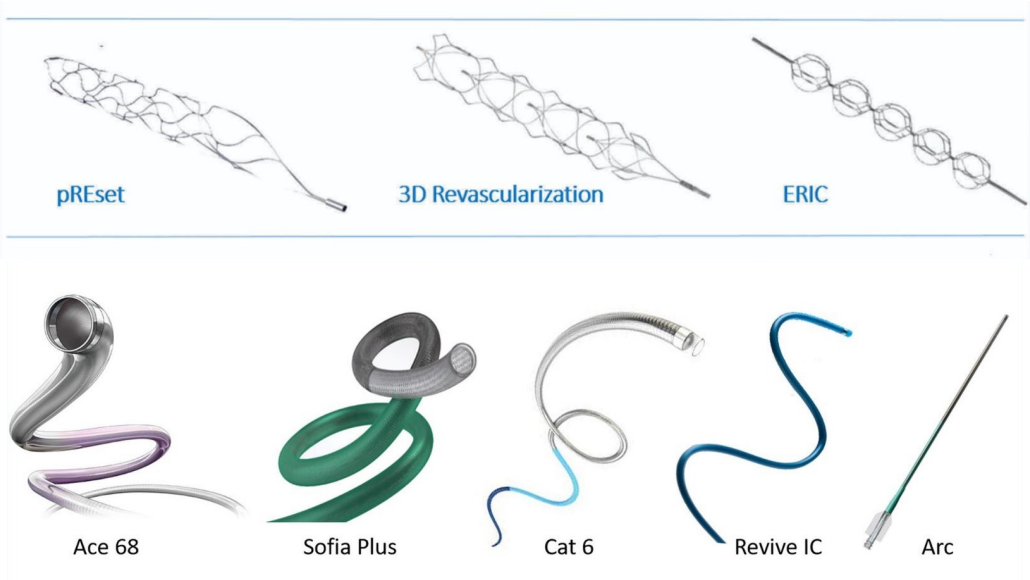



Figure 2.3: First picture: used stent retriever. Second picture: Example stent retriever and aspiration designs, adapted from [31].

Mechanical thrombectomy devices come in two main types, aspiration devices and stent retrievers (or a combination of both). These devices are delivered to the location of the blood clot by following a guide wire that is inserted into the bloodstream. For stent retrievers, a microcatheter is used to facilitate this delivery [32]. Many designs of these devices exist, varying in size and shape, for a comprehensive list see [32].

Aspiration devices function by applying a negative pressure to a hollow tube that is connected to the device. The operator does this by using a syringe manually or an electric pump. The blood clot is then sucked into device, either traveling through the tube and outside the body, or is stuck to tip and is removed along with the device. These devices vary in the design of their tips and in their size. Examples of aspiration devices include Penumbra Ace 68 [33] and Medtronic React 68 [34].

Stent retrievers consist of *metal meshes* designed to grab hold of the blood clot. They are delivered to the blood clot in a collapsed state inside a sheath, where they puncture the blood clot. Once inside they exit the sheath and expand. When expansion is complete the stent retriever is pulled, along with the blood clot. Stent retrievers come in many different designs where not only their size, but the entire structure of their mesh varies. Examples of stent retrievers include Stryker Trevo XP [35] and Terumo Neuro Eric [36].

## 2.2 Numerical Physics Simulations

Computer simulations of physical processes allow the inspection of and the experimentation with physical systems in ways that would be infeasible or expensive otherwise. They do this by applying known physical laws to the state of a system to find the state that follows it. This then allows for inspection of details or failures by viewing the sequence of states and for experimenting with designs by changing the initial state or the simulation parameters. The use of physics simulations in these ways dates back to the 1950's [37] and they have since been used in applications across many subfields of engineering and physics like in applied mechanics [38], rocket engine design [39] and nuclear physics [40], including in the intersection with the medical domain [41].

In this section we will describe how numerical physics simulations work and what decisions are required when building one after which we will describe their advantages and disadvantages in the medical domain. Finally, we will showcase the ways in which AI can be used to improve simulations and which challenges arise when doing so.

### 2.2.1 Defining a Simulation

In the simplest terms, a numerical physics simulation needs a start state at time $t = 0$: $s_0$ and a transition function $F(s_t)$ with gives the state at time $t + \Delta t$, where $\Delta t$ is some small timestep. Then $F$ can be applied iteratively to $s_0$ to produce an entire sequence of states. To make it clear how the simulations described in the method section came to be we will now discuss the process of defining a state and a transition function and what decisions need to be made along the way.

#### State

The form that the state takes must be carefully considered as it has a large impact on the simulation as a whole. For some systems defining the state is straightforward, a ball flying through the air could be modeled as an $x, y, z$ position and velocity (although even here, polar coordinates could be considered as an alternative). For systems with many moving elements, such as fluid simulation or the simulation of galaxy formation an important decision is if the state should be discretized or not. A common approach in fluid simulation is to divide up the space into squares or boxes and to only model the interaction between these while ignoring any interactions on a smaller scale. This allows for fast computation, but comes at the cost of accuracy. A related approach is to treat surfaces as a mesh, with points sampled on the surface that are connected based on distance. This approach has the advantage of the fast computation, but for the right problem it suffers less from the loss in accuracy than the box approach. This is because it discretizes only the points of interaction, but not space itself, as the points can still have real valued positions.

#### Transition function

Finding the transition function entails the following steps: finding the derivatives governing the system, deciding how to use those derivatives, filling out the other parameters in the derivatives and defining boundary conditions.

Firstly, when considering all the forces involved in a physical process we receive a set of equations that govern how the system behaves. These can take two forms, either an explicit form of a derivative of $s$ over time, or an implicit form of a system of equations that the current and future states must satisfy.

Secondly, we then need to take these sets of equations and compute the next state. The best way to do this is very problem dependent. For the explicit case, the simplest method, known as the forward Euler method looks as follows:

$$s_{t+\Delta t} = F(s_t) = s_t + \Delta t \frac{d}{dt} s_t. \tag{2.1}$$

It must be noted that in many systems forward Euler is a very unstable approach and it will often diverge from the correct values. Several other methods exist, like the Leap Frog method and a family of methods known as Runge-Kutta methods, which also incorporate higher order derivatives. For the implicit case, the backward Euler method involves solving the following equation:

$$s_{t+\Delta t} = s_t + \Delta t \frac{d}{d(t+\Delta t)} s_{t+\Delta t} \tag{2.2}$$

Thirdly, we need to fill in some parameters present in the sets of equations. Some generic parameters might already be known, such as the thermal expansion coefficient of water. However, others will have to be measured to achieve accurate results. These could include the exact geometry of the starting state or exact physical conditions like air pressure or friction coefficients. Additionally, the size of $\Delta t$ is very important, as the larger it is the faster the simulation can run but also the less stable it becomes. Lastly, boundary conditions can be the defined if the system has some limited size. Examples of such conditions are loops (the left boundary connects with the right boundary), hard walls and voids (everything inside is destroyed).

**An example system: Heat in a rod**

As an example we consider a system of heat spreading throughout a thin rod made of a homogeneous material and go through the steps described above. First, we define a state by splitting the rod up into N finite parts of size $\Delta x$ and assigning a temperature value to each, i.e. $u_1$ until $u_N$. In this case the physical laws dictate that heat spreads according to:

$$\frac{\partial u_j^t}{\partial \Delta t} = \alpha \frac{\partial^2 u_j^t}{\partial u_{j-1}^t \partial u_{j+1}^t}, \tag{2.3}$$

with constant $\alpha$ which using Taylor expansions can be rewritten as:

$$\frac{\alpha}{\Delta x^2}(u_{j+1}^t - 2u_j^t + u_{j-1}^t) \tag{2.4}$$

Which then gives us the following transition function using Forward Euler:

$$F(u_j^t) = u_j^t + \frac{\alpha \Delta t}{\Delta x^2}(u_{j+1}^t - 2u_j^t + u_{j-1}^t) \tag{2.5}$$

Now we should replace our $\alpha$ with the correct constant, which in this case would be a combination of the density, heat conductivity and thermal mass of the material. The starting state could now also be determined, for example, the rod could be heated on one side at beginning of the simulation. Lastly, boundary conditions need to be specified as the transition function is otherwise undefined near the edges of the rod as $u_{j-1}$ or $u_{j+1}$ do not exist there. A common approach would be to assign $F(u_1^t) = F(u_N^t) = 0$.

### 2.2.2 Simulations in the Medical Domain

In the medical domain the advantages of simulation (possibly in addition to real experiments) are apparent. In this domain it is often not possible to perform experiments directly due to

medical and ethical concerns. Furthermore, collecting high quality scans to study the dynamics of an ongoing procedure is difficult if that procedure must be completed quickly (such as with ischemic stroke). For example, it is difficult to do so when studying how a stent retriever unfolds in a blood clot, as that likely requires pausing the mechanical thrombectomy procedure.

Lastly, there are also processes that can be studied outside the body using real world physical models, called *phantoms* (e.g., an organ taken from a donor, or a fake organ that has similar properties to the real one) However, these *phantoms* are often limited in quantity (in case of real organs) or very labor-intensive to produce [42] (in case of fake organs). In contrast, physical simulations do not have these disadvantages. They allow for endless experimentation and may involve patients only in minimal ways, such as using the patients physical characteristics as parameters (such as blood pressure or brain structure), which often need to be recorded regardless. However, the costs associated with performing accurate numerical simulations can be high, as the more accurate a simulation needs to be the more computing power is required. One other disadvantage specific to the medical domain is the calibration of model parameters to the real world. For this, actual data needs to be collected on factors like blood viscosity, coefficients of frictions, etc. These are not easy to collect, but fortunately they only need to be collected once and some studies have already done so [43].

### 2.2.3 Physical simulations & AI

Even though the knowledge of the physical laws and enough compute is in theory enough to simulate any physical system, AI can be a useful tool for physical simulations to achieve results otherwise infeasible. For our purposes, it is the most interesting to discuss how machine learning can speed up simulations. However other possible beneficial approaches exist, such as symmetry finding [44] or generating new examples of stable physical systems such as molecules [45]. Before discussing how simulation speed can be increased we will explain how a machine learning model can learn a set of equations in the first place.

#### Approximating a Transition function using Machine Learning

In the explicit case it is straightforward how a sufficiently expressive machine learning model can approximate a set of equations, as $F(s_t)$ can be approximated directly by training a model to predict $s_{t+\Delta t}$ from $s_t$ sampled from previously done simulations. In the implicit case where there is no clear $F(s_t)$, this is less obvious, but as it turns out we can in fact use the exact same approach. This is because by the implicit value theorem, for any set of equations (that follow certain smoothness and differentiability constraints) there exists some function that directly computes the variables (the next state in this case). This function often does not have an analytical solution, but since our machine learning model is capable of approximating any continuous function by the universal approximation theorem this does not pose an issue in practice. Thus, by training to predict $s_{t+\Delta t}$ from $s_t$ we are approximating a function $F'(s_t)$ which is implicitly defined by the set of equations governing the system.

#### Speed and Robustness Advantages of Machine Learning

As we have seen before, for numerical simulations there are tradeoffs between computational speed and accuracy. However, in many cases it is possible to do slow and accurate simulations in advance, which can be taken advantage of. This can be done by using machine learning, which can approximate the $F(s_t)$ function by using these accurate simulations as training data ahead of time. Then when speed is needed the machine learning models only do inference. This comes with a large speedup as in general machine learning methods are much faster than numerical simulations. Since they do not suffer as much from the need for high resolutions and small

timesteps for stable results. Moreover, many machine learning methods are highly optimized for speed and can perform most of their calculations in parallel on the GPU, something that is still not widely available for numerical physics simulations.

These machine learning models can thus provide a faster way of emulating these physical processes after training is done, for example in our case a 12 hour thrombectomy simulation [10] would be too slow for clinical purposes, but a neural network trained on those simulations could be fast enough to be practical. Papers that have tried this in similar domains have already achieved a 10-1000 times speed up with minimal loss in accuracy [46, 47, 48]. The fact that the speedup factors take values is such a large range indicates that some tasks are much more difficult emulate than others.

Another benefit of using machine learning models is that they are also more suited to dealing with noisy input data, such as scans of a patients vascular system. This is in contrast to numerical physics simulations, for which great care must be taken to make sure the starting *mathematical meshes* are of the correct resolution and contain no mistakes in their topology. Machine learning methods can be trained for robustness, by incorporating such imperfections in the training data.

#### Challenges of Mesh Data

As we discussed, discretization using meshes has advantages for simulation accuracy, however it also has its challenges when it comes to building a machine learning *surrogate model*. Firstly, the amount of nodes used in some simulations is very large, 10.000-100.000+, each of which can possibly interact with every other node. This means that any machine learning method must either be able to handle such large inputs in terms of speed and memory efficiency, or it must be able to work with a sparser down-sampled version. The former can be a major issue for some model architectures like the transformer, which can have its memory cost scale quadratically with the input size. The latter is something that machine learning can deal with as discussed before, but it always comes at some loss of accuracy. Secondly, the exact structure of the mesh can have a large impact on how difficult training is. Some simulations add more mesh nodes in areas were the numerical solver needs more detail. These changes can also impact the performance of the machine learning model that is trained on that data as this means the nodes have a varying density, which the model must be able to handle.

## 2.3 Existing Machine Learning Methods

The methods explained in this section are those that are likely to be the most relevant for our use-case. They are either specifically made for use in clinical settings related to ours or have been designed for approximating numerical physics simulations involving meshes. To facilitate our explanations we will first cover two techniques that are present in many of the methods we will cover.

#### The Transformer model

The transformer [50] is an architecture that uses self-attention in combination with Multi-Layer-Perceptrons (MLP), normalization and skip connections. Its main feature, self-attention, works as follows. For each input of the self-attention block it calculates a query $Q$, a key $K$ and a value $V$, each represented as a vector and each calculated using a different linear projection. Each input is then compared to every other input (including itself) in the following way. First, for input i, the value $Q_i^T K_j$ is calculated for every $j$. The softmax is then taken over these values resulting in attention weights that add to one, these weights represent the relevancy of

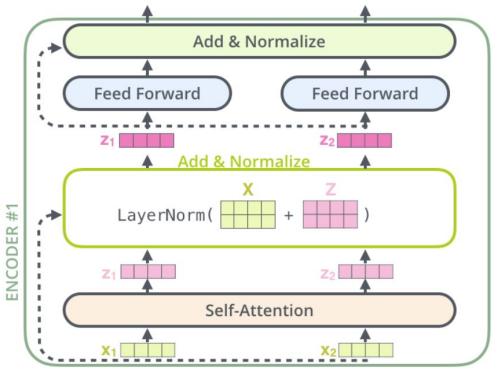



Figure 2.4: Basic Transformer model. Image shows two inputs. Taken from [49].

input $j$ to the output of input $i$. Lastly the attention weights are multiplied by their respective values, i.e. $V$ and summed, resulting in the output for $i$. Putting these steps together gives us the following for output $Z$:

$$Z_i = \sum_j \text{softmax}(Q_i^T K_j) V_j \tag{2.6}$$

or in matrix form, where $Q$, $K$, $V$ and $Z$ are stacks of horizontal vectors:

$$Z = \text{softmax}(QK^T)V \tag{2.7}$$

This then gives us an embedding for each input of the self-attention layer. In most applications, self-attention is done multiple times in parallel, called multi-head self-attention. Each head has its own linear weights for calculating the queries, keys and values and performs self-attention separately. The outputs of each of these heads is then combined with an MLP.

Incorporating attention, the entire structure of a Transformer block then looks like the following, consisting of two parts. The first part starts with multi-head self-attention on the input of the block. Then, using a skip-connection, the input of the transformer block is added to each of these values. Finally a layernorm is applied. The second part takes the output of the first part and applies an MLP to each value separately. Through a skip connection the outputs of the MLP are then connected to the inputs of the second part. Finally layernorm is applied once again. For an illustration see figure 2.4.

### Message Passing Neural Network

The Message Passing Neural Network (MPNN) [51] is an architecture designed to specifically work on graphs. To use an MPNN, a graph must be defined by a list of nodes and a list of edges between these nodes. Each node and each edge has a feature vector $v_i$ for node $i$ and $e_{ij}$ for a (possibly directed) edge between node $i$ and $j$. For both the nodes and the edges some starting value must be assigned to their feature vector, which can come directly from the data, or be replaced by some arbitrary value if the data does not contain them.

An MPNN works using a series of message passing blocks, which function as follows. A message passing block works in three steps. In the first step, which is optional, the edges are updated using an MLP $f^E$:

$$e'_{ij} = f^E(e_{ij}, v_i, v_j). \tag{2.8}$$

In the second step messages are calculated, using a different MLP $f^M$:

$$m_{ij} = f^M(v_i, v_j, e'_{ij}). \tag{2.9}$$

Finally, for each node the messages to each node are aggregated and its information is updated using a third MLP $f^V$ and an aggregation function $\sigma$:

$$v'_i = f^V(v_i, \sigma_j^J(m_{ij})). \tag{2.10}$$

With the sum commonly used as an aggregation function. As can be seen, each message passing layer the information of a node is spread to its neighbors, in other words it increases the receptive field size by 1 for each layer. At the final layer, feature vectors can then be used for downstream tasks, such as edge prediction, node classification or graph classification (by aggregating information from all nodes).

### 2.3.1 End-to-end Approaches

Before we dive into the methods that can provide a transition function based on simulations we must first discuss some that do not. These methods take an end-to-end approach and take only the geometry at $t = 0$ and from that directly predict the physical or clinical outcome. Unfortunately, this end-to-end approach results in a loss of insight into the dynamics of every step between the beginning and the end. At most, these methods can offer insight into the aspects of the input that were relevant to the predicted outcome. For example, such a method could predict that a blood clot will fracture and highlight what part of the geometry it used to make that prediction. While useful, not being able to see how exactly the clot fractured will make it more difficult to adjust the operation such that the fracture would not occur. However, while less insightful, end-to-end methods also have the major advantage of having a much simpler task to train on. If such models are accurate and general enough, their increased performance could outweigh their downside.

#### Handcrafted Geometric Features

Various quantifiable geometric factors have been discovered that have a simple impact on the *physical outcome* of mechanical thrombectomy, such as the length of the clot, how much of the blood vessel is blocked by the clot, the total volume of the clot and the general area of the brains blood vessel system the clot is located [8]. These relations however do not account for the fact that the brain's vascular system varies greatly among people and as such have only a small amount of predictive power.

Some approaches take it one step further and try to select geometric features that are more granular. In [10] the author performed very detailed simulations that involved stent retrievers removing blood clots from vessels with realistic geometries, each with slightly different starting variables. These simulations were then placed into 4 classes based on their likely *clinical outcome* according to the TIMI grade (see Table 1 in [52]). For each simulation, the starting position, clot type and clot length were recorded. These were then used for fitting various ML models to predict the TIMI grade, namely a small MLP, Naïve Bayes, Descision Trees and Random Forests. The results of these methods were however not good enough to be clinically useful, reaching accuracies of around 70 percent.

#### Principal Component Analysis and Neural Networks

In [46] the authors use simulations [18] of the stress on a blood vessel, the aorta, to train a system of neural networks. In these simulations the blood vessels were divided up into small elements. The pressure inside the blood vessel was increased until at least one element of the blood vessel reached a critical stress value, which in the real world would result in a rupture. A snapshot of this final step was then taken as a datapoint, with the shape of the blood vessel as the input and the stresses on each element as the desired output. This process was then done for each digital aorta model that was available. This dataset was then used to train a machine learning system that predicts the output given the input using three subsystems. The first compresses information about the shape using principal component analysis combined with a linear layer, the second uses a 3 layer neural network to map these values to a set number of

stress values and the third which decodes these stress values using one transposed convolutional layer. These subsystems are all trained individually and then combined into an entire end to end model.

With this method the authors achieved results that came very close to the simulated results, reaching less than 1% deviation from the numerical physics simulation when measuring the Von Mises stress (a measure of stress that combines different types of stress). The speed up of their method was around 1800 times compared to the full numerical physics simulation.

### 2.3.2 Transolver

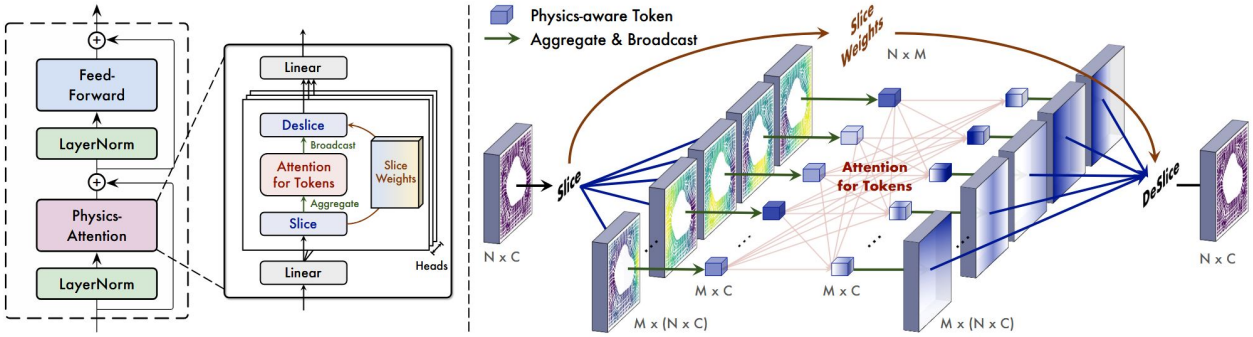



Figure 2.5: Overview of the Transolver architecture. On the left the Transolver blocks can be seen, which is a modified version of the transformer architecture. On the right the PhysicsAttention mechanism is shown in more detail for a single attention head. Taken from: [20].

The Transolver [20] is a method that predicts features of each input node and is based on the transformer architecture. Its core idea is to split the input nodes into slices based on physical similarity and then performing attention between slices. This allows for the attention mechanism to scale with the amount of slices instead of being performed between every node.

The Transolver method on meshes works as follows. First each nodes information, a concatenation of its geometric information and its other properties (for example velocity, pressure etc), is processed with a small MLP. Then the nodes go through a set of Transolver blocks, which special feature is the Physics Attention block which replaces normal attention.

These Physics attention blocks work as follows. At the start of each of these blocks each input node is put through two separate linear layers, the first computes weights using a softmax operation, which indicates how much a node belongs to each slice of each head. This results in weights $w_{ijk}$ for each input $i \in N$, attention head $j \in H$ and slice $k \in M$, where $\sum_m^M w_{ijm} = 1$ for all $i$ and $j$. The second linear layer simply projects an input node to the embedding dimensions of each slice of each head. For each slice and each head all nodes are then aggregated into one feature using a normalized weighted average in the following way:

$$z_{jk} = \frac{\sum_n^N w_{njk} x_n}{\sum_n^N w_{njk}} \tag{2.11}$$

For each head attention is done on this aggregated information between all slices, the output of this we will call $z'_{jk}$. As a second to last step of the Physics Attention block, for each head the nodes are reconstructed using a deslicing operation:

$$x'_{ij} = \sum_m^M w_{ijm} z'_{jm} \tag{2.12}$$

Finally, following the multi-head attention paradigm, the outputs for each node are aggregated across attention heads using a linear layer.

These Transolver blocks then consist of a LayerNorm operation, a Physics Attention block, another Layernorm operation and finally a small MLP. Using the same structure of LayerNorms and skip connections as the normal Transformer. After applying a number Transolver blocks a linear layer projects the information of each node to the correct output dimension.

### 2.3.3 Erwin

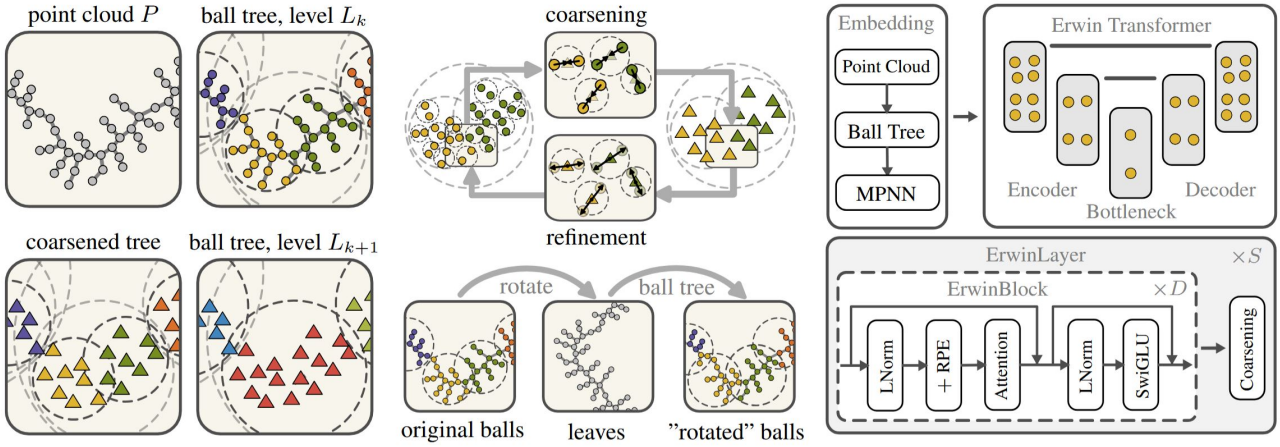



Figure 2.6: Overview of the Erwin Architecture. On the left the construction of the nodes into a balltree can be seen. In the top image in the middle, an example coarsening and refinement operation is displayed. The image on the bottom in the middle shows how the rotation changes which ball contains which nodes when constructing the ball tree. In the top right the general U-shape of the architecture can be seen. In the bottom right a BallProcessing block is displayed (referred to as an ErwinLayer in the image). Taken from: [21]

Erwin [21] is a hierarchical transformer based model that performs attention within subsets of the input, combines those subsets and then repeats this process. These subsets take the form of balls, which have a center and a radius and which divide the nodes into disjoint subsets spatially. It decides which nodes belong to which ball at each level by constructing a ball-tree, which recursively divides the nodes between balls of equal size. At the first level self-attention is performed within balls of input points and the output of this results in a singular feature vector for each ball. On subsequent levels these singular feature vectors are then divided into balls again (using the centers of the previous level as positions) and the process repeats. Erwin employs some additional tricks to improve performance. Firstly, it can do additional coarsening, which can combine multiple feature vectors using a linear projection, resulting in a bigger receptive field. Secondly, it constructs the ball-tree in an efficient way such that it forms a perfect binary tree which allows for fast indexing. Thirdly, which nodes fall into which ball is slightly different every other layer which is achieved by rotation, resulting in more cross-ball information exchange. Lastly, it uses a learned embedding in the form of a Message Passing Neural Network (MPNN), see section 2.3.

In detail, Erwin works as follows. First, we will explain the BallProcessing block as it is used throughout the architecture. In each BallProcessing block for each ball a positional encoding is applied according to the following formula:

$$X_{Bi} = X_{Bi} + (P_{Bi} + c_B)W_{pos} \tag{2.13}$$

**Example.** *Let* $P = \{\mathbf{p}_1, ..., \mathbf{p}_8\}$*, then a ball tree* $T = \{L_0, L_1, L_2, L_3\}$ *is stored after the permutation* $\pi$ *as*

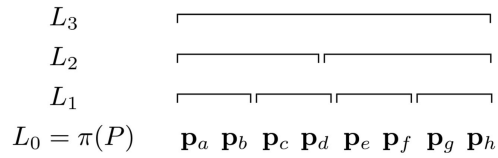



Figure 2.7: Example balltree formation. $P$ is the set of nodes and $T$ is the collection of layers of the tree. The permutation $\pi()$ could for example be a rotation as described below. Taken directly from [21].

where $X_{Bi}$ is the feature vector of node $i$ in ball $B$ and $P_{Bi}$ its position, $C_B$ the center of the ball and $W_{pos}$ a learnable projection. Then a number of BallAttention blocks is applied, which compute mulit-head self-attention and then use a Swiglu MLP [53] [54] using Root Mean Square Normalization [55] and skip connections. Additionally these BallAttention blocks use a distance fall-off that is applied to the attention values according to:

$$D_{Bij} = \sigma^2 ||P_{Bi} - P_{Bj}||_2 \tag{2.14}$$

with $\sigma$ a learnable parameter and $i \neq j$. After the BallAttention blocks an optional coarsening or refinement operation is applied. The coarsening operation selects an amount of nodes equal to the stride size in the form of a ball, concatenates their features and projects them resulting in a decrease in the amount of nodes and an increase in receptive field size. It does this according to:

$$x_{B'} = \text{concat}_{i \in B}(\text{concat}(X_{Bi}, P_{Bi} - c_B))W_c \tag{2.15}$$

The refinement operation works by projecting a singular point back up to all the points in a ball as such: For all nodes $i \in B'$ do

$$X_{B'i} = \text{concat}(X_{Bi}, P_{Bi})W_r \tag{2.16}$$

The BallProcessing blocks are made more efficient by the underlying binary tree the ball structure is stored in. The construction of this tree is done recursively starting from the complete set of nodes. At each step the current level is split into two sets of identical size, introducing virtual nodes when necessary to keep the sizes the same. This split is made along the axis (x, y or z) with the largest difference between the highest and the lowest value and is split according to the median of that axis. Doing this allows for two things, firstly it allows for fast indexing by reshaping the nodes using tensor operations. For example, reshaping a tensor of 32 nodes into a $[8, 4]$ tensor gives eight balls of size 4. Secondly, since the ball tree is constructed according to the axis it is possible to construct a distinct ball tree by rotating the input 45 degrees. This results in the nodes falling into different balls, altering which nodes interact with the attention mechanism. In the architecture the alternative ball tree is used every other BallProcessing block, to increase cross-ball connections.

The complete architecture then looks like the following. Firstly, it embeds the nodes with a MPNN with mean pooling using the provided node features and the pairwise distances as edge features. Secondly, the main encoder is applied, which consists of BallProcessing blocks that each use the coarsening operation. Thirdly, a bottleneck is applied, this is a single BallProcessing block, without any coarsening or refinement. Lasty, the decoder is applied, similarly to the encoder but now with refinement operations. Crucially, the encoder and the decoder are connected with skip connections in a U-net [56] like fashion (although with addition instead of the concatenation of U-net), with each coarsened level of the encoder connected to its corresponding level of the decoder. As such the amount of BallProcessing blocks in each must be the same.

### 2.3.4 MeshGraphNet

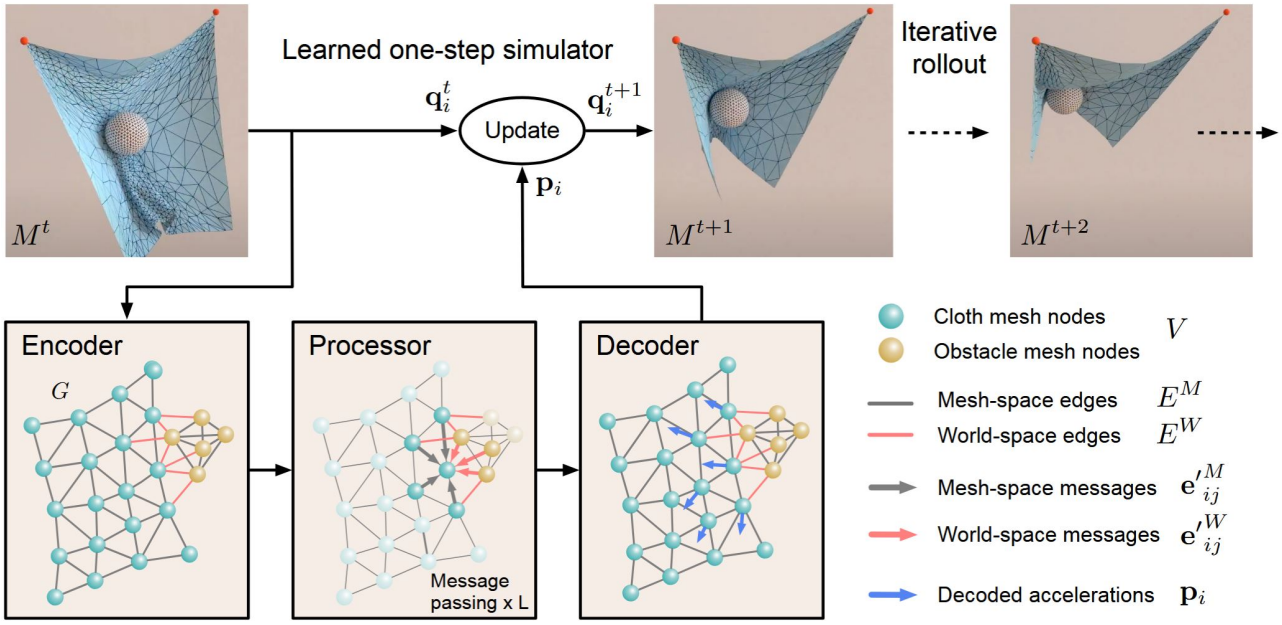



Figure 2.8: Overview of the architecture of MeshGraphNet. As can be seen it uses an encoder, followed my message passing layer and a decoder. Taken from: [19]

.

MeshGraphNet [19] is a graph neural network [57] that introduces some additional techniques, like processing mesh and world edges separately, to work well on *mathematical meshes*. MeshGraphNet then works as follows. Firstly, it encodes the edge and node information separately using two MLP's, which gives an embedding for every edge i.e. $e_{ij}$ for an an edge between node $i$ and $j$ and an embedding for every node i.e. $v_i$ for node $i$. Secondly, it does a set of message passing layers, as explained in section 2.3, each of which uses three separate MLP's, two for the edges and one for the nodes. In the paper, they describe that they differentiate between mesh edges, which represent a physical link between two nodes, and world edges, which represent that two nodes may interact due to physical proximity. This then gives us the following, where $f^{EM}$ and $f^{EW}$ are the edge MLP's for mesh edges and world edges respectively and $f^N$ is the node MLP, both using a residual connection:

$$e'^M_{ij} = f^{EM}(e^M_{ij}, v_i, v_j), \tag{2.17}$$

$$e'^W_{ij} = f^{EW}(e^W_{ij}, v_i, v_j), \tag{2.18}$$

$$v'_i = f^N(v_i, \sum_j e'_{ij}). \tag{2.19}$$

As can be seen it uses the sum operator to aggregate the edge information when updating the nodes, summing over both world and mesh edges. Furthermore, no explicit message is calculated. Instead the newly calculated edge feature vector is simply used instead. Lastly, an MLP is used to decode the node features to the relevant number of output dimensions.

In addition to this architecture, MeshGraphNet uses two additional techniques. Firstly, it creates edges between nodes that are physically close, i.e. if $|x_i - x_j| < r$ for some range $r$ so as to model interactions between nodes that might be far removed from each other in the mesh. Secondly, at every timestep it recalculates where on the surface of the object the resolution needs to be high and where it does not. It then adds and removes nodes in certain areas where needed. It does this by using a domain specific remesher during training and learning

parameters for a generic remesher from it for use during inference. Importantly, this technique is completely depended on the availability of such a remesher specific to the dataset and can therefore not always be used.

## 2.4 Bary Centric Coordinate Sampling

In this section we describe a method for sampling new pointcloud datapoints from existing pointclouds that evolve over time, provided that the order and the amount of nodes is consistent. The main idea of the method is to pick a triangle and sample a point on it in such a way that the relative position of that point on the triangle stays the same across timesteps. This then allows a machine learning model to predict the next location of the point, as it moves consistently and with similar dynamics as the nodes that form the triangle.

To sample a point on a triangle so called barycentric coordinates are used, which relate the point to the three vertices of the triangle. Specifically, with barycentric coordinates $\lambda_1$, $\lambda_2$ and $\lambda_3$ the position of a point $p_s$ in the Cartesian coordinate system becomes

$$p_s = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3 \tag{2.20}$$

with $p_1$ to $p_3$ representing the coordinates of the vertices of the triangle and with all positions represented as vectors. If we add the constraint that for each barycentric coordinate $\lambda \in [0, 1]$ and that they sum to 1 then our barycentric coordinates define a unique point that lays on the triangle.

We can then sample barycentric coordinates with respect to this constraint to sample a point on the triangle. If we then keep the barycentric coordinates stationary across time, in the Cartesian coordinate system, it moves on the triangle proportionally to the movement of the vertices, as desired.

To use this approach to create a new datapoint we must sample points on many triangles across two timesteps, the first step as input for the model and the second to calculate the loss. We provide the pseudocode in algorithm 1.

**Algorithm 1** BaryCentricSampling

**Require:**

$\mathcal{N}_1$ : set of node positions at timestep 1
$\mathcal{N}_2$ : set of node positions at timestep 2
$\mathcal{E}$ : set of edges between nodes in $\mathcal{N}_1$ ▷ Ex. Constructed based on distance
$n$ : amount of nodes to sample

**Ensure:**

$\mathcal{P}_1$ : sampled point cloud at timestep 1
$\mathcal{P}_2$ : sampled point cloud at timestep 2
$\mathcal{T} \leftarrow \textsc{FindTriangles}(\mathcal{E})$
$\mathcal{P}_1 \leftarrow [\,]$
$\mathcal{P}_2 \leftarrow [\,]$
**for** 1 to $n$ **do**
    Sample $(c_1, c_2, c_3)$ from $[0, 1]$ such that $c_1 + c_2 + c_3 = 1$
    Sample triangle $t \in \mathcal{T}$
    Let $(v_1, v_2, v_3)$ be the vertices of $t$
    $\mathbf{p}_1 \leftarrow c_1\mathcal{N}_1[v_1] + c_2\mathcal{N}_1[v_2] + c_3\mathcal{N}_1[v_3]$
    $\mathbf{p}_2 \leftarrow c_1\mathcal{N}_2[v_1] + c_2\mathcal{N}_2[v_2] + c_3\mathcal{N}_2[v_3]$
    Append $\mathbf{p}_1$ to $\mathcal{P}_1$
    Append $\mathbf{p}_2$ to $\mathcal{P}_2$
**end for**
    **return** $\mathcal{P}_1, \mathcal{P}_2$

# Chapter 3

# Method

This chapter describes our experiments to evaluate machine learning based step-by-step surrogate models for numerical mechanical thrombectomy simulations. First, we describe the datasets used, including one dataset with simple and one with complex geometry. Second, we describe how we configured Erwin, Transolver and MeshGraphNet to for these tasks, followed by the training procedure used. Finally, we present the experiments that will answer our research questions.

## 3.1 Datasets

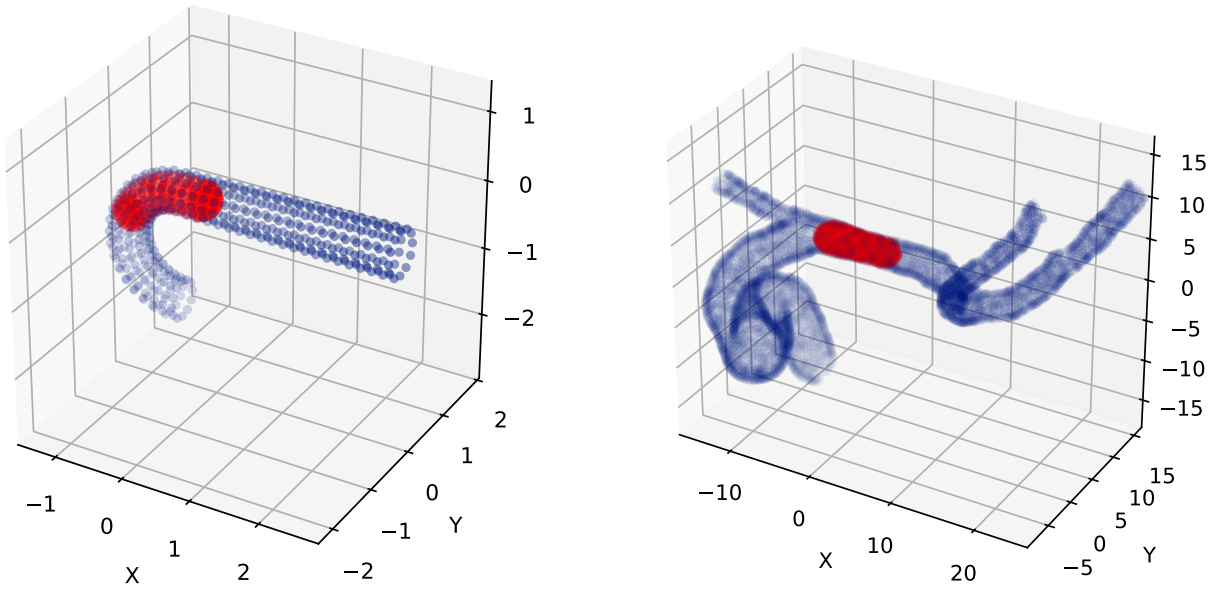
The datasets presented here were build by inSteps B.V. using the implicit finite element solver Abaqus from Dassault Systèmes. The hardware this was done with was an AMD Ryzen 7900 X CPU using 4 cores. The simulations were finetuned and checked by hand to ensure their stability and to ensure the blood clots were of the appropriate size to get stuck in one of the blood vessels. These simulations were then turned into datapoints such that the input of each datapoint consists of the current node positions, the previous node positions and a one hot encoding of the node types (representing either blood clot or blood vessel), with the task being to predict the node positions in the next timestep. For both datasets simulations were done from multiple initial conditions to increase the variation of the data. After simulation, each dataset was scaled to $[-1, 1]$ to provide a consistent scale.

### 3.1.1 BendVessel

BendVessel consists of a simplified blood vessel with a bend in it. In this bend a blood clot is placed and pushed out by applying a force, to emulate an aspiration device pulling it out. For this dataset, 20 different simulations were run each slightly rotated (from 0-180 degrees around one axis) and with slightly different initial conditions. Each simulation contains 977 nodes and each simulation lasts 40 timesteps. This dataset resembles a smaller data regime as it only has 800 datapoints and each datapoint has a small number of nodes. In our datasets, we consider an 'average' blood clot with a homogeneous composition throughout and of an average size. The average simulation time for this dataset was 3 minutes.

### 3.1.2 ClotEntry

ClotEntry consists of 10 realistic synthetic blood vessels, taken from [58] which generated these using a machine learning model that was trained on real blood vessels. In each of these vessels a blood clot is placed and a force is applied to it, in the same manner as for BendVessel. Each simulation begins with a stuck blood clot and ends when the blood clot has fully exited the




(a) (b)

Figure 3.1: Frame from BendVessel (left) and ClotEntry (right). The blood clot is shown in red and the vessel walls in blue. Both frames show a blood clot stuck in the blood vessel.

entire blood vessel, which is around 10 times the length of the clot. As such, a large part of the dataset involves the blood clot moving dynamically around the blood vessel. ClotEntry consists of 10 simulations each with 200 timesteps and with around 27000 nodes. This dataset represents an intermediate data regime, as there are 2000 datapoints and each datapoint is large enough to challenge current machine learning architectures when it comes to memory efficiency and speed. The average simulation time for this dataset was 25 minutes.

## 3.2 Machine Learning Models

Since MeshGraphNet and Erwin require edges between nodes to function we generate edges between nodes at runtime based on pairwise distance as we assume we are not given any (just as we would need to do when simply given a scan of a patients vascular system). We do this by connecting nodes within a radius $r$, i.e. if $|x_i - x_j| < r$. For each model a hyperparameter search was done using the Bayesian method, including searching for the optimal $r$. Importantly, this search was done once on the ClotEntry dataset using $\pi$ radians of rotations on the training data and 2000 nodes. These hyperparameters were then used for every experiment. To see the parameters used in the final tests see Appendix A.

### 3.2.1 Erwin

Erwin was used as provided by the official Github implementation, with a small MLP on top to project down to the correct dimension. The hyperparameter search was done on a smaller subdomain of the total possibilities of the hyperparameters, as the amount of combinations grows rapidly with the amount of BallProcessing layers. Instead, only the depth (i.e. the amount of BallAttention blocks) and the head count of the first BallProcessing layer of the encoder and of the decoder were directly searched over. The deeper layers of the encoder and decoder were made by simply adding or subtracting a hyperparameter value from the previous layer, massively reducing the search space. For example if the first BallProcessing layer of the

encoder had depth 10 and the hyperparameter value was 2 then the second layer would have depth 12, the third depth 14 and so on. Attention head count was kept the same for each BallProcessing layer (for the encoder and the decoder separately).

### 3.2.2 Transolver

The main part of the Transolver model was taken from the Nvidea PhysicsNemo library [59]. By default, Transolver requires a spatial embedding, but the paper does not provide any guidance on how to define and calculate it. To provide one, we used the same MPNN based embedding as Erwin. Using this learned embedding ensures our results were not skewed by the selection of the wrong spatial features (which must be picked by hand otherwise).

### 3.2.3 MeshGraphNet

The complete MeshGraphNet model was taken from the Nvidea PhysicsNemo library [59]. Just as in the paper [19] we use edge information consisting of two parts: the vector between the nodes i.e. $x_i - x_j$ if there is an edge between node $i$ and node $j$, where $x$ represents the 3D position of a node and the norm of that vector i.e. $|x_i - x_j|$. It must be noted that in the code of the MeshGraphNet paper and that of the NVIDIA implementation the world and mesh edges are treated as the same, are concatenated and use a single MLP. Here, we will also do so for consistency. To adapt MeshGraphNet to our datasets, we gave our point cloud data in the form of a graph as input to the model using the edges we constructed.

## 3.3 Training Parameters

For each dataset we train each model on transitions from $s_t$ to $s_{t+\Delta t}$ from a subset of the simulations, leaving all transitions from one simulation as a test set (and two others for validation purposes). Training was done using an A100 Nvidea GPU with 40 GB of VRAM. This slightly older hardware was chosen to provide a scenario that is more realistic to the types of hardware hospitals might have access to. Training is done using the AdamW optimizer [60] using a learning rate determined by the hyperparameter search for 300 epochs. This learning rate is scaled according to a warmup of 10 epochs, followed by cosine decay, see figure A.1 for a visualization.

Before predicting the next timestep, an approximation of the node velocities was calculated with $v_t = s_t - s_{t-\Delta t}$, which was then given to the model in addition to $s_t$. Models were evaluated according to the Mean Square Error (MSE):

$$\text{MSE} = \frac{1}{3|C|} \sum_{i}^{C} \sum_{j}^{\{x,y,z\}} ((\hat{s}_{t+\Delta t})_{ij} - (s_{t+\Delta t})_{ij})^2 \tag{3.1}$$

where $\hat{s}_{t+\Delta t}$ is the predicted next step, $C$ is the list of indexes of blood clot points and the state $s$ consists of a matrix of stacked vectors of the node positions. As can be seen, the loss is masked so that only the loss that comes from predicting the blood clot is counted, as in our simulations the vessel walls are assumed to be rigid (as discussed in section 2.1.1).

For ClotEntry, we sampled a subset of size 2048 out of the available 27000 nodes each time we trained on a sample. This allows for much faster inference and training, while still giving an insightful visual result. For BendVessel, we used all available nodes.

## 3.4 Experiments

### 3.4.1 Data Augmentations

To investigate the impact of rotational data augmentations, datapoints from both datasets were randomly rotated in all 3 dimensions up to a maximum angle. For each model, we measured the loss for a range of maximum angle values. For BendVessel, we used BaryCentricSampling and measured the loss for different amounts of generated data. For each of these experiments, we used the optimal values found for further experiments, to facilitate this we evaluate our models on the validation set. Furthermore, we also performed an experiment on the ClotEntry dataset where we changed the amount of sampled nodes to a range of $[500-6000]$ to investigate how well each of our models scales with the node count.

### 3.4.2 Performance on the Test Set and multi-step rollout

Finally, we combined all our best data augmentations and compared the single-step performance of our three models over 5 different seeds. To evaluate performance we showed the loss on the test set when predicting the next timestep. Additionally, we perform multi-timestep rollouts of a single seed on the training data to show reliability in longer time domains. To make these rollouts more reliable, the first steps of each simulation were not used as they involve the blood clot being partially outside the entrance to the blood vessel. We cut the first 40 timesteps for ClotEntry and the first 5 for BendVessel. Additionally, during rollout we collect inference speed data.

### 3.4.3 Generalization on unseen Geometries

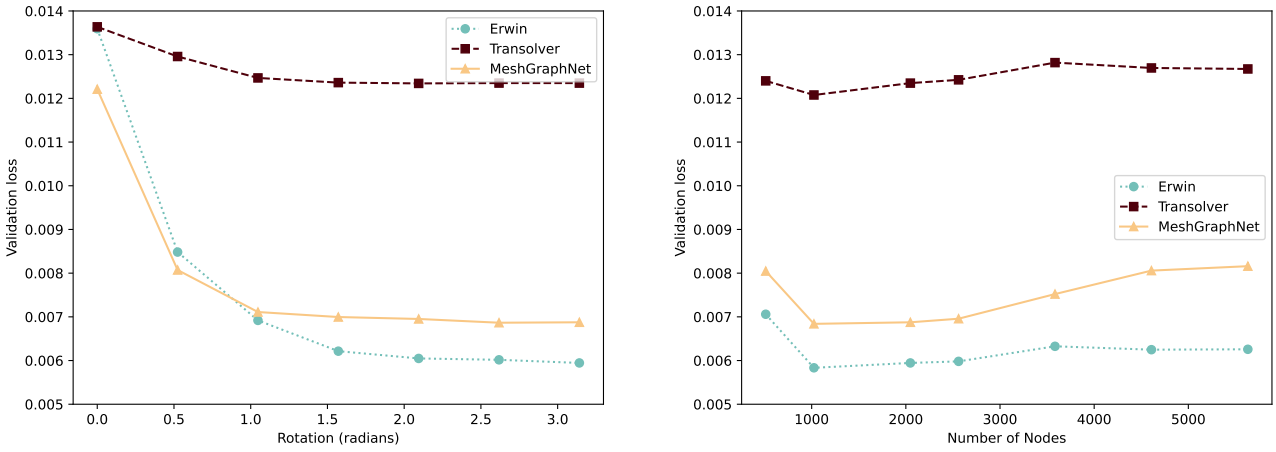
To gain a deeper understanding of the impact of rotational augmentations we applied rotations in a non-random way on the BendVessel dataset, starting with only data points rotated 0 degrees and predicting a datapoint rotated 180 degrees. We then introduced larger and larger rotations until we trained on every datapoint (0 - 170 degrees of rotations) except the 180 degree one to observe the impact on generalizability.

# Chapter 4

# Results

We compared Erwin, Transolver and MeshGraphNet on the BendVessel and the ClotEntry datasets. We first studied the impact of data augmentations on single-step validation loss, namely rotations, datapoint size and data generation. We then used the optimal data augmentation configuration found in those experiments to compare the performance of all three models on the test sets, again for a single step. Additionally, we investigated the loss on a multi-step rollout, while measuring inference speed. Finally, we performed an ablation study to investigate the how well Erwin generalizes to unseen rotations.

## 4.1 Data Augmentation




(a) (b)

Figure 4.1: Effects on performance of data augmentations for the validation set of the ClotEntry dataset. Figure a shows the impact of random rotations up to the rotation value specified. Figure b shows the impact of sampling different amounts of nodes from the maximum of 27000.

As can be seen in figure 4.1.a, increasing the rotations of the training set increased performance on ClotEntry across all models, with the increase for Erwin and MeshGraphNet being especially pronounced. In figure 4.1.b, a sharp increase in performance can be seen in the step from 500 to 1000 nodes. However, after this the number of nodes in each datapoint only has a marginal impact on the the performance past 1000 nodes for Erwin and Transolver. In contrast, the loss for MeshGraphNet increases slowly after 2500 nodes has been reached.

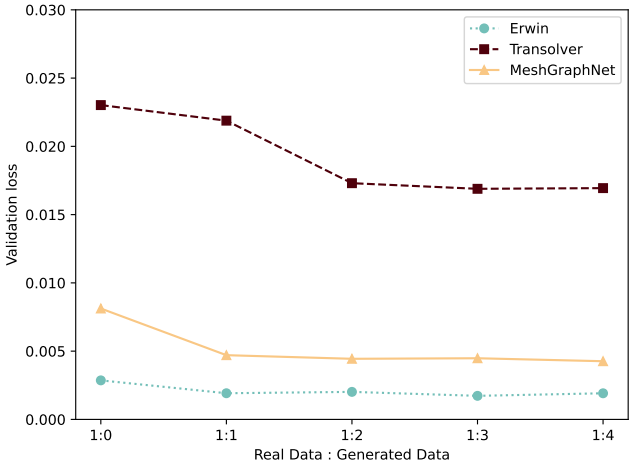



Figure 4.2: The effect on performance of adding generated datapoints using BaryCentric sampling on the BendVessel dataset for different ratios between real and generated data.

Figure 4.2 shows that performance increases when adding generated data in an amount equal to the real training data. However, this benefit quickly plateaus when the ratio is increased to more than $1:2$ real data to generated data.

## 4.2 Suitability as Surrogate Models

| Dataset | Erwin | Transolver | MeshGraphNet |
|---|---|---|---|
| BendVessel | $0.00166 \pm 0.00012$ | $0.02336 \pm 0.01016$ | $0.00469 \pm 0.00052$ |
| ClotEntry | $0.00799 \pm 0.00044$ | $0.01434 \pm 0.00020$ | $0.00784 \pm 0.00014$ |

Table 4.1: MSE loss on test set across 5 runs (mean $\pm$ std). Using the best data augmentations found in section 4.1, namely $\pi$ radians rotation for both datasets, a 1:2 ratio of real data to generated data for BendVessel and 2000 nodes for ClotEntry.

As shown in table 4.1 Erwin achieves the lowest test loss for BendVessel, followed by MeshGraphNet. On ClotEntry, Erwin and MeshGraphNet perform similarly, within one standard deviation of each other. On both datasets Transolver has the highest loss by a large margin.

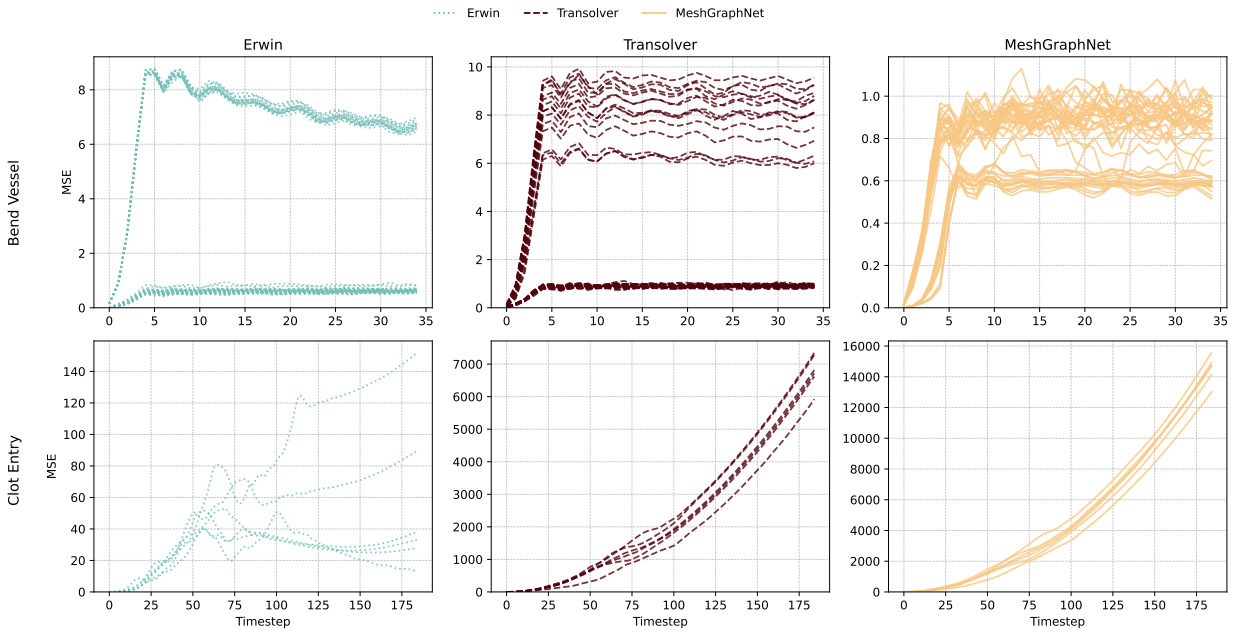



Figure 4.3: Rollouts for Erwin, Transolver and MeshGraphNet on BendVessel and ClotEntry. Rollouts are performed on all training set simulations. Note that each subplot has a different y-axis scale.

As seen in figure 4.3 the performances of our models on multi-step rollout between BendVessel and ClotEntry were quite different.

On BendVessel, the loss of all models stabilized after an initial increase. Each rollout of each model seemed to stabilize into one of two different regions, with the bottom region having a low loss for each model. The top region differed per model, with Erwin and Transolver both having a high loss, while the top region of MeshGraphNet had a loss just a little bit higher than its lower region.

For ClotEntry, the loss of each model diverged rapidly. The values of the loss were much higher than the size of the blood vessel indicating that the blood clot moved outside the blood vessel and away from it.

| Dataset | Metric | Erwin | Transolver | MeshGraphNet | Abaqus |
|---|---|---|---|---|---|
| BendVessel | Rollout time | $2.167 \pm 0.010$ | $0.65 \pm 0.003$ | $0.56 \pm 0.003$ | 180 |
| | Speedup | $\times 82$ | $\times 278$ | $\times 322$ | – |
| ClotEntry | Rollout time | $13.19 \pm 0.03$ | $4.33 \pm 0.02$ | $3.77 \pm 0.02$ | 1500 |
| | Speedup | $\times 113$ | $\times 346$ | $\times 398$ | – |

Table 4.2: Average rollout times for one simulation in seconds (mean $\pm$ std) and speedups relative to the traditional solver (Abaqus). The results shown are multiplied by $\frac{40}{35}$ for BendVessel and $\frac{200}{160}$ for ClotEntry to adjust for the fact that the beginning of each simulation was not included in rollout for the models, but was included for the traditional solver.

The results in table 4.2 show that MeshGraphNet performed the fastest rollouts on both datasets, closely followed by Transolver, while Erwin was significantly slower than the other models. Compared to the traditional solvers all models showed a significant speedup.

## 4.3 Generalization to unseen Geometries

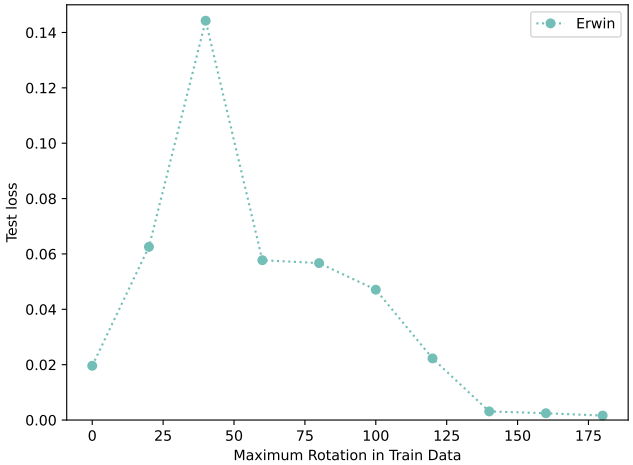



Figure 4.4: Effect of including different rotations in the training data on performance on the test set for Erwin on BendVessel. Each simulation in BendVessel uses the exact same geometry rotated around one axis in 10 degree increments, with the test set rotated 190 degrees. The x-axis displays the simulations that are included in the training set. For each x-value all simulations are included that have a rotation between 0 and that x-value inclusive (for example, an x-value of 20 entails the inclusion of simulations for 0, 10 and 20 degrees).

As shown in figure 4.4 the loss decreases when the maximum rotation of the training data comes close to the rotation of the test data. However, when the maximum rotation is not close to the test data rotation the loss is much higher, spiking at maximum rotation 40.

# Chapter 5

# Discussion

In this thesis, we investigated how well current machine learning methods are suited as step-by-step surrogate models for numerical mechanical thrombectomy simulations by training Erwin, Transolver and MeshGraphNet on the BendVessel and ClotEntry datasets. Regarding single step performance, we found that Erwin performed well on BendVessel, while both Erwin and MeshGraphNet achieved good results for ClotEntry. On longer timeframes, we found that the loss of all models of stabilized for BendVessel, with MeshGraphNet stabilizing at lower loss values more often. In contrast, we found that the loss of all models quickly diverged for ClotEntry. Furthermore, we found that all three models provided significant speedups over traditional numerical solvers, with MeshGraphNet being the fastest.

To improve single-step performance, we ran experiments using various data augmentations, namely rotations, datapoint size and data generation. We found that larger rotations improved performance, datapoint size has a marginal impact for our models and that a ratio of $1:2$ real data to generated data was optimal. Lastly, we performed an ablation study that investigated the generalizability of Erwin on unseen rotations and found that generalization only occurred when the training data closely resembled the test data.

## 5.1 Suitability as step-by-step surrogate models for numerical mechanical thrombectomy simulations

Our results of single-step performance on the test set showcased Erwins ability to perform well in both the small and medium data regimes. Erwin performed especially well on BendVessel compared to Transolver and MeshGraphNet, see table 4.1. One possible explanation of this fact is that Erwin can take almost full advantage of the attention mechanism at this lower datapoint size. This is because it performs full self-attention within each ball and the ball size (128) is of the same order of magnitude as the node count for BendVessel (768). When this is the case, the first BallProcessing layer of the encoder can encode each node with information of a large portion of the total nodes, as there are few balls each with relatively many nodes. This then results in the nodes in subsequent balls having more informative embeddings compared to when Erwin has to rely on aggregating information using its balltree, which is not a lossless process.

Furthermore, MeshGraphNet performed on par with Erwin for ClotEntry, but underperformed for BendVessel. This showcases that the best performing model can differ depending on the specifics of the dataset. Furthermore, it shows that MeshGraphNet could not take full advantage of this smaller datapoint size. This might be related to its use of the message-passing framework. This gives the model a certain receptive field size, which indicates how far apart (in terms of edges) the nodes can still interact. Since it performed well on the larger dataset

we can assume it was large enough for that task. It then makes sense why the performance did not increase as much as Erwins when trained on the smaller BendVessel dataset. If the receptive field size was already large enough for the large task we would expect there to be no gain from decreasing the amount of nodes, as far apart nodes relevant to each other could already interact.

Lastly, Transolver underperformed considerably for both datasets providing evidence that it is less suited to simulation emulation than the other models. Since both Erwin and Transolver are based on the transformer and Erwin did perform well this shows that the other parts of their architecture had a large impact. Since their main difference is how they apply attention, it could be the case that the approach of Transolver to divide the inputs into slices does not work well for 3D pointclouds. These slices are made globally (although they can use the spatial embeddings), while the balls that Erwin uses are explicitly made to be based on local position, which might explain this. However, Erwin and Transolver have many smaller design details, which each could explain this difference.

In contrast to the single-step performance, the rollout results show less positive outcomes. While all models showed stable results for BendVessel, none of them performed well on ClotEntry across a longer time domain. Since these datasets differ mainly in the complexity of their geometry and in their node count, this indicates that one of these factors is important to stability. One possibility is that the vessel walls of ClotEntry are less smooth and regular than those of BendVessel, which makes rollout less forgiving as mistakes are quickly amplified. Interestingly, for BendVessel the loss of all models seemed to stabilize into two regions. Since the datapoints in BendVessel are very similar to each other, this could indicate that the models predict two separate outcomes, with one better matching the outcome of the numerical solver. If this is the case, then when using our methods in practice it would be useful to run the model multiple times and to determine if different outcomes occur as this would give insight into the models uncertainty.

Our results regarding the speed of our models show that all three models are capable of large speedups over traditional solvers. MeshGraphNet was the fastest and Erwin the slowest, with a difference of a factor of 4. However, since the speedups over traditional solvers were in the range of 80 - 400 times this difference is small in absolute rollout time, which was in the range of 0.5 - 13.2 seconds. This meant that in the worst case, Erwin was 10 seconds slower than MeshGraphNet. We do see this difference in absolute rollout time increase between BendVessel and ClotEntry, which indicates that this difference could become significant as the datapoints increase in size. This is supported by the fact that in general the speed of MeshGraphNets message passing architecture scales linearly with the number of nodes with constant node connectivity while attention scales quadratically. Of course, Erwin and Transolver try to mitigate this quadratic scaling as much as possible and thus comparing the scaling of these models is not straightforward, especially when considering that datasets with more nodes likely also require different hyperparameters for good performance.

Collectively, these findings suggest that both Erwin and MeshGraphnet are suitable as step-by-step surrogate models in the small data regime while Transolver is not. Erwin is very well suited to the small data regime, but as the size of the datapoints grow the speed advantage of MeshGraphNet becomes more significant. In the medium data regime, Erwin and MeshGraphNet show promising results as they are both fast and accurate for single steps. However, they currently miss the stability required to perform longer rollouts, making them not practically useful for this task as of yet. Broadly, this means that as datasets increase in complexity we expect that more care needs to be taken to ensure stability. However, as we have seen, in terms of single-step performance and speed the methods, specifically Erwin and MeshGraphNet, presented in this thesis are likely well-suited to more complex domains.

## 5.2 Impact of data augmentations on performance

Firstly, the results of our experiments using rotations show that using no rotations, the differences in performance between Erwin, Transolver and MeshGraphNet are small. However, Erwin and MeshGraphNet benefit greatly from adding rotations to the training data. In contrast, Transolver seems to have its loss only decreased slightly when rotations are introduced. This suggests that one of the reasons that Transolver underperformed on the test set could be that it can not take advantage of the addition of rotational data augmentation. This is interesting as we used the same positional embeddings for Transolver and Erwin, which imply that the BallTree structure, and its breaking of rotational symmetry, of Erwin is one of its key contributing factors to its success using rotation augmentation.

Secondly, the evidence from our experiments varying the number of nodes indicates that all three models are robust to changes in node densities. After an initial increase in loss when the node count was too low, the loss stayed consistent when the amount of sampled nodes increased. This robustness to the node count shows that the hyperparameters, including the node connection radius, that were chosen generalize well to different node densities, as they were the same for all experiments.

Lastly, the results from our experiments using varying amounts of generated training data indicate that more generated data improves performance up to a point after which it has no impact. Furthermore, these improvements for all models show that BaryCentricSampling creates useful additional training datapoints. This implies that the new nodes that it creates on triangles represent the original nodes in a consistent way while they are different enough to still provide new information.

## 5.3 Generalizability to unseen geometries

The results from our experiment on generalizability show that Erwin did not generalize to an unseen rotation of the data, unless the training data contained datapoints had a rotation very similar to it. This suggests that in order for Erwin to perform well its training data must be adjusted to closely resemble the task it will be used on. In our experiments using rotations as data augmentations we saw that Erwin already performed well using 1.5 radians of rotations on ClotEntry, which is not enough to transform every training datapoint into every direction (but is enough to rotate some to each direction as each datapoint is a vessel with different geometry). In combination, these findings suggest that while Erwin requires examples that closely match the test data, it can already perform well if only a portion of the training data does so.

Regarding our other models, since Transolver did not benefit from rotations nearly as much as Erwin did for our data augmentation experiment, it is possible that it would perform better on this generalizability task. This would for example be the case if the results from the data augmentation experiment were due to Transolver being partially rotation equivariant.

## 5.4 Limitations

The main limitation of this thesis is the completeness of the simulations that were used. They included large amounts of nodes, realistic vessel geometry and accurate physics simulation, but they did not include any devices explicitly nor did they use realistic blood clots (with different shapes and compositions). Simulations with explicit devices could prove challenging to emulate using the methods described in this thesis, especially if they use stent retrievers as these are complex devices with many small moving parts. However, since the simulations included realistic physics and many nodes, we argue that it still provides an adequate test

to challenge our models. Another limitation is that no extensive efficiency optimization was done. As such, the speed of the methods presented here might not scale well as the size of the datapoints increases, possibly rendering them impracticable. Furthermore, the methods used in this thesis mostly focused on single-step performance and not on rollout performance. As such, methods that specifically focus on longer timeframes were excluded from our analysis. Lastly, our methods did not take into account all the information provided by the physical nature of our simulation. Since the simulation was based on physical laws, the constraints of these laws (conservation of momentum etc.) are information that could be incorporated into our models, possibly improving performance and stability.

## 5.5 Outlook

The methods presented in this thesis show promising results for the emulation of numerical physics simulations of mechanical thrombectomy using machine learning. However, how these methods will perform for more realistic simulations, those that include detailed devices and more nodes, is still not clear. Thus, before it can be considered how machine learning models can be used practically for this problem, future research must investigate how these methods perform on realistic simulations.

Future research could perform experiments on some of the simulations that are already avaliable today, as they already include detailed thrombectomy devices. This is currently challenging as most simulations that are presented in research papers are not made available as datasets with machine learning in mind. Instead the data needed for training machine learning models is discarded as soon as analysis is done.

Furthermore, for our methods, stability over longer time periods is a problem for more complicated datasets. This poses another challenge for the practical application of these methods as generating rollouts is their most interesting use case. Future research could incorporate existing methods for improving stability such as training on multi-step rollouts or use machine learning architectures that are specifically made for increased rollout stability, such as LE-PDE [48].

To improve performance even more, further research could focus on finding priors that can be added to the loss during training. It is possible that useful priors exist that can take advantage of the specific dynamics present in mechanical thrombectomy. An example of such a prior is that the distances between adjacent blood clot nodes largely stay the same across timesteps, as blood clots are not fluid and thus nodes can not move freely throughout the clot. Additionally, since this is a known physical system, physics based constraints can be used, such as those used in Physics-Informed Neural Networks [61].

As the quality of simulations improve their usefulness in clinical settings will increase. As such, the potential for the use of machine learning methods to speed them up increases as well. The methods we have presented are a first step to this use-case, as they show how machine learning methods can be used to speed up simplified numerical simulations of thrombectomy step-by-step. If these methods are eventually improved enough they might enable the practical use of increasingly more complicated simulations in time-sensitive clinical situations. In the long term, methods such as ours can contribute to the use of fast emulations of patient-specific numerical simulations for ischemic stroke patients, giving operators more insight into the potential results of their procedures. While challenges such as rollout stability and scaling to large datasets remain, the findings of this thesis indicate that machine learning based step-by-step surrogate models provide a promising approach for speeding up numerical physics simulations of mechanical thrombectomy and thus of their use in clinical settings.

## 5.6 Conclusion

This thesis investigated the suitability of step-by-step machine learning surrogate models for accelerating numerical simulations of mechanical thrombectomy. We trained three machine learning models on two datasets that involved simplified mechanical thrombectomy procedures, showing that two models achieved high single-step accuracy and that all models provided a significant speedup over traditional solvers. However, for multi-step rollout, stable results were only observed for simple geometries, while rollouts on complex geometries were unstable for all models, showcasing the need for techniques that provide multi-step stability. Additionally, we investigated the impact of data augmentations and found that they were critical to model performance. Namely, we found that larger rotations and the addition of generated data improved performance, while datapoint size had a marginal impact for our models. Lastly, we investigated the generalizability of one of our models to unseen geometries and found that generalization requires a close match between training and testing data, underscoring the important role of data augmentation in the performance of surrogate models. Overall, our findings highlight the potential for machine learning based step-by-step surrogate models to speed up numerical thrombectomy simulations and provides a foundation for future studies to develop stable methods that scale to realistic simulations.

# Appendix A

# Hyperparameters

Any hyperparameters not mentioned here were left at their default values.

## A.1 Erwin

| **Parameter** | **Value** |
|---|---|
| Batch size | 8 |
| Radius | 0.766 |
| Hidden dimension | 32 |
| Encoder heads | [4, 4, 4, 4] |
| Encoder depths | [8, 10, 12, 14] |
| Decoder heads | [4, 4, 4] |
| Decoder depths | [4, 4, 4] |
| Strides | [2, 2, 2] |
| Ball sizes | [128, 128, 128, 128] |
| MLP ratio | 4 |
| Rotate | 45 |
| Optimizer | AdamW |
| LR scheduler | Linear Warmup + Cosine Decay |
| Learning rate | 1.00E–04 |
| Warmup epochs | 10 |
| Total epochs | 300 |

Table A.1: Hyperparameter settings for Erwin

## A.2 Transolver

| Parameter | Value |
|---|---|
| Batch size | 16 |
| Radius | 0.576 |
| Hidden dimension | 64 |
| Learning rate | 1.00E–04 |
| Dropout | 0.576 |
| Number of heads | 16 |
| Number of layers | 5 |
| Slice number | 24 |
| Optimizer | AdamW |
| LR scheduler | Linear Warmup + Cosine Decay |
| Warmup epochs | 10 |
| Total epochs | 300 |

Table A.2: Hyperparameter settings for Transolver

## A.3 MeshGraphNet

| Parameter | Value |
|---|---|
| Batch size | 8 |
| Radius | 0.56 |
| Hidden dimension | 64 |
| Learning rate | 10E–04 |
| Processor size | 15 |
| Optimizer | AdamW |
| LR scheduler | Linear Warmup + Cosine Decay |
| Warmup epochs | 10 |
| Total epochs | 300 |

Table A.3: Hyperparameter settings for MeshGraphNet

## A.4 Learning Rate Scheduler

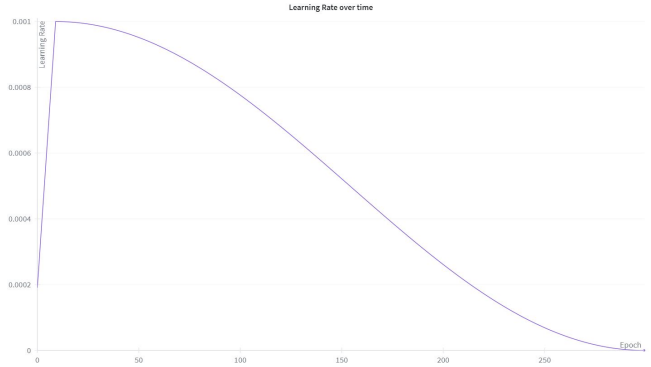



Figure A.1: Example learning rate schedule, with 300 epochs, a learning rate value of 0.001 and using a linear warmup of 10 epochs + Cosine Decay

# Bibliography


[1] Jiahui Fan, Xiaoguang Li, Xueying Yu, Zhenqiu Liu, Yanfeng Jiang, Yibin Fang, Ming Zong, Chen Suo, Qiuhong Man, and Lize Xiong. Global burden, risk factor analysis, and prediction study of ischemic stroke, 1990–2030. *Neurology*, 101(2):e137–e150, 2023.

[2] Ivo GH Jansen, Maxim JHL Mulder, and Robert-Jan B Goldhoorn. Endovascular treatment for acute ischaemic stroke in routine clinical practice: prospective, observational cohort study (mr clean registry). *bmj*, 360, 2018.

[3] Sharath Kumar Anand, William J Benjamin, Arjun Rohit Adapa, Jiwon V Park, D Andrew Wilkinson, Badih J Daou, James F Burke, and Aditya S Pandey. Trends in acute ischemic stroke treatments and mortality in the united states from 2012 to 2018. *Neurosurgical focus*, 51(1):E2, 2021.

[4] Maria Giulia Mosconi and Maurizio Paciaroni. Treatments in ischemic stroke: current and future. *European Neurology*, 85(5):349–366, 2022.

[5] Stephan A Munich, Kunal Vakharia, and Elad I Levy. Overview of mechanical thrombectomy techniques. *Neurosurgery*, 85(suppl_1):S60–S67, 2019.

[6] Osama O Zaidat, Alicia C Castonguay, Italo Linfante, Rishi Gupta, Coleman O Martin, William E Holloway, Nils Mueller-Kronast, Joey D English, Guilherme Dabus, Tim W Malisch, et al. First pass effect: a new measure for stroke thrombectomy devices. *Stroke*, 49(3):660–666, 2018.

[7] J Kaesmacher, T Boeckh-Behrens, S Simon, C Maegerlein, JF Kleine, C Zimmer, L Schirmer, H Poppert, and T Huber. Risk of thrombus fragmentation during endovascular stroke treatment. *American Journal of Neuroradiology*, 38(5):991–998, 2017.

[8] Leonard LL Yeo, Pervinder Bhogal, Anil Gopinathan, Yang Cunli, Benjamin Tan, and Tommy Andersson. Why does mechanical thrombectomy in large vessel occlusion sometimes fail? a review of the literature. *Clinical Neuroradiology*, 29(3):401–414, 2019.

[9] B. Lapergue, R. Blanc, B. Gory, et al. Effect of endovascular contact aspiration vs stent retriever on revascularization in patients with acute ischemic stroke and large vessel occlusion: The aster randomized clinical trial. *JAMA*, 318(5):443–452, 2017.

[10] Mahdi Daei Daei. *Computational modeling of interventional and implantable endovascular devices*. PhD thesis, Institut Polytechnique de Paris, 2023.

[11] Giulia Luraghi, Sara Bridio, Jose Felix Rodriguez Matas, Gabriele Dubini, Nikki Boodt, Frank JH Gijsen, Aad van der Lugt, Behrooz Fereidoonnezhad, Kevin M Moerman, Patrick McGarry, et al. The first virtual patient-specific thrombectomy procedure. *Journal of Biomechanics*, 126:110622, 2021.

[12] Ronghui Liu, Chang Jin, Lizhen Wang, Yisong Yang, Yubo Fan, and Weidong Wang. Simulation of stent retriever thrombectomy in acute ischemic stroke by finite element analysis. *Computer Methods in Biomechanics and Biomedical Engineering*, 25(7):740–749, 2022.

[13] Mathias Grøan, Johanna Ospel, Soffien Ajmi, Else Charlotte Sandset, Martin W Kurz, Mona Skjelland, and Rajiv Advani. Time-based decision making for reperfusion in acute ischemic stroke. *Frontiers in Neurology*, 12:728012, 2021.

[14] Tapuwa D Musuka, Stephen B Wilton, Mouhieddin Traboulsi, and Michael D Hill. Diagnosis and management of acute ischemic stroke: speed is critical. *Cmaj*, 187(12):887–893, 2015.

[15] Zongyi Li. Neural operator: Learning maps between function spaces. In *2021 Fall Western Sectional Meeting*. AMS, 2021.

[16] AmirPouya Hemmasian and Amir Barati Farimani. Multi-scale time-stepping of partial differential equations with transformers. *Computer Methods in Applied Mechanics and Engineering*, 426:116983, 2024.

[17] Hidehisa Nishi, Naoya Oishi, Akira Ishii, Isao Ono, Takenori Ogura, Tadashi Sunohara, Hideo Chihara, Ryu Fukumitsu, Masakazu Okawa, Norikazu Yamana, et al. Predicting clinical outcomes of large vessel occlusion before mechanical thrombectomy using machine learning. *Stroke*, 50(9):2379–2388, 2019.

[18] Liang Liang, Minliang Liu, Caitlin Martin, John A Elefteriades, and Wei Sun. A machine learning approach to investigate the relationship between shape features and numerically predicted risk of ascending aortic aneurysm. *Biomechanics and modeling in mechanobiology*, 16(5):1519–1533, 2017.

[19] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International conference on learning representations*, 2020.

[20] Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for pdes on general geometries (2024). *URL https://arxiv.org/abs/2402.02366*.

[21] Maksim Zhdanov, Max Welling, and Jan-Willem van de Meent. Erwin: A tree-based hierarchical transformer for large-scale physical systems. *arXiv preprint arXiv:2502.17019*, 2025.

[22] Debbie Beumer, Maxim JHL Mulder, Ghesrouw Saiedie, Susanne Fonville, Robert J van Oostenbrugge, Wim H van Zwam, Philip J Homburg, Aad van der Lugt, and Diederik WJ Dippel. Occurrence of intracranial large vessel occlusion in consecutive, non-referred patients with acute ischemic stroke. *Neurovascular Imaging*, 2(1):11, 2016.

[23] Kanchan Kapoor, Balbir Singh, and Inder Jit Dewan. Variations in the configuration of the circle of willis. *Anatomical science international*, 83(2):96–106, 2008.

[24] Carmelo Lucio Sturiale, Alba Scerrati, Luca Ricciardi, Oriela Rustemi, Anna Maria Auricchio, Nicolò Norri, Amedeo Piazza, Fabio Raneri, Alberto Benato, Alessio Albanese, et al. Geometry and symmetry of willis' circle and middle cerebral artery aneurysms development. *Journal of Clinical Medicine*, 13(10):2808, 2024.

[25] Ashutosh P Jadhav, Shashvat M Desai, and Tudor G Jovin. Indications for mechanical thrombectomy for acute ischemic stroke: current guidelines and beyond. *Neurology*, 97(20_Supplement_2):S126–S136, 2021.

[26] Yilmaz Onal, Murat Velioglu, Ugur Demir, Erhan Celikoglu, and Hakki M Karakas. Feasibility of distal mechanical thrombectomy in m3, a3 and p3 segments via a 0.013-inch delivery system: preliminary experience. *Turk Neurosurg*, 30(4):614–620, 2020.

[27] Don F Schomer, Michael P Marks, Gary K Steinberg, Iain M Johnstone, Derek B Boothroyd, Michael R Ross, Norbert J Pelc, and Dieter R Enzmann. The anatomy of the posterior communicating artery as a risk factor for ischemic cerebral infarction. *New England Journal of Medicine*, 330(22):1565–1570, 1994.

[28] Nadine Wortmann, Thomas Andersek, Helena Guerreiro, Anna A Kyselyova, Andreas M Frölich, Jens Fiehler, and Dieter Krause. Development of synthetic thrombus models to simulate stroke treatment in a physical neurointerventional training model. *All Life*, 15(1):283–301, 2022.

[29] Precious Jolugbo and Robert AS Ariens. Thrombus composition and efficacy of thrombolysis and thrombectomy in acute ischemic stroke. *Stroke*, 52(3):1131–1142, 2021.

[30] Lucas Di Meglio, Jean-Philippe Desilles, Véronique Ollivier, Mialitiana Solo Nomenjanahary, Sara Di Meglio, Catherine Deschildre, Stéphane Loyau, Jean-Marc Olivot, Raphaël Blanc, Michel Piotin, et al. Acute ischemic stroke thrombi have an outer shell that impairs fibrinolysis. *Neurology*, 93(18):e1686–e1698, 2019.

[31] Sharath Kumar GG and Chinmay P Nagesh. Acute ischemic stroke: a review of imaging, patient selection, and management in the endovascular era. part ii: Patient selection, endovascular thrombectomy, and postprocedure management. *Journal of Clinical Interventional Radiology ISVIR*, 2(03):169–183, 2018.

[32] Raphaël Blanc, Simon Escalard, Humain Baharvadhat, Jean Philippe Desilles, William Boisseau, Robert Fahed, Hocine Redjem, Gabriele Ciccio, Stanislas Smajda, Benjamin Maier, et al. Recent advances in devices for mechanical thrombectomy. *Expert review of medical devices*, 17(7):697–706, 2020.

[33] Penumbra Inc. Penumbra ace. https://www.penumbrainc.com/products/penumbra-ace/. Accessed: 2025-11-28.

[34] Medtronic. React aspiration catheter. https://www.medtronic.com/en-us/healthcare-professionals/products/neurological/neurovascular/catheters/aspiration-neurovascular-catheters/react-catheter.html. Accessed: 2025-11-28.

[35] Stryker. Trevo xp provue retriever. https://www.stryker.com/us/en/neurovascular/products/trevo–xp-provue-retriever-.html. Accessed: 2025-11-28.

[36] Terumo Neurovascular. Eric device. https://www.terumoneuro.com/emea/products/eric. Accessed: 2025-11-28.

[37] Frank J Rizzo. An integral equation approach to boundary value problems of classical elastostatics. *Quarterly of applied mathematics*, 25(1):83–95, 1967.

[38] Ted Belytschko, Jerry I Lin, and Tsay Chen-Shyh. Explicit algorithms for the nonlinear dynamics of shells. *Computer methods in applied mechanics and engineering*, 42(2):225–251, 1984.

[39] Yen-Sen Chen, TH Chou, BR Gu, Jong-Shinn Wu, Bill Wu, YY Lian, and Luke Yang. Multiphysics simulations of rocket engine combustion. *Computers & Fluids*, 45(1):29–36, 2011.

[40] Karlheinz Langanke, Joachim A Maruhn, and Steven E Koonin. *Computational nuclear physics*. Springer, 1991.

[41] Ivan Fumagalli, Stefano Pagani, Christian Vergara, Dilachew A Adebo, Maurizio Del Greco, Antonio Frontera, Giovanni Battista Luciani, Gianluca Pontone, Roberto Scrofani, Alfio Quarteroni, et al. The role of computational methods in cardiovascular medicine: a narrative review. *Translational Pediatrics*, 13(1):146, 2024.

[42] Leah A Groves, Mohamed Keita, Saidou Talla, Ron Kikinis, Gabor Fichtinger, Parvin Mousavi, and Mamadou Camara. A review of low-cost ultrasound compatible phantoms. *IEEE Transactions on Biomedical Engineering*, 70(12):3436–3448, 2023.

[43] Nikki Boodt, Philip RW Snouckaert van Schauburg, Hajo M Hund, Behrooz Fereidoonnezhad, J Patrick McGarry, Ali C Akyildiz, Adriaan CGM Van Es, Simon F De Meyer, Diederik WJ Dippel, Hester F Lingsma, et al. Mechanical characterization of thrombi retrieved with endovascular thrombectomy in patients with acute ischemic stroke. *Stroke*, 52(8):2510–2517, 2021.

[44] Rui Wang, Elyssa Hofgard, Han Gao, Robin Walters, and Tess E Smidt. Discovering symmetry breaking in physical systems with relaxed group convolution, 2024. *URL https://arxiv. org/abs/2310.02299*.

[45] W Patrick Walters and Regina Barzilay. Applications of deep learning in molecule generation and molecular property prediction. *Accounts of chemical research*, 54(2):263–270, 2020.

[46] Liang Liang, Minliang Liu, Caitlin Martin, and Wei Sun. A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. *Journal of The Royal Society Interface*, 15(138):20170844, 2018.

[47] Francesco Regazzoni, Stefano Pagani, Matteo Salvador, Luca Dede', and Alfio Quarteroni. Learning the intrinsic dynamics of spatio-temporal processes through latent dynamics networks. *Nature Communications*, 15(1):1834, 2024.

[48] Tailin Wu, Takashi Maruyama, and Jure Leskovec. Learning to accelerate partial differential equations via latent global evolution. *Advances in Neural Information Processing Systems*, 35:2240–2253, 2022.

[49] Jay Alammar. The illustrated transformer. https://jalammar.github.io/illustrated-transformer/, 2018. Accessed: 2026-01-31; originally published June 27, 2018.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[51] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. Pmlr, 2017.

[52] TIMI Study Group. The thrombolysis in myocardial infarction (timi) trial. phase i findings. *The New England Journal of Medicine*, 312(14):932–936, April 4 1985.

[53] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[54] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[55] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

[56] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[57] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[58] Thijs P Kuipers, Praneeta R Konduri, Henk Marquering, and Erik J Bekkers. Generating cerebral vessel trees of acute ischemic stroke patients using conditional set-diffusion. In *Medical Imaging with Deep Learning*, 2024.

[59] PhysicsNeMo Contributors. Nvidia physicsnemo: An open-source framework for physics-based deep learning in science and engineering, February 2023. Accessed: 2026-01-21.

[60] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[61] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.