TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

TUM Chair of Aerodynamics and Fluid Mechanics

Final report of project:

Design Optimization with Differentiable Particle-Based Solvers

Authors	Jiayi Li, Nikhita Kurupakulu Venkat, Uppili Srinivasan Venkatesan, Meike Tütken
TUM Mentors	M.Sc. Artur Toshev, M.Sc. Benjamin Holzschuh
Project lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (MDSI)

Feb 2024

Acknowledgements

We express our sincere gratitude to the Chair of Aerodynamics and Fluid Mechanics at the Technical University of Munich (TUM), the Munich Data Science Institute (MDSI), and Prof. Dr. Thuerey at the Associate Professorship of Physics-based simulation for granting us the opportunity to collaborate on this project. Our heartfelt appreciation goes to Mr. Artur Toshev and Mr. Benjamin Holzschuh for their invaluable support and guidance during the project. Special thanks to Mr. Harish Ramachandran for laying the groundwork for this project and providing valuable assistance. We greatly appreciate the support of Dr Ricardo Acevedo Cabra as the project lead and are grateful for the supervision of Prof. Dr. Massimo Fornasier.

Abstract

This report investigates the optimization of hot wall shapes in a channel flow simulation to maximize the heat outflow, which is crucial in the aerospace and automotive industries for efficient engine cooling and battery performance in electric vehicles. The optimization utilizes a Smoothed Particle Hydrodynamics (SPH) solver and a Graph Network-based Simulation (GNS) model. The SPH solver simulates fluid dynamics and heat transfer, while the surrogate model, built on GNS, aims to provide stable gradients for optimization. Gradient checkpointing and accumulation techniques are utilized to manage memory and computational resources. While optimizing with the SPH solver, we compare optimization with and without gradient accumulation, showing improvements in stability and efficiency with the implementation of gradient accumulation. We observed that the simulation converges to a specific type of shape with different initial conditions, which agrees with theoretical analysis. Our results using the surrogate model can emulate the behavior of the SPH solver to a certain extent, albeit with some limitations. Unlike the numerical solver, the surrogate model condenses the equivalent of 100 SPH solver steps into each surrogate model step, resulting in more stable gradients. We successfully executed the surrogate simulation and computed the resulting loss.

Contents

1 Introduction

The integration of Machine Learning with Computational Fluid Dynamics has evidently seen great advances in recent years with the development of differential fluid mechanics solvers such as PhiFlow [12] JAX-Fluids [4]. Design optimization is a key area where such solvers can be utilized and where the availability of gradients is highly advantageous. To expand the range of differentiable solvers, the implementation of a Smoothed Particle Hydrodynamics (SPH) solver in JAX has been accomplished based on the formulation presented in the work by [1].

In this project, we focus on optimizing the shape of the hot wall inside a channel flow simulation. This baseline scenario case holds relevance across a wide range of applications. Effective management of heat diffusion and convection is critical in industries such as aerospace and automotive, where it contributes to enhanced engine cooling and improved battery performance in electric vehicles.

To optimize the hot wall shape of the fluid channel, we use either the SPH solver or its respective surrogate model. For optimization we implement the methods used by [1] and [2]. The key idea of SPH is to use a smoothing kernel and consequently calculate the spatial derivatives only from interactions with neighboring particles. The experiments from [2] conclude that Graph Neural Networks (GNNs) can now support gradient-based, high-dimensional inverse design which demands high accuracy, well-behaved gradients, long-term rollout stability, and generalization beyond the training data. Therefore, we will not only use the SPH solver for the optimization but also a surrogate model as explained further in 4, which is a graph network-based simulation [17]. This surrogate model is trained on numerous simulations on a dataset of the SPH solver with different shapes of the hot wall. In figure 1 you can see an overview of the general optimization process.

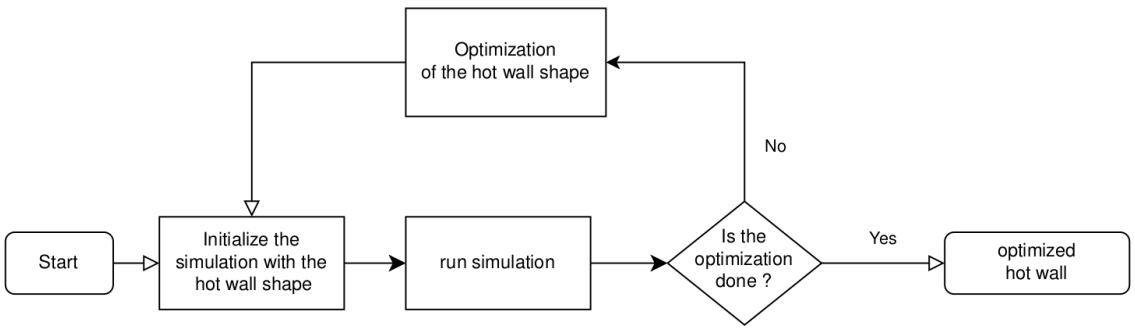


Figure 1: Overview of the optimization process.

For optimization we employ the Adam optimizer [13]. We test our method with different surface shapes selected as the initial condition of the degrees of freedom. We observed that, with different initial conditions, the simulation converges to a specific type of shape. The results also coincide with the theoretical analysis by [14]. We also observed that

we obtained a more stable and accurate optimization with gradient accumulation. The simulation of the trained surrogate model shows a decrease in evaluation and training error while producing stable gradients.

In section 2, we introduce the numerical methods and inverse design optimization and meticulously define the problem setup. Firstly, the optimization process and its results with the SPH solver are presented in section 3. Then, an introduction to the surrogate model and the process of optimization with it is presented in section 4. Finally, we conclude the project and give an outlook.

2 Problem Definition

2.1 Numerical Methods for Modeling Complex Dynamical Systems

Partial differential equations play an important role in describing the spatial and temporal behavior of systems in engineering and physics. For simpler dynamical systems, analytical solutions, if they exist, provide the most accurate solutions. However, most dynamical systems are complex, and analytical solutions are often intractable or difficult to compute. Therefore, popular numerical methods like finite elements or finite volumes are essential to model such systems [20].

Broadly, numerical methods are classified into two primary approaches: Eulerian and Lagrangian. Eulerian methods use grid-based or mesh-based methods. For example, in our case of fluid flow, Eulerian methods use fixed grids or meshes to simulate fluid flow. On the other hand, Lagrangian methods use particles or material points that move with the local deformation of the continuum [21]. The difference between Eulerian and Lagrangian methods can be easily understood from Figure 2, which describes the Eulerian and Lagrangian method of delivering a package from point A to point B.

SPH is a Lagrangian method, meaning that particles are not fixed in space but move following the flow. The mesh-free characteristic of SPH renders it highly suitable for modeling extremely dynamic and turbulent flows, such as tank sloshing, dam breaches, nozzle flows, and similar scenarios [21]. SPH performs better when dealing with free surface flows than in cases where the fluid is not entirely enclosed within the boundaries. Also, the Lagrangian nature of SPH can automatically handle the interaction between particles and boundaries, thus reducing efforts for preliminary work.

2.2 Inverse Design Optimization

Automatically designing objects to exhibit a desired property, often called inverse design, holds the potential to revolutionize various fields such as aerodynamics, material science, optics, and robotics. Through learned generative models or policies, these approaches can suggest potential design parameters specific to a particular objective. While these

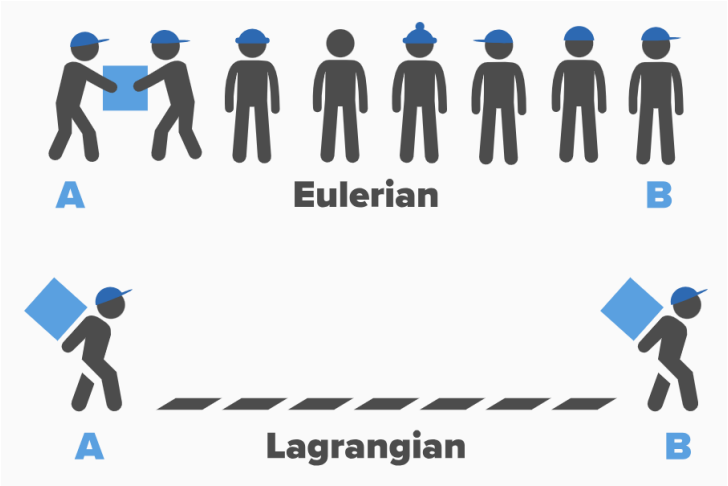


Figure 2: Comparison of the Eulerian (top) and Lagrangian approaches (bottom) [10].

techniques excel at solving specific tasks with greater efficiency compared to conventional methods, they are not without their limitations. They rely on access to a dataset comprising example designs and tasks for training and often struggle to adapt to tasks or designs beyond this predefined scope. A category of learned physics simulators utilizing Graph Neural Networks (GNNs) emerges as a promising solution to support inverse design [2].

We take inspiration from [2] for our design optimization task. In the paper, high dimensional fluid manipulation experiments demonstrate that gradient-based optimization on learned models can find high-quality designs over hundreds of time steps.

The idea proposed in [2] is implemented to suit the use case. The SPH solver is used as a baseline, and a learned GNN is used to evaluate the comparisons. The idea of maximizing the reward function is implemented, where the main goal is maximizing the heat outflow (Q). Similar to the paper, we are interested in evaluating the gradient-based optimization methods (e.g., we are using Adam instead of SGD), which require fewer function evaluations and scale better to large design spaces than sampling-based techniques. Our design parameters are denoted by ϕ, which are the degrees of freedom or the z-coordinates of the particles. We employ a gradient-based reward function optimization by finding the design parameters ϕ^* that maximize the loss function $\phi^* = \text{argmax}(\text{Q})$. We use Adam optimizer to find ϕ^* by computing the gradient $\nabla_\phi \text{Q}(\phi)$. This involves backpropagating gradients through the loss function.

Similar to [2], we evaluate the quality of the optimized design ϕ^* using the ground truth objective $\text{Q}(\phi^*)$.

2.3 Setup

The setup of our optimization is a channel with a hot wall at the bottom. The particles pick up the heat and gain temperature as the fluid flows over this wall. The goal is to optimize the wall shape to maximize the heat outflow between the inlet and outlet of the

channel. The degrees of freedom for the design problem are given by the surface structure on the flat plate. More precisely, the degrees of freedom ϕ are the z-co-ordinates of the particles, representing the part of the channel, which is heated as seen in Figure 3, where the position of the red particles is referred to as the hot wall. All SPH visualizations are created using the open source software, ParaView [16] for which a brief introduction is described in 6.

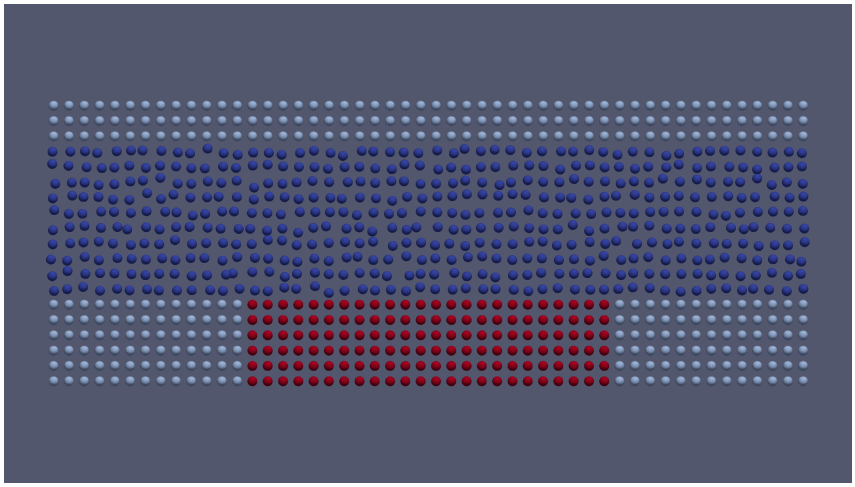


Figure 3: The vertical displacement of the red points on the bottom corresponds to the degree of freedom ϕ. The blue points are the fluid particles, and the white points represent the non-slip walls.

In the SPH solver, we need to run 2000 simulation steps for the optimization because the information of our design parameter, the wall shape, has to be transferred to the heat outflow, which is our design objective. This is also approximately the number of steps the simulation needs to reach a steady state. The time steps are bounded by the Courant-Friedrichs-Lewy (CFL) condition [9] and cannot be arbitrarily large otherwise, the simulation becomes unstable. This can potentially lead to unstable gradients.

3 Optimization using SPH Solver

In this section, we introduce the optimization of the hot wall using the SPH solver. We first discuss the intricacies of the solver, focusing on the smoothing kernel approximation, the governing equations, and the numerical schemes used. We then give an outlook of the loss function evaluation. We also introduce the techniques of gradient checkpointing and gradient accumulation we implemented. Finally, we discuss the results we obtained.

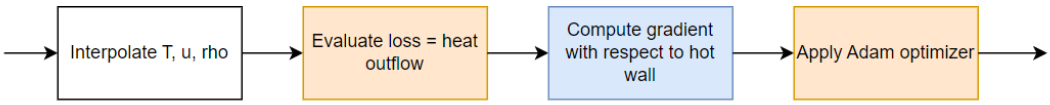


Figure 4: Simulation part of the optimization process in Figure 1 with the SPH solver.

3.1 Methodology

3.1.1 SPH Solver

SPH [1] is a technique that does not rely on grids and instead uses a smoothing kernel to estimate field properties at various points (particles) and then moves these points based on their local velocities. This method calculates spatial derivatives based on interactions with nearby particles only, thanks to the smoothing effect. Typically, the smoothing kernel has a limited range of influence to reduce the number of particle interactions, and it approximates Dirac masses to ensure accurate space discretization.

Smoothing Kernel Approximation

In the particle model, the Dirac function turns the mass in space into a superposition of multiple impulse functions. Thus, any scalar field function in space can be represented as a convolution of the Dirac kernel and itself [15].

$$A(\mathbf{r}) = \int A(\mathbf{r'})\delta(\mathbf{r}\text{-}\mathbf{r'})d\mathbf{r'} \tag{1}$$

However, we cannot use the Dirac function for simulation because the Dirac function is a generalized function and cannot be used to perform a numerical integration. We can define a continuous kernel function to approximate the process of the Dirac function so that the field can be numerically integrated in a continuous space. Hence, we use a quintic spline kernel in our project. This is a compactly supported kernel, and at each position, neighboring particles contribute to the interpolated quantities at each position.

The interpolated value of a function A at position $\mathbf{r}$ is given by [7]

$$A(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} W(\mathbf{r} - \mathbf{r}_b, h). \tag{2}$$

Then the gradient of A is obtained by taking the differential of the interpolated formula [7],

$$\nabla A(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} \nabla W(\mathbf{r} - \mathbf{r}_b, h) \tag{3}$$

The SPH solver offers convenience by eliminating the necessity for individual function values for each particle. Instead, it allows for direct calculation. For instance, when

determining mass density, we can approximate density using the mass and kernel function [7]

$$\rho_a = \sum_b \rho_b W_{ab}, \tag{4}$$

where we can see that our density estimate only relies on discrete mass and kernel function.

Governing Equations and Numerical Schemes

In our project, the equations of motion, as well as the temperature are integrated in time with an explicit Euler scheme:

$$\mathbf{v}_a^{n+1} = \mathbf{v}_a^n + \Delta t \left(\frac{d\mathbf{v}_a^n}{dt} \right) \tag{5}$$

$$\mathbf{r}_a^{n+1} = \mathbf{r}_a^n + \Delta t \mathbf{v}_a^n \tag{6}$$

$$T_a^{n+1} = T_a^n + \Delta t \left(\frac{dT_a^n}{dt} \right) \tag{7}$$

Then it remains to give an SPH momentum equation to compute $\frac{d\mathbf{v}}{dt}$ and an SPH energy equation to compute $\frac{dT}{dt}$ in each time step.

To compute discretized momentum equation, we start with the inter-particle-averaged shear viscosity (harmonic mean) given by [1],

$$\tilde{\eta}_{ij} = \frac{2\eta_i \eta_j}{\eta_i + \eta_j + \epsilon} \tag{8}$$

and the density-weighted pressure (weighted arithmetic mean) is given by [1],

$$\tilde{p}_{ij} = \frac{\rho_j p_i + \rho_i p_j}{\rho_i + \rho_j}. \tag{9}$$

The discretized momentum equation is given by [1],

$$\frac{d\mathbf{v}_i}{dt} = \sum_j \frac{\left(\frac{m_i}{\rho_i}\right)^2 + \left(\frac{m_j}{\rho_j}\right)^2}{m_i(d_{ij} + \epsilon)} (-\tilde{p}_{ij} r_{ij} + \tilde{\eta}_{ij} v_{ij}) \nabla W_{ij} \tag{10}$$

where $r_{ij} = r_i - r_j$ and $v_{ij} = v_i - v_j$. The SPH energy equation below ensures that the heat flux is automatically continuous across material interfaces [8].

$$\frac{dU_a}{dt} = \sum_b \frac{4m_b}{\rho_a \rho_b} \frac{k_a k_b}{k_a + k_b} T_{ab} \frac{r_a b \nabla_a W_{ab}}{r_{ab}^2 + \eta^2} \tag{11}$$

3.1.2 Loss Function

The loss function is a crucial metric that quantifies a model's performance by measuring the cost of its predictions. It plays a central role in guiding the model's training process. To optimize the model's parameters and improve its accuracy, the gradient of the loss function is computed, making it necessary for the loss function to be differentiable. This gradient guides optimization algorithms in adjusting the model's parameters to minimize errors. To find the best parameters for a particular task, we minimize an appropriate loss function specifically defined for that task. In the context of optimizing the hot wall's shape, our loss function assesses heat flow within the channel. This involves evaluating heat outflow in the system and using it to calculate the loss.

Heat Outflow Formulation

Various formulations exist to define the heat outflow (Q) in a channel. We evaluate Q using the formula in 12. We calculate the heat flow for each particle i and evaluate the total heat outflow via a summation over n particles. We use an interpolator to evaluate the corresponding quantities at target points. The interpolator function takes the state of an SPH simulation and interpolates a specific property to a set of target points on the mesh grid. It uses the current state of the system and target points to interpolate values for temperature (T), density (ρ), and the x-component of velocity (u_x). These interpolated values are then used to compute the heat outflow:

$$Q = \sum_{i=1}^{n} \rho_i u_{x_i} \Delta T_i \tag{12}$$

where,
ρ is the density of the particles.
u_x is the velocity of the particles in the x-direction.
ΔT is the change in temperature.

Loss Function

An iterative process simulates the system's dynamics over time, allowing for the accumulation of heat outflow effects. Within the loss function, a loop runs for a fixed number of iterations/steps, during which the system's state and neighbor relationships are updated. This represents the evolution of the system. Additionally, a smoothing factor is applied to the hot wall. By smoothing the parameters, we can reduce the generation of local optima. The algorithm used in our code to evaluate the loss function is shown in 1.

3.1.3 Gradient Checkpointing and Gradient Accumulation

In the loss function, heat outflow is computed by simulating for 2000 steps with the SPH solver and the heat flow value is obtained at the last step. We also need to compute the gradient of the loss by backpropagation. This part requires significant memory and computation resources. To reduce required memory, two memory-saving methods, gradient checkpointing and gradient accumulation are used.

Algorithm 1 Loss function evaluation

1. Determine the initial position **r** of particles based on a predefined 2D wall shape
2. Update initial position **r** using 6
4. Update the state of the system using a time evolution function as in 4 5 7, resulting in the final state of the system.
5. Define a set of target points **r_target**.
6. Interpolate the properties (u,ρ,T) at **r_target** and calculate Q using Equation 12.
7. Derive a smoothed loss function: (Smoothing factor) $*$ $(\text{hw}_{i+1})^2 - (\text{hw}_i)^2$.
10. Calculate the total loss.

Gradient Checkpointing

For a basic feed-forward neural network comprising of n layers, the computation graph used to derive gradients appears as follows:

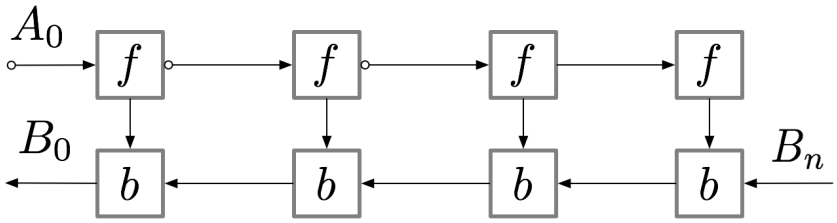


Figure 5: Computing of gradient without checkpointing.

The nodes marked with 'f' in Figure 5 correspond to the activations of neural network layers. Throughout the forward pass, these nodes are sequentially evaluated. Nodes labeled 'b' represent the gradient of the loss concerning activations and parameters of these layers. During the backward pass, these nodes are assessed in reverse order. Results obtained for the 'f' nodes are essential for computing the 'b' nodes; thus, all 'f' nodes are retained in memory following the forward pass. An 'f' node can only be erased from memory once backpropagation has advanced enough to compute all dependencies or children of that 'f' node. Consequently, the memory required by simple backpropagation grows linearly with the number of neural net layers, n. The order in which these nodes are computed is illustrated below, with purple shaded circles as seen in Figure 6 indicating which nodes need to be stored in memory at any given time.

Simple backpropagation, as described above, is optimal in terms of computation since it evaluates each node only once. However, by allowing nodes to be recomputed, significant memory savings are possible [6]. For instance, we could simply recalculate every node from the forward pass whenever required. The order of execution and the associated memory usage would then appear as seen in 6:

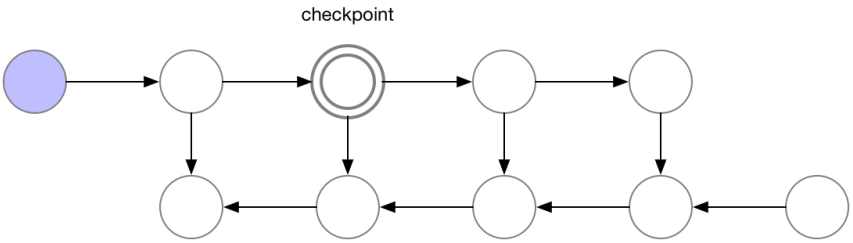


Figure 6: Computing of gradient with checkpointing.

By employing this strategy [6], the memory necessary to compute gradients in our graph remains constant with respect to the number of neural network layers, n, which is memory-optimal. However, it is worth noting that the number of node evaluations now scales with n^2, compared to the previous linear scaling of n: each of the n nodes is recomputed approximately n times. Consequently, the computation graph becomes substantially slower to evaluate for deep networks, rendering this method impractical for deep learning. To achieve a balance between memory and computation, a strategy allowing nodes to be recomputed, but not excessively, is required. The approach adopted here involves marking a subset of the neural net activations as checkpoint nodes.

Gradient computation with checkpointing

1. We set two checkpoints, bubble 7 and bubble 3 on the first row of the graph.
2. The forward propagation has been completed and the backpropagation begins as seen in Figure 7, i.e., the backpropagation starts from bubble 1 in the lower row.
3. We come to bubble 2 in the lower row, which relies on bubble 3 above to calculate (recall that the backward propagation calculation requires the output of the forward calculation). Bubble 3 is the checkpoint, which exists in memory, so the backpropagation is performed normally.
4. We come to bubble 4 in the lower row, which relies on bubble 5 on the top to calculate. Bubble 5 is not a checkpoint, not in memory, and needs to be recalculated by the checkpoint in front of it, that is, from bubble 7. Calculate a new checkpoint, and delete the original bubble 5 on the top row because it is no longer needed.
5. The new bubble 4 below is calculated to continue the backward calculation.

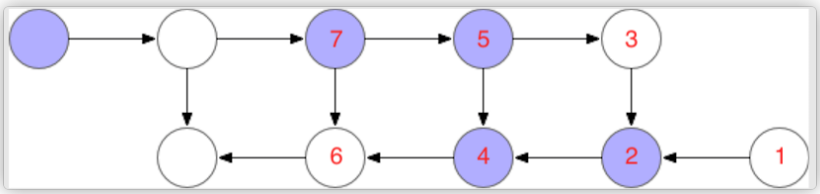


Figure 7: The process of gradient checkpointing.

Gradient Accumulation

If the memory requested by the model exceeds the actual RAM size of the device, an Out of Memory error is reported. This can also happen when many gradients need to be stored for backpropagation, which is the case in our project. One solution is to use gradient accumulation, in which the gradients are aggregated or summed together in mini-batches before updating the model parameters as seen in 8.
Gradient accumulation, as described [11], involves accumulating gradient values computed multiple times and then updating the parameters collectively. In scenarios with large datasets, such as ours, where numerous gradients are involved, insufficient memory necessitates dividing it into multiple smaller mini-batches. The gradients computed repeatedly are accumulated, after which the parameters are updated. In short, extending the training time in exchange for being able to track many gradients.

When using global batch training, the parameter update formula is:

$$V_t = V_{t-1} - \text{learning rate} \times \text{gradient}. \tag{13}$$

When using gradient accumulation the parameter update formula becomes:

$$V_t = V_{t-1} - \text{learning rate} \times \sum_i \text{gradient}_i. \tag{14}$$

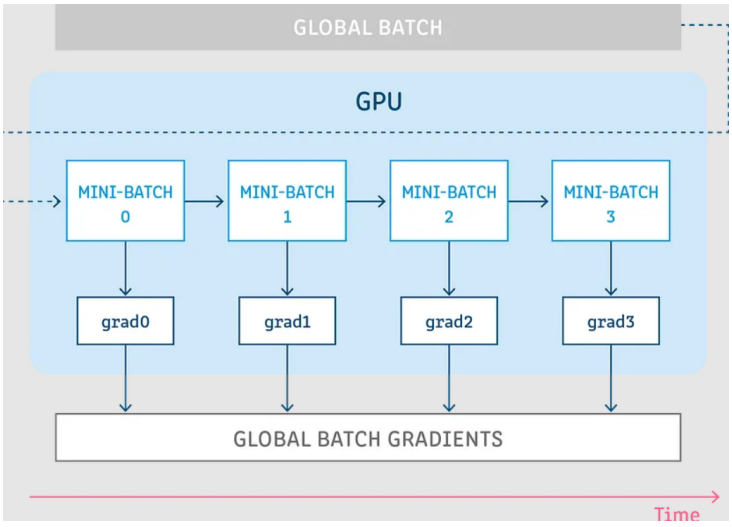


Figure 8: Gradient accumulation.

3.2 Results

We performed optimization using the SPH solver. The overview of the approach can be seen in Figure 4. In the training process, we first draw the displacement from a uniform distribution from -0.01 to 0.01. Then the hyperparameters are set as seen in table 1:

Table 1: Table of hyperparameters.

Hyperparameter	Value
Learning rate	2e-2
Smoothing factor	5e-2
Simulation steps	2000
Batch size	200
Epochs	10

The heat outflow and hot wall shape are recorded in each Adam optimization step, and plots of them are as follows:

Case 1: Optimization with the flat initialization

The process of optimization is presented in Figure 17 and a simulation of the heat outflow with and without gradient accumulation is presented in Figure 9.

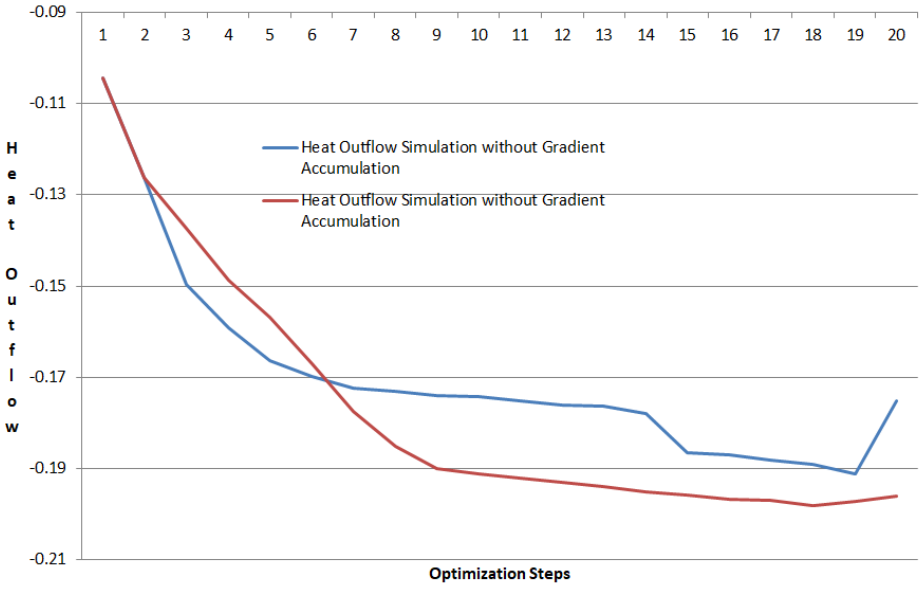


Figure 9: Simulation of the negative heat outflow with and without gradient accumulation.

From the result, we can see that we obtain a more stationary and accurate optimization with gradient accumulation. In our experiment, the time spent without gradient accumulation is 467.57 s. However, the time spent with gradient accumulation is 1034.42s. This is observed because although intermediate values are saved, more intermediate steps are generated in gradient accumulation. In the following cases, we will only do experiments and show the result with gradient accumulation because the convergence behavior with gradient accumulation is better.

Case 2: Optimization with a randomized initialization

The process of optimization with a randomized initialization is presented in Figure 18. It shows an oscillation in the beginning steps and then convergence to a "W"-like shape.

Case 3: Optimization with a monotonically increasing initialization

The process of optimization with a monotonically increasing initialization is presented by Figure19. In this case, the heat outflow still decreases with an oscillation during the optimization, and the result of the wall shape is that there is a higher inclination on the right side of the hot wall compared to the left.

Case 4: Optimization with sine function initialization

The process of optimization with a sine function initialization is presented in Figure 20. The graph of the heat outflow simulation presents that the convergence is not as good as that in other cases. From the plotted intermediate and final hot wall, we can see that the shape still remains a sine shape between the two ends, and the amplitude is becoming smaller. From these results, we can conclude that the initialization of the hot wall shape has a big effect on the resulting optimized shape and that our optimization is limited to some degree.

Lastly, a comparison of heat outflow simulation during the optimization process with different initializations is shown in Figure 10. During the initial optimization stages,

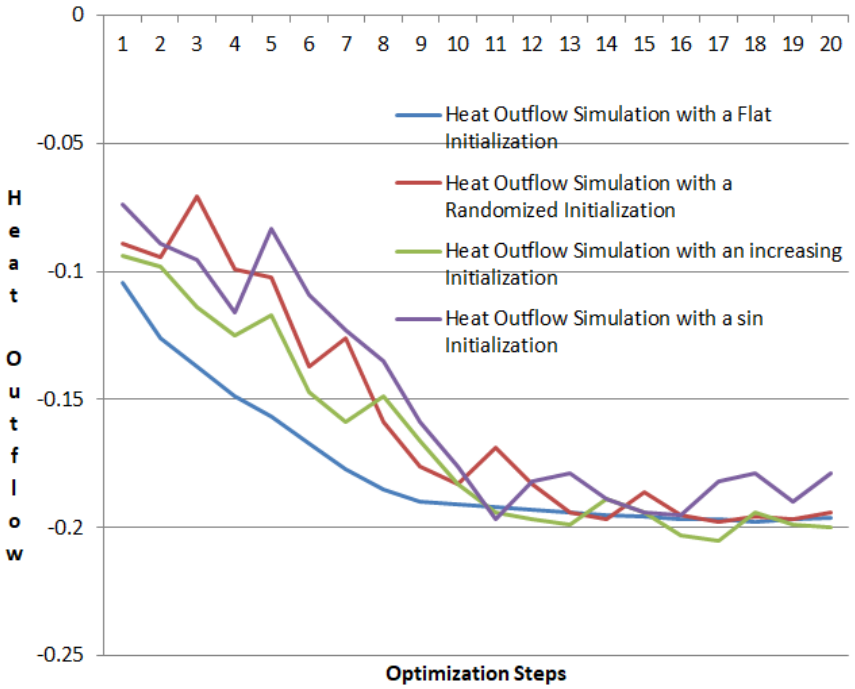


Figure 10: Simulation of heat outflow with different initializations.

simulations exhibit a monotonically decreasing trend when initialized with a flat configuration. However, in cases with alternative initializations, oscillations occur initially. This

discrepancy may arise from significant variations in vertical displacement across different points, leading to the smoothed loss function becoming influential and competing with the heat outflow component of the loss function. The parameter ϕ targeted for optimization converges to a shape resembling the letter "W" across various scenarios.

For all our initialization cases, we achieved approximately double the heat outflow of our channel, as seen in figure 10, which is commendable. Especially as we have very different initial shapes and also because the resulting shapes do not all look similar.

The theoretical best shape for such a problem was outlined in [14] and matches with the W shape we observed although not with the results from the sine function and the linearly increasing function. The simulation with flat and randomized initialization reaches the theoretically optimized surface shape. Considering that the simulated graph with flat initialization is much more stationary, we conclude that the flat wall gives the best optimization.

4 Optimization using Surrogate Model

We not only used the SPH solver to optimize the hot wall, but we also trained a surrogate model of the SPH solver to potentially obtain more stable gradients for the optimization process. In this chapter, we first discuss the theoretical background of the surrogate model, then explain how we generated the training data for the surrogate model and how we trained our model using it. The resulting model is then verified and analyzed. Finally, the process of optimization with the surrogate model is outlined.

4.1 Surrogate Model of the SPH solver

4.1.1 Graph Network-based Simulator

Our work focuses specifically on thermal and fluid flow physics while providing a machine learning model applicable to the Lagrangian method. Lagrangian machine learning has evolved to the point that now we are able to leverage graph-networks (GN) formalism, improving the network architecture and achieving great performance on control tasks. The work is an extension of the implementation of the Lagrangian method by [21] to fluid systems to predict thermal data along with dynamic motion data.

Before understanding the Graph Network-based simulator(GNS), we would like to formalize the learning problem. The task at hand is the autoregressive prediction of the next state of a Lagrangian flow field. SPH involves particles moving in tandem with local flow velocities, embodying a genuine Lagrangian approach. Lagrangian Machine Learning has evolved thus allowing us to leverage graph-based representations like Interaction Networks and hierarchical versions, especially through graph networks (GN) [20].

GNNs were chosen for Lagrangian machine learning due to their innate suitability for point cloud data, facilitating the representation of relational information between nodes and local neighborhood interactions. The GNS model, a popular surrogate model, utilizes

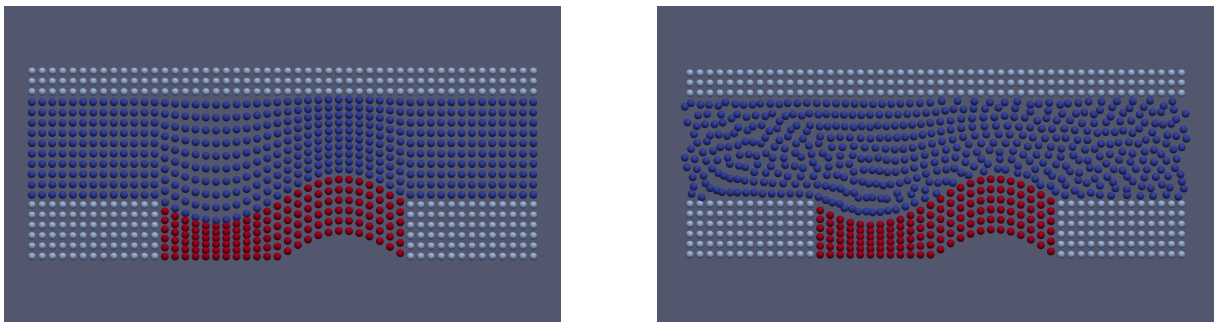
an encoder-processor-decoder architecture. GNS has shown effectiveness in 2D and 3D particle systems modeling, especially with generalized cases created by adding Gaussian noise to data [21].

4.1.2 Training Data Acquisition from Numerical Simulation

To train the surrogate model of numerical simulation, sufficient training data must be available. In our case, we use the SPH solver to generate training data. Initialization with a different boundary condition, different ways of defining this boundary condition, and constraints imposed on it are discussed below.

Hot wall Shape Adjustment

The original SPH implementation, which was based on [1] did not allow custom initialization of the hot wall boundary. We have modified the code to enable the shape of the hot wall boundary to be changed. One possibility we explored is to change the tags of particles, which consist of the moving fluid particles, the hot wall, and the non-slip wall, according to the shape of the hot wall. However, this results in a different total number of fluid particles. JAX [5] is a functional numerical computing library and needs to know the size of the data set, i.e. the number of particles, beforehand. To exploit the full computational potential of JAX, we found a way to keep the number of all particles constant for different wall shapes. The solution is to change the position of the fluid particles according to the hot wall offset. Specifically, for a positive hot wall offset, the channel becomes thinner, and the same number of fluid particles are distributed over a shorter line, as can be seen in figure 11a. This results in an initial configuration that is not physical. Fluid particles have less density when moving faster in smaller enclosures, which is the opposite of what happens with this initialization. However, after the first few frames, the numerical simulation reaches a physically meaningful state, see figure 11b. Therefore, the first 200-300 steps of the SPH simulation are not used to train the GNS model.



(a) (b)

Figure 11: Example of initializing the hot wall with a sine function. The blue particles are the fluid particles, the white particles represent the non-slip, and the red particles represent the hot wall. In a) the particle position directly after initialization is shown, and in b) the fluid particles are shown after the fluid particles are properly distributed.

Definition of the Hot Wall Shape

There are multiple methods for defining the configuration of the hot wall. One approach involves choosing from a predefined set of 10 distinct functions. These functions have been specifically designed to cover a variety of patterns, primarily aligning with trigonometric forms in line with Fourier series principles. The first two functions ensure a consistent hot wall, one with no offset and the other with a maximum offset. Following that, there are seven functions that incorporate variations of sine and cosine waves at different frequencies, in addition to one linear function.

To enhance flexibility in defining the hot wall, an additional method has been incorporated. This method enables direct manipulation of the hot wall array within the codebase itself or by passing the array as a script parameter. Moreover, recognizing the necessity for an ample amount of training data, an automated generation of diverse hot wall shapes was implemented. Specifically, we employ a uniform distribution to determine each point along the hot wall randomly. Consequently, the various points are entirely independent of one another.

Constraints on the Hot Wall Shape

A constraint is imposed on the offset to ensure minimal interference of the hot wall shape with fluid flow. This constraint defines maximum and minimum values for the offset. Another potential issue, particularly with a pseudo-randomly generated hot wall, is an excessive fluctuation that might challenge the SPH solver. As detailed in 3.1.1, the SPH solver requires a radius of 3 particles in each direction. This stipulation ensures that the transition between consecutive particles defining the hot wall is limited by the size of two particles.

Composition of the Training Data

- Dataset generated for training: Comparing our scenario to [21], our case is similar to 2D TGV case. In our case, we have different shapes of the hot wall (i.e., initial conditions or trajectory count) and a trajectory length that includes data from the steady-state simulation. Basing our scenario similar to 2D TGV, we decided on having around 200 trajectory counts, i.e., train = 100, valid = 50, test = 50 and a trajectory length of 147.

Table 2: Datasets overview: Trajectory Count is used to split into training, validation, and testing datasets.

Dataset	Particle Number	Trajectory Length	Trajectory Count	$\Delta t \times 10^{-3}$	$\Delta x \times 10^{-3}$	$L \times H$
2D TGV	2500	126	100/50/50	40	20	1×1
2D HT	950	147	100/50/50	40	20	1×0.38

- Data pre-processing: This process includes the computation of important statistical

metrics like mean and standard deviation for parameters including position, velocity, acceleration, temperature, and temperature difference. Thus, the raw data obtained from the SPH solver is transformed into structured sets for train, valid, test datasets. These datasets are in H5 file formats. This approach ensures the integrity and richness of the dataset, allowing ease in modeling and analysis.

4.1.3 GNS Surrogate Modelling with LagrangeBench

In our SPH study comprising thermal and fluid dynamics, the machine learning task involves predicting future positions and temperatures of fluid. To perform the machine learning task, we utilize the popular package of LagrangeBench [19] available for predictions of position for fluid and extend it to predict temperatures as well [21]. This problem formulation requires leveraging an input of some number of initial trajectories, denoted as input-seq-length(typically set to 6) to predict subsequent time steps, denoted by extra-seq-length (default value set to 20). In short, we use the initial 6 trajectories to predict the positions and temperatures of the next 20 steps.

As in Lagrangian methodology, fluid particles are considered individually, we employ the Euler integrator function or equations of motion as in 15 to extrapolate motion dynamics for position predictions. This entails using GNS architecture, which utilizes velocity, boundary information, and forces as node features and relative distance and displacement as edge features to predict accelerations. Subsequently, these predicted accelerations are utilized to compute the particle's position, and a comparison is drawn to ground truth values to compute training and validation losses.

$$\mathrm{v} = \frac{dx}{dt} \qquad \text{and} \qquad \mathrm{a} = \frac{dv}{dt} = \frac{d^2x}{dt^2}. \tag{15}$$

Similarly, the task extends to predicting temperatures within SPH. Here, the network utilizes velocities and temperature difference magnitudes as node features and, again, relative distance and displacement as edge features to predict the temperature differences. Mirroring the methodology used for position predictions, we utilize the fundamental heat equation as in 16 to compute the effects of advection and diffusion on temperature iteratively using predicted velocities and temperature differences.

$$\frac{\partial T}{\partial t} + (\mathbf{u} \cdot \nabla)T = \alpha \nabla^2 T. \tag{16}$$

We chose mean square error(MSE) as in 17 to test the performance of the model. While training the model, we used a combined MSE loss. On the other hand, we observed separate validation losses during validation to provide us with results of position and temperatures.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{17}$$

4.1.4 Results

For training the data, we use our train data set, which comprises 100 distinct trajectories, each spanning a trajectory length of 147 discrete time steps. Drawing inspiration from the seminal work of [21], we begin calibrating our hyperparameters for the training case similar to 2D TGV simulations. The following are the important hyperparameters set:

- Learning rate: Started at 10e-4, decay rate as 0.1 and final learning rate as 10e-6
- Number of MLP layers: 2
- Latent Dimension: 128
- Standard deviation for additive noise: 0.0003.

We set the maximum steps of training iterations to 500000, where we meticulously log training error at every 1000^{th} step. Simultaneously, we evaluate the evolution of validation loss at every 10000^{th} step. This methodology lets us discern the optimal duration required to train our model effectively all while avoiding the chances of overfitting the data. From the intricate analysis presented in Figure 12, Figure 13, and Figure 14, we extrapolate the duration for training. Our twofold objective of mitigating the risk of overfitting while attaining the most favorable validation loss ensures the selection of the surrogate model.

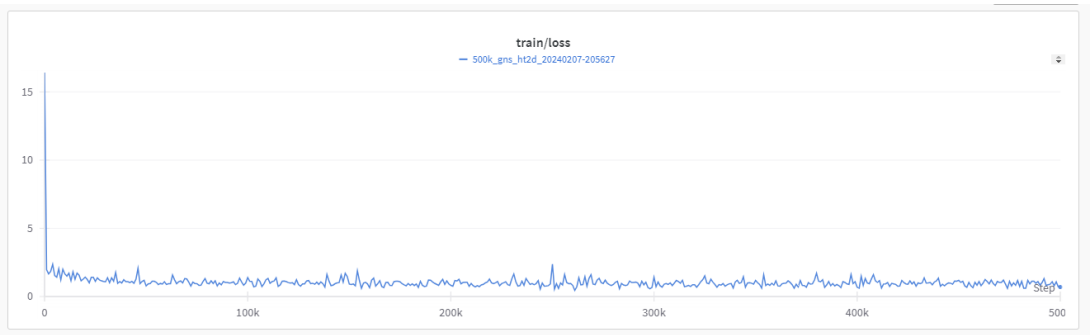


Figure 12: Training loss curve for maximum steps = 500000.

4.2 Analysis and Visualization of Test Data

Following the training phase, we utilized the best model to infer the test dataset. This allows the visualization of results in ParaView. Notably, the simulated trajectory derived from the predicted results exhibits a commendable semblance to ground truth within the initial temporal frames (0-15). However, an appreciable disparity between predicted and ground truth trajectories emerged during subsequent time frames(15-25). This discrepancy required a comprehensive exploration of various factors. This can be seen from the comparison of predicted and ground truth trajectory images in Figure 25 from 6
Among the considerations, foremost, was the phenomenon of fluid re-circulation for simulation and its effects on prediction values. An important factor in temperature prediction is the methodology used to compute temperature from GNS-predicted accelerations and

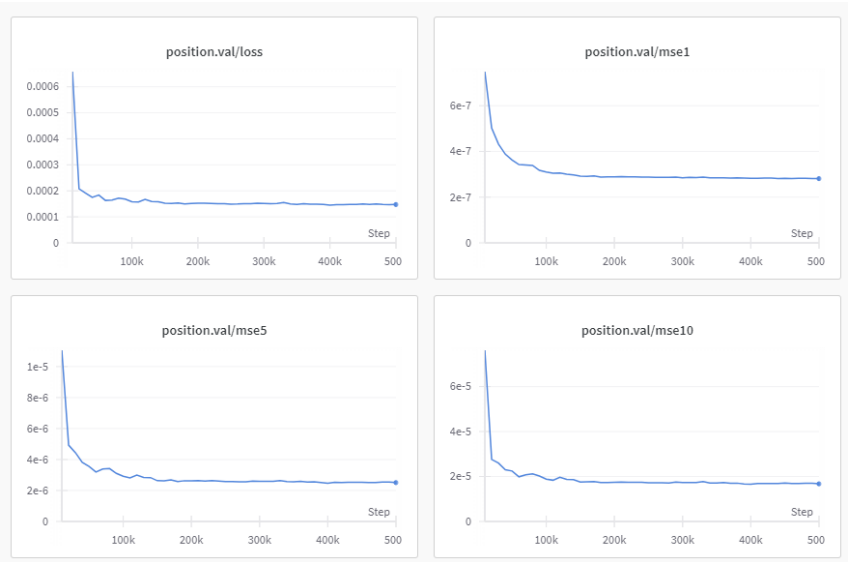


Figure 13: Position validation loss to determine the number of maximum steps using MSE, MSE1, MSE5, MSE10 at 1, 5, 10, and 20th steps.

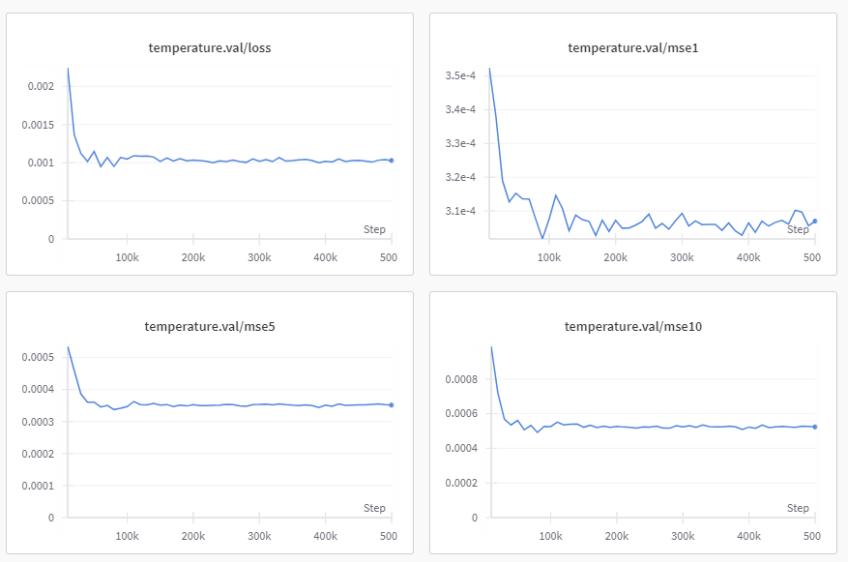


Figure 14: Temperature validation loss to determine the number of maximum steps using MSE, MSE1, MSE5, MSE10 at 1, 5, 10, and 20th steps.

temperature differences. We see the effect on validation loss using different weights for advection and diffusion.

Hence, further work in the form of utilizing other features for prediction in the GNS network or a better methodology to compute temperatures will allow us to simulate the behavior of fluid in a much better way.

4.3 Analysis of Hyperparameters

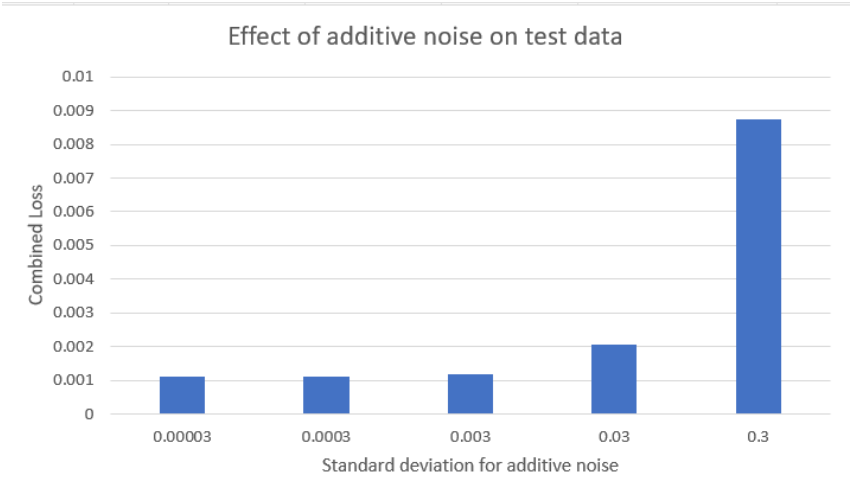
We further analyze the effects of hyperparameters like latent dimension and standard additive noise used in our model. Figure 21 and 22 show hyperparameter changes' effects on position and validation losses. The effect of additive noise is much more evident or easy to compare when it comes to position validation loss compared to temperature validation loss, which can be observed from Figure 21 and Figure 22 in 6. But we can compare the effects of standard deviation for additive noise using a combined validation as shown in Figure 15a. The change in the loss is less when the standard deviation for additive noise is very low (e.g., std-dev: 0.00003), while when it is increased to, say, 0.3, we see a sudden increase in loss for both position and temperature. The changes are much more evident in position loss than temperature loss for low noise values.

Also, we observed the effects of latent dimension changes on the loss. For a low value of a latent dimension, i.e., 32 in our case, we get a higher loss initially than when we have higher values like 128 and 256, as seen in Figure 15b. Additionally, we observe that the loss in each case (changes in latent dimension values) converges to comparable values. It is just the maximum number of steps that are required for changes. This is evident from Figures 23 and 24 in 6. After comparing the models with test data, we decided on using latent dimension 128 for our model.

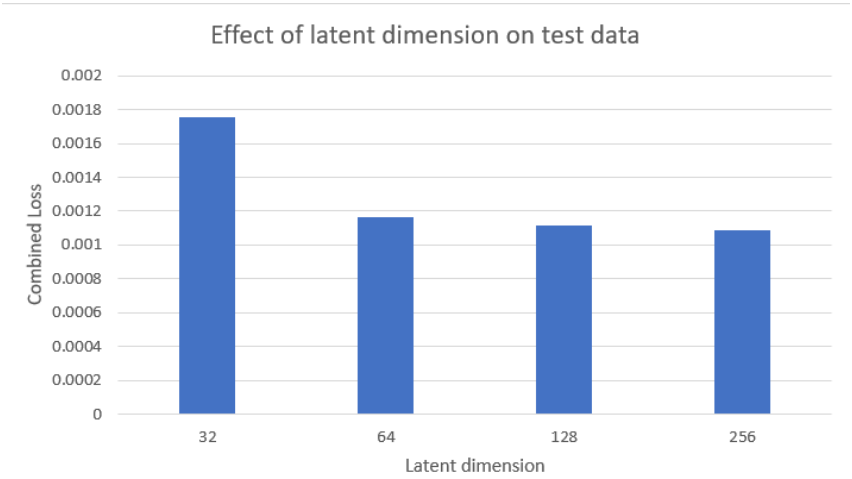
4.4 Optimization using the Surrogate Model

Integrating the surrogate model into the optimization process involves several steps. Figure 16 provides an overview of these steps essential for using the surrogate model within our optimization framework.

Initially, the SPH solver simulation must run for about 300 steps to ensure that equilibrium is reached. This is particularly important given the absence of this non-physical initialization step in our training data. This preliminary step is essential for traditional SPH optimization and the surrogate model approach. Subsequently, additional simulation steps are performed to generate sufficient data for initialization. This data is then stored in a VTK file and processed similarly to the surrogate model training data. Specifically, the H5 files from the rollouts are merged into a single H5 file to form a comprehensive dictionary. The simulation is ready to run with this prepared H5 file and the trained surrogate model. Notably, only 20 simulation steps are required for the surrogate model, which is equivalent to 2000 SPH simulation steps.



(a) Effect of additive noise on test data



(b) Effect of latent dimension on test data

Figure 15: Effect of hyperparameters: standard deviation of additive noise and latent dimension on test data.

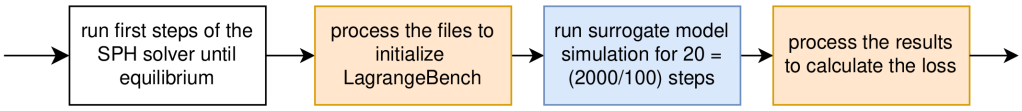


Figure 16: Simulation part of the optimization process for the surrogate model.

Once the inference is complete, the results are saved and processed further to ensure compatibility with the existing loss calculation from the optimization code. While these preparatory steps have been completed, problems remain with the gradients and the optimization process.

In theory, with this processed data, we should be able to calculate the loss function and execute the optimization. Our attempts were unsuccessful as we encountered an error that we were unable to interpret at this juncture.

5 Conclusions

Summary of Results

In this project, we implemented two methods to optimize the shape of a hot wall in a fluid channel. The main objective is to obtain an optimal shape of the hot wall to maximize the heat outflow. One method covered the implementation using a Smoothed Particle Hydrodynamics (SPH) solver using methods put forward by [2] and [1]. The second approach involves training a surrogate model of the SPH solver to obtain more stable gradients for the optimization process.

Regarding optimization with the SPH solver, we performed the optimization with four different initialization cases: with a flat, randomized, monotonically increasing, and using a sine function. We can conclude that a similar surface shape is obtained with varying initialization cases, and it presents a convergence in optimization steps. However, the time steps are bound by CFL number, which binds the stability of the simulation, consequently leading to unstable gradients. We used gradient checkpointing and gradient accumulation to manage memory and computational resources. Results were compared with and without gradient accumulation. The optimization with gradient accumulation shows improvements in stability and accuracy.

The idea of this project was, as suggested by [2], this project's idea was to use a surrogate model of the numerical solver in addition to the numerical solver to have more stable gradients. This surrogate model is a Graph Network-based Simulator (GNS) trained on numerical simulations of the SPH solver with different hot wall shapes. Only every 100^{th} frame of the rollout is used for the training to ensure easier backpropagation of gradients. During training, the surrogate model shows the reduction of training and validation loss with the number of steps as expected from a machine learning model. However, there are still some discrepancies between the trained model and the ground truth. This will probably require more feature engineering and trying out other models. Integrating the surrogate model into the optimization process involves several steps. While we have successfully completed replacing the SPH solver with the surrogate model in our implementation, we have encountered some mistakes in the gradient optimization that, unfortunately, we have not been able to resolve.

Future Work

Regarding optimization with the SPH solver, further experiments can be conducted. One potential area for extending the project involves applying the methodologies outlined in this report to other domains or expanding its scope to three-dimensional simulations.

Regarding the surrogate model, there are a few ways the model could be enhanced. Firstly, more hyperparameter training of the GNS and other integrate functions(function name is as used in code) can be adapted for the rollout. Another idea is to replace the trained temperature modeling with an analytical expression similar to the one in the SPH solver. This would make the inference more stable. Also, instead of using a standard GNS model, better GNNs like E(n)-equivariant Graph Neural Networks (EGNNs) [21] or Steerable E(n)-equivariant Graph Neural Networks (SEGNNs) with Historical Attribute Embedding (HAE) [20].

The optimization procedure utilizing the surrogate model is currently experiencing debugging challenges and demands a more diagnostic approach for resolution. Because of time limitations, we directed our attention to other matters, leaving the task for future investigation and continuation from where we left off. Additionally, we can enhance optimization outcomes by modifying the GNS network within the surrogate model.

References

[1] S. Adami, X.Y. Hu, and N.A. Adams. "A transport-velocity formulation for smoothed particle hydrodynamics". In: *Journal of Computational Physics* 241 (2013), pp. 292–307. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2013.01.043. URL: https://www.sciencedirect.com/science/article/pii/S002199911300096X.

[2] Kelsey R Allen et al. *Inverse Design for Fluid-Structure Interactions using Graph Network Simulators*. 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/59593615e358d52295578e0d8e94ec4a-Paper-Conference.pdf.

[3] Utkarsh Ayachit. *The paraview guide: a parallel visualization application*. Kitware, Inc., 2015.

[4] Deniz A. Bezgin, Aaron B. Buhendwa, and Nikolaus A. Adams. "JAX-Fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows". In: *Computer Physics Communications* (2022), p. 108527. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2022.108527. URL: https://www.sciencedirect.com/science/article/pii/S0010465522002466.

[5] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax.

[6] Tianqi Chen et al. *Training Deep Nets with Sublinear Memory Cost*. 2016. arXiv: 1604.06174 [cs.LG].

[7] Paul W Cleary and Joseph J Monaghan. "Conduction Modelling Using Smoothed Particle Hydrodynamics". In: *Journal of Computational Physics* 148.1 (1999), pp. 227–264. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1998.6118. URL: https://www.sciencedirect.com/science/article/pii/S0021999198961186.

[8] Paul W. Cleary. "Modelling confined multi-material heat and mass flows using SPH". In: *Applied Mathematical Modelling* 22.12 (1998), pp. 981–993. ISSN: 0307-904X. DOI: https://doi.org/10.1016/S0307-904X(98)10031-8. URL: https://www.sciencedirect.com/science/article/pii/S0307904X98100318.

[9] Richard Courant, Kurt Friedrichs, and Hans Lewy. "Über die partiellen Differenzengleichungen der mathematischen Physik". In: *Mathematische Annalen* 100.1 (1928), pp. 32–74. DOI: 10.1007/BF01448839.

[10] Simone Sommavilla / Dive. *Smoothed Particle Hydrodynamics - A Guided Journey into the Basics of the SPH Method*. https://www.dive-solutions.de/blog/sph-basics. Accessed: 09.02.2024. 2020.

[11] Priya Goyal et al. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2018. arXiv: 1706.02677 [cs.CV].

[12] Philipp Holl, Vladlen Koltun, and Nils Thuerey. *Learning to Control PDEs with Differentiable Physics*. 2020. arXiv: 2001.07457 [cs.LG].

[13] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[14] Theodoros Leontiou, Marios Kotsonis, and Marios M. Fyrillas. "Optimum isothermal surfaces that maximize heat transfer". In: *International Journal of Heat and Mass Transfer* 63 (2013), pp. 13–19. DOI: https://doi.org/10.1016/j.ijheatmasstransfer.2013.02.078. URL: https://www.sciencedirect.com/science/article/pii/S0017931013002032.

[15] Joe J Monaghan. "Smoothed particle hydrodynamics". In: *Annual review of astronomy and astrophysics* 30.1 (1992), pp. 543–574.

[16] *ParaView*. https://www.paraview.org/. Accessed: January 2024.

[17] Alvaro Sanchez-Gonzalez et al. "Learning to Simulate Complex Physics with Graph Networks". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal DaumÃ© III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 8459–8468. URL: https://proceedings.mlr.press/v119/sanchez-gonzalez20a.html.

[18] Yang Song and Stefano Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. 2020. arXiv: 1907.05600 [cs.LG]. URL: https://arxiv.org/abs/1907.05600.

[19] Artur Toshev et al. "LagrangeBench: A Lagrangian Fluid Mechanics Benchmarking Suite". In: *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2023. URL: https://openreview.net/forum?id=8ZRAHNT7E9.

[20] Artur P Toshev et al. "Learning Lagrangian Fluid Mechanics with E (3)-Equivariant Graph Neural Networks". In: *arXiv preprint arXiv:2305.15603* (2023). URL: https://arxiv.org/pdf/2305.15603.pdf.

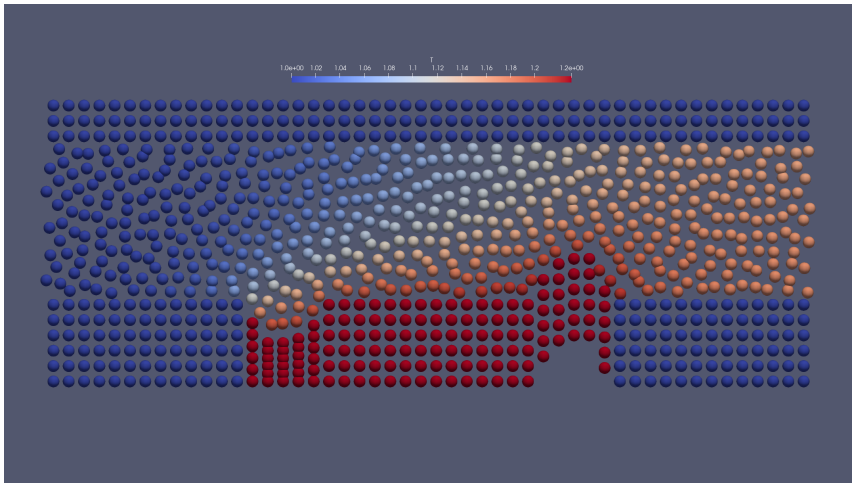
[21] Artur P. Toshev et al. "LagrangeBench: A Lagrangian Fluid Mechanics Benchmarking Suite". In: (Sept. 2023). URL: http://arxiv.org/abs/2309.16342.

[22] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386.

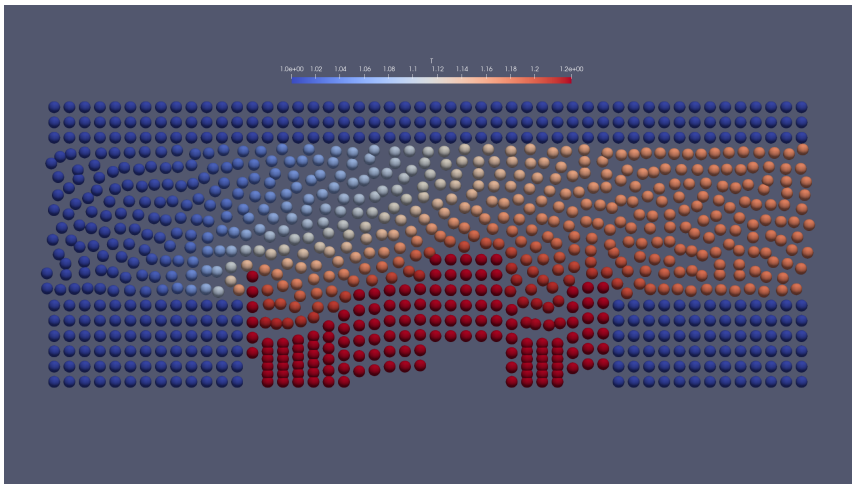
6 Appendices

The Optimization of the Hot Wall Shape with Different Initializations

This part of the appendix shows the different results of the hot wall shape optimization process. They all start with a different initial hot wall shape, which is then updated during the optimization process. For each case, the initial state, an intermediate state, and the final state are shown. All figures depict the temperature fields of an SPH simulation after 3 seconds, and the deep red dots at the bottom of the figures represent the shape of the hot wall.

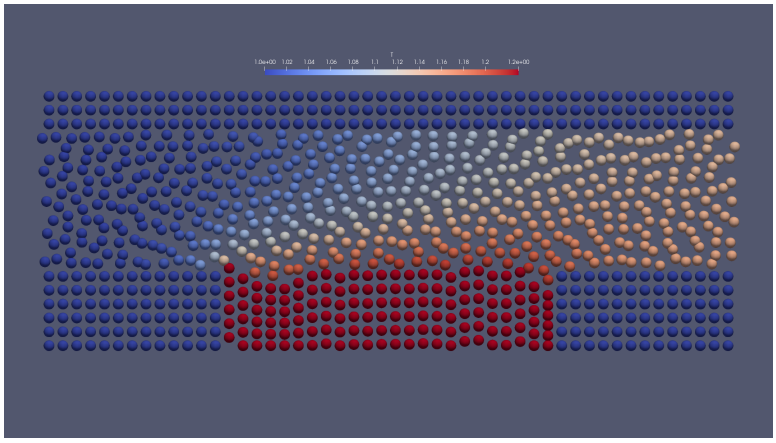


(a) Intermediate optimized hot wall shape.

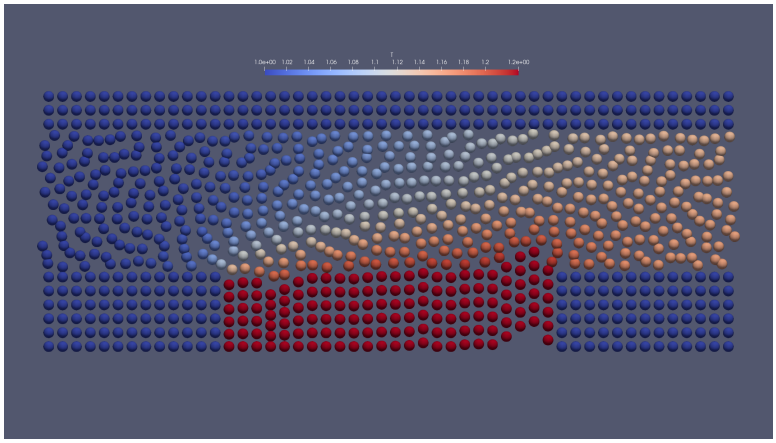


(b) Final optimized hot wall shape.

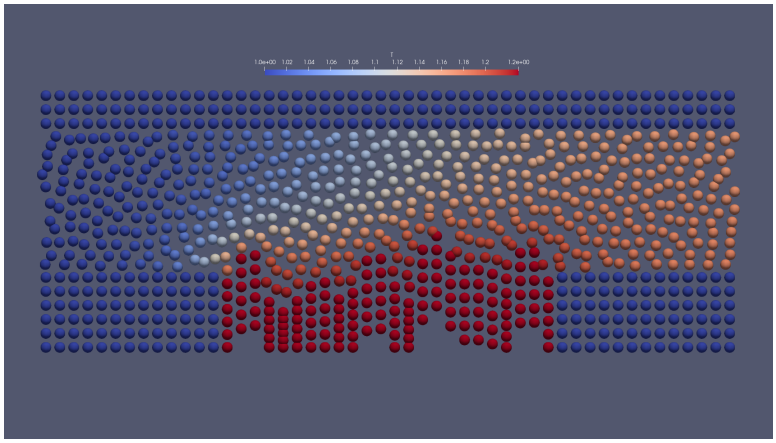
Figure 17: The optimization of the Hot Wall with a flat initialization.



(a) A randomized initialization of the hot wall shape.

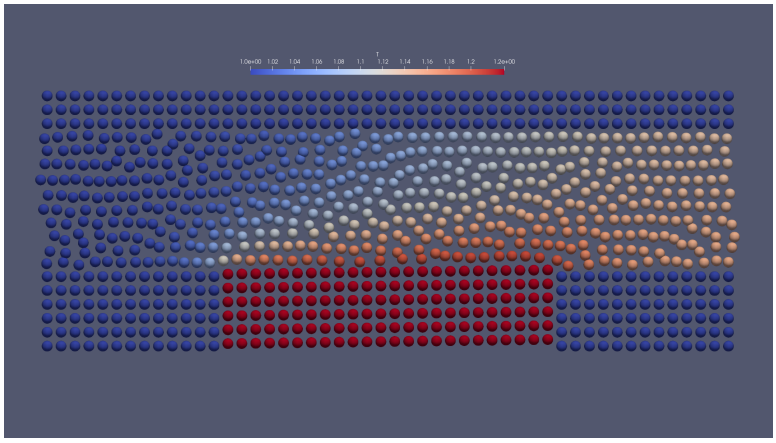


(b) Intermediate step of the optimized hot wall shape.

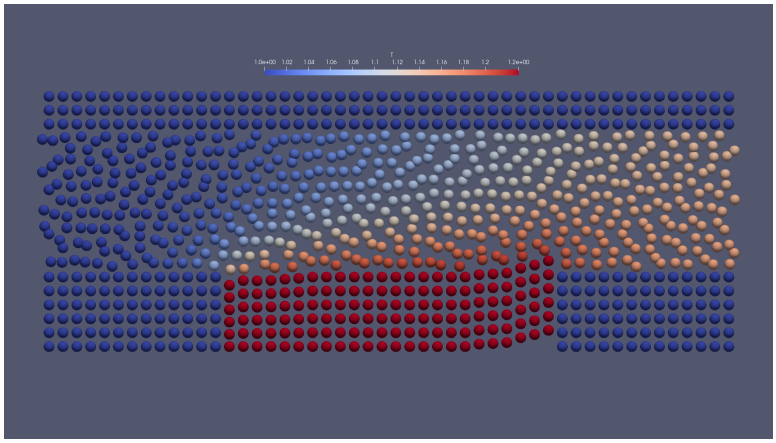


(c) Final optimized hot wall shape.

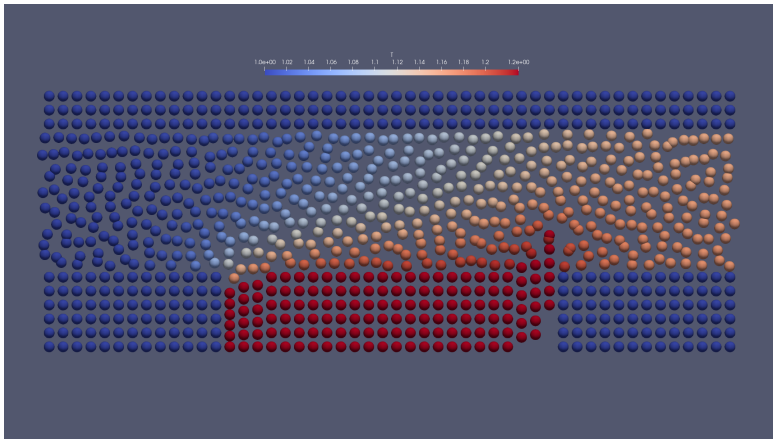
Figure 18: The optimization of the hot wall with a randomized initialization.



(a) Hot wall initialization with linearly increasing function.

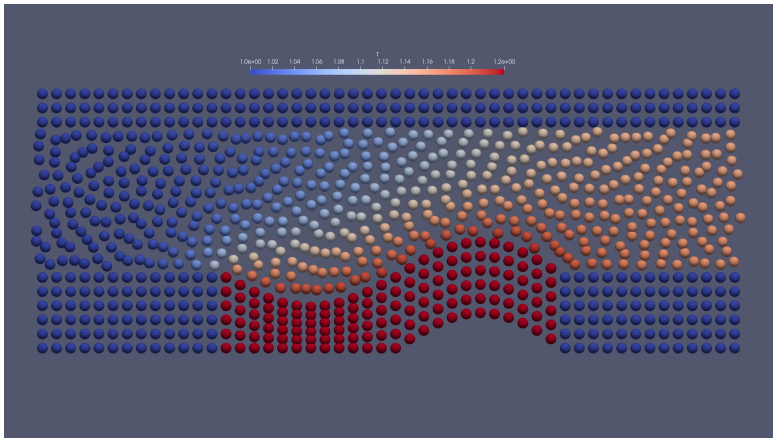


(b) Intermediate step of the optimization.

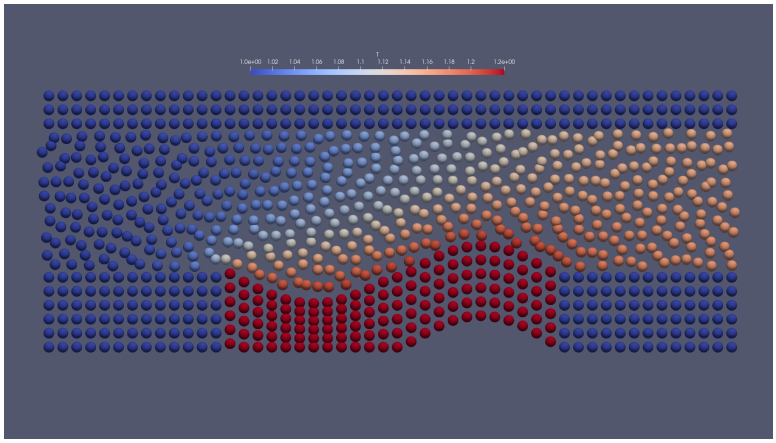


(c) Final optimized hot wall.

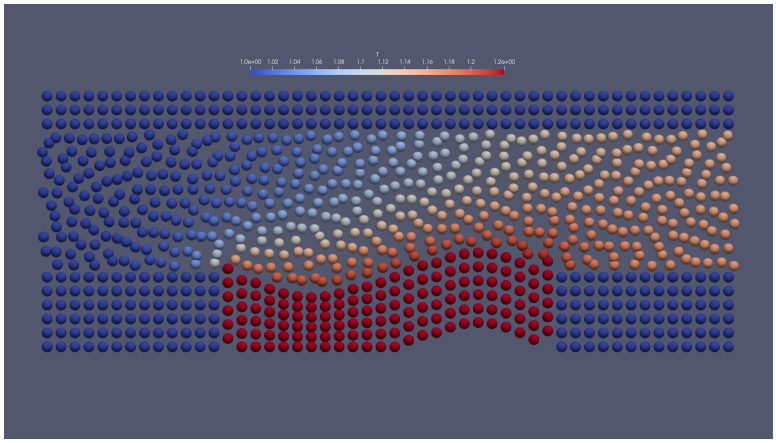
Figure 19: The optimization of the hot wall with a randomized initialization.



(a) The initial hot wall shape, which is a sine function.



(b) Intermediate optimized hot wall.



(c) Final optimized hot wall.

Figure 20: The optimization of the hot wall with a sine function initialization.

Effects of Noise Perturbations

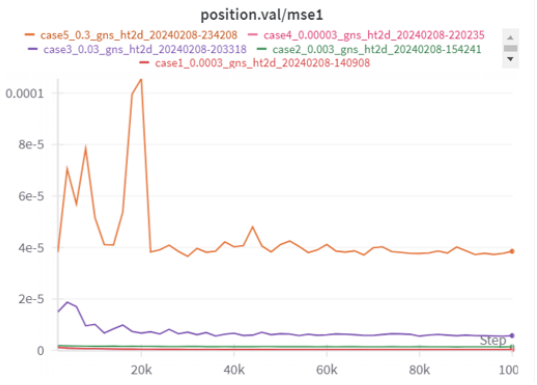
To generalize the data and overcome difficulties with low data density regions, we add some Gaussian noise or apply noise perturbations at varying scales [18].

$$S = \frac{n}{\sigma_1, \ldots, \sigma_L} \left\{ \frac{\sigma_1}{\sigma_2} = \ldots = \frac{\sigma_{L-1}}{\sigma_L} > 1 \text{ for } i = 1, \ldots, L-1 \right\} \tag{18}$$

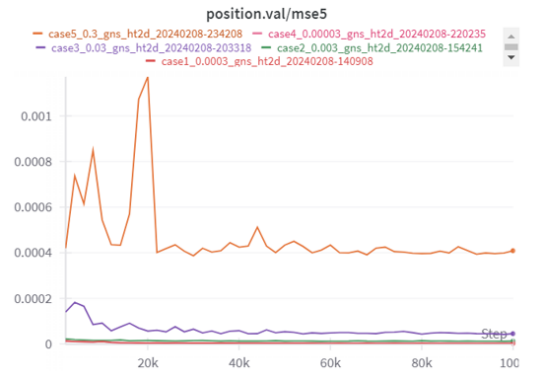
We later perturb the data by adding a Gaussian noise.

$$\tilde{x} = x + \sigma_i \epsilon_i, \quad \text{where } \epsilon_i \sim \mathcal{N}(0, 1). \tag{19}$$

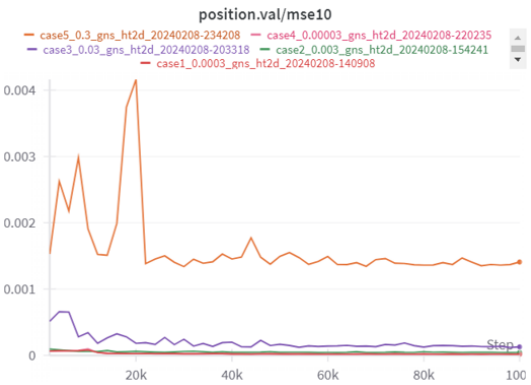
We vary the standard deviation of noise addition by multiplying it by multiples of 10, i.e., from 0.00003 to 0.3, and observe the validation losses for position and temperature and their effects.



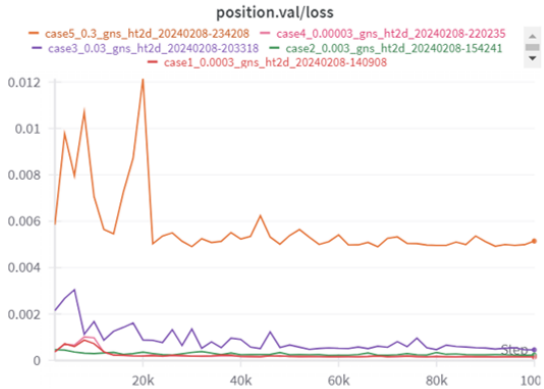
(a) Timestep 1.



(b) Timestep 5.

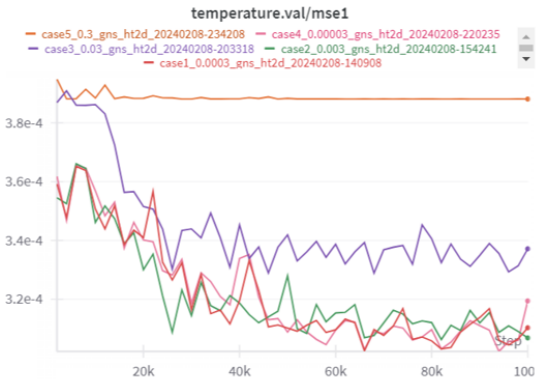


(c) Timestep 10.

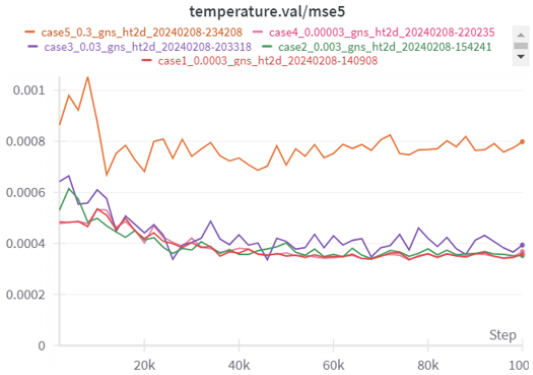


(d) Timestep 20.

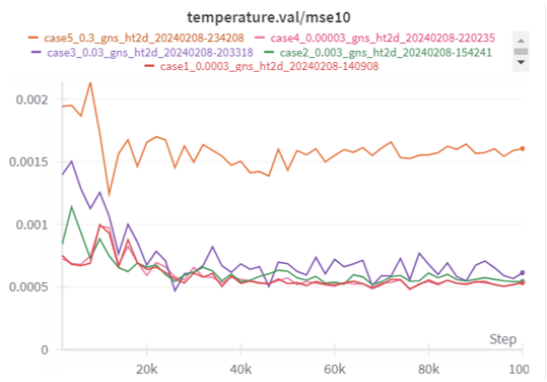
Figure 21: Effect of change in standard deviation for additive noise on position validation loss.



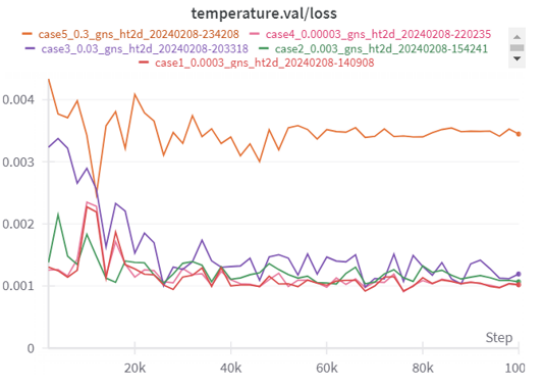
(a) Timestep 1.



(b) Timestep 5.



(c) Timestep 10.



(d) Timestep 20.

Figure 22: Effect of change in standard deviation for additive noise on temperature validation loss.

Effects of Latent Dimension

The latent dimension is pivotal in Graph Neural Networks (GNNs), particularly in models like Graph Network-based Simulators (GNS). It determines the representation capacity of the network and directly influences its ability to capture and encode essential features from the input data. In the context of GNS, a higher latent dimension allows for a more expressive feature space, enabling the model to learn intricate relationships and patterns within the graph structure.

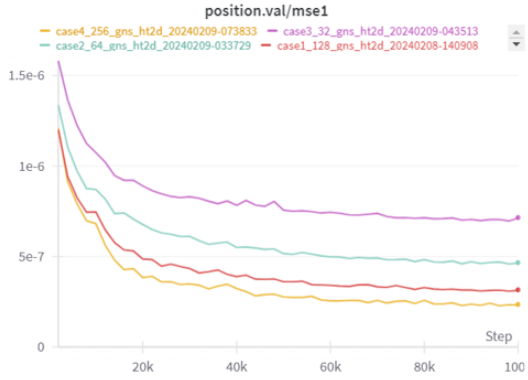
Mathematically, the latent dimension d is often incorporated into the formulation of GNS models through the dimensionality of the hidden layers. For instance, in a typical GNN layer, the hidden representation h_v for node v can be computed as seen in 20

$$h_v = \sigma\left(\sum_{u \in \mathcal{N}(v)} f(h_u, h_v, e_{uv})\right) \tag{20}$$

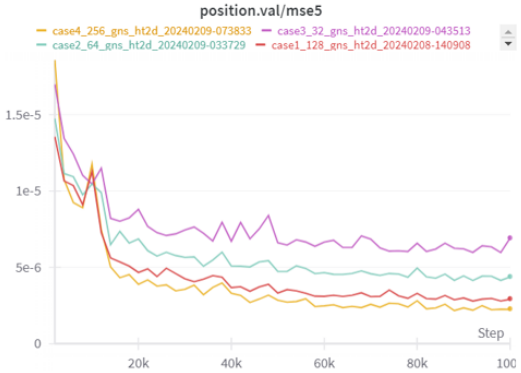
where σ is the activation function, f is the message passing function, h_u and h_v are the hidden representations of nodes u and v, respectively, and e_{uv} is the edge attribute between nodes u and v [22].

The latent dimension d is implicitly incorporated into the dimensionality of the hidden representations h_v. Specifically, the dimension of h_v is typically determined by the latent dimension d. For example, if d is chosen to be 128, then each h_v would be a vector of size 128, capturing the latent features of the corresponding node v. This latent dimensionality influences the expressive power of the GNN model, as a higher d allows for richer feature representations that can potentially capture more complex relationships and patterns in the graph data.

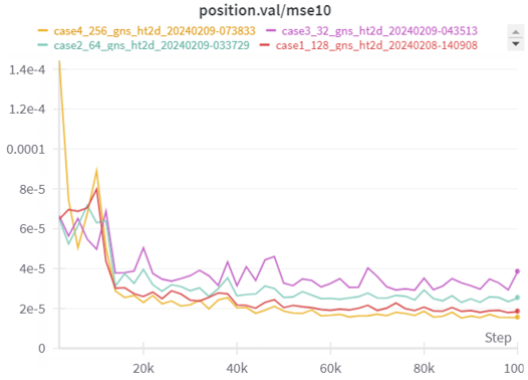
To analyze the effects of latent dimensions on validation loss of position and temperature, we consider four values of latent dimensions, namely, 32, 64, 128, and 256.



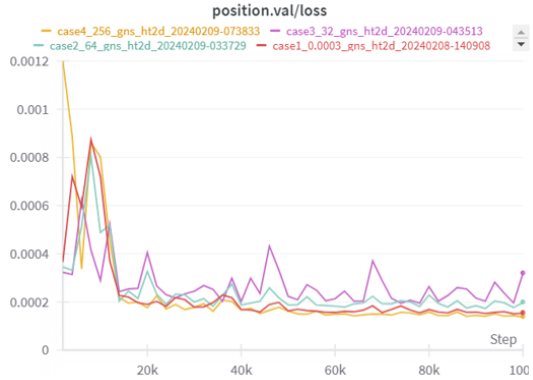
(a) Timestep 1.



(b) Timestep 5.

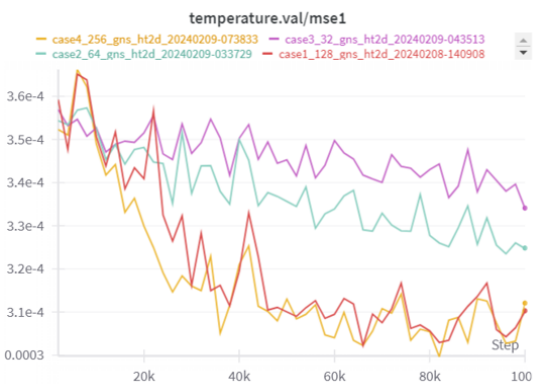


(c) Timestep 10.

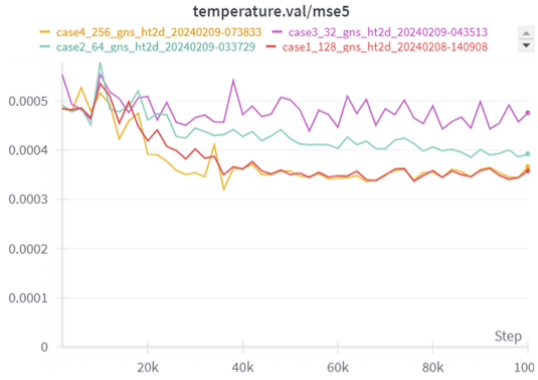


(d) Timestep 20.

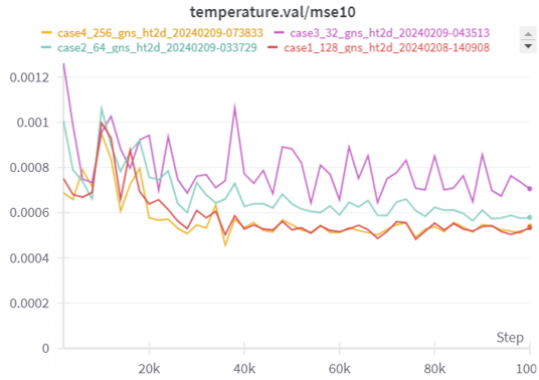
Figure 23: Effect of change in latent dimension on position validation loss.



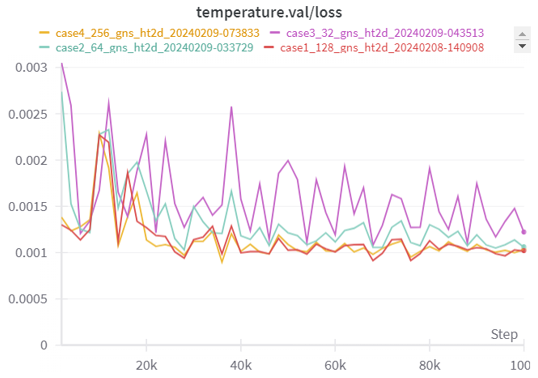
(a) Timestep 1.



(b) Timestep 5.



(c) Timestep 10.



(d) Timestep 20.

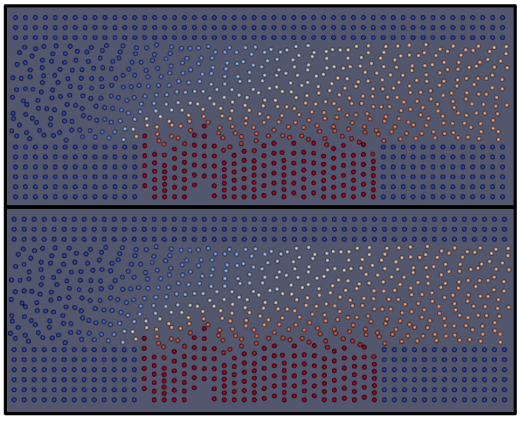
Figure 24: Effect of change in latent dimension on temperature validation loss.

Predicted and Ground Truth trajectories in ParaView

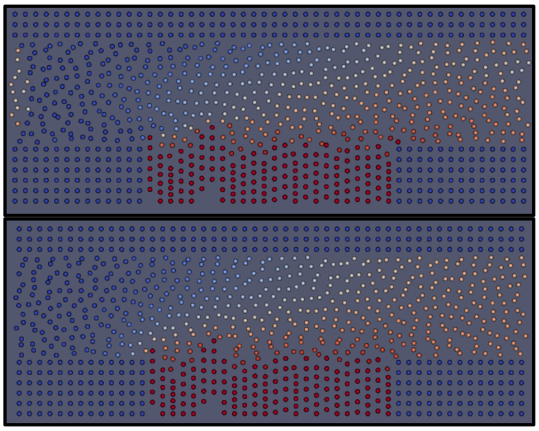
ParaView is an open-source visualization tool widely used in scientific computing and engineering disciplines to analyze and visualize large-scale datasets. Developed by Kitware Inc., ParaView offers a varied suite of features custom-made to meet the needs of researchers, engineers, and data scientists working with complex numerical simulations and experimental data.

With its intuitive graphical user interface (GUI) and powerful rendering capabilities, ParaView enables users to interactively explore and analyze data from diverse sources, including Computational Fluid d Dynamics (CFD), Finite Element Analysis (FEA), Molecular Dynamics (MD), and more. ParaView supports a wide range of data formats, including VTK, HDF5, Exodus, and others, allowing seamless and aesthetic visualization for SPH solver [16].

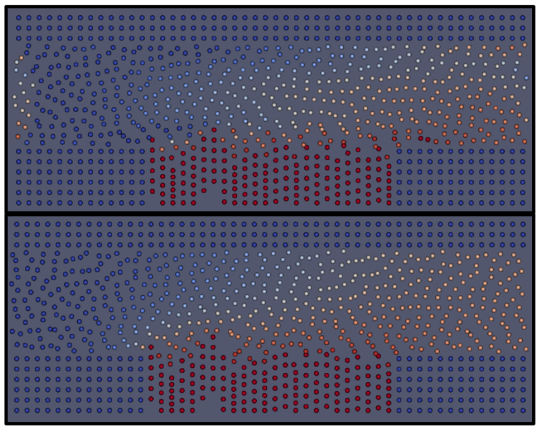
ParaView allows users to visualize key parameters such as positions, velocity fields, and temperature profiles, allowing for an in-depth understanding of the underlying fluid dynamics simulated by the SPH solver [3].



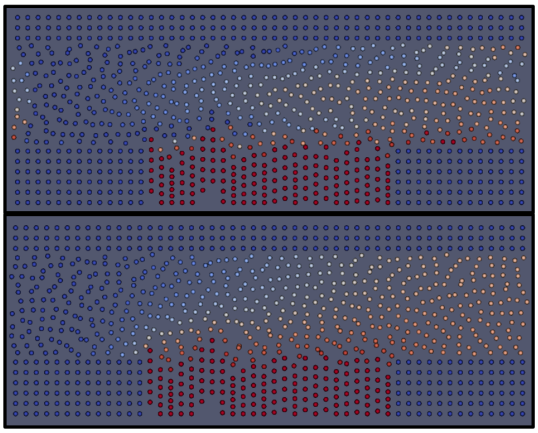
(a) Frame 5.



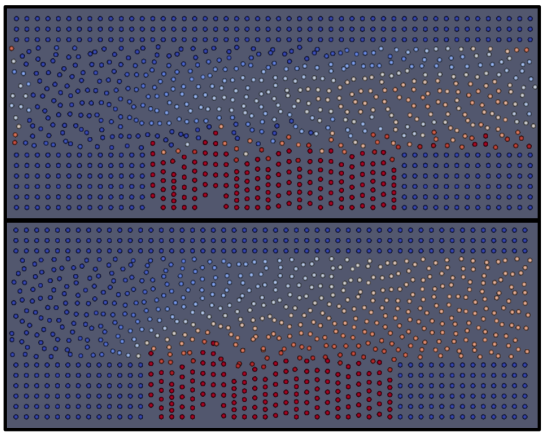
(b) Frame 10.



(c) Frame 15.



(d) Frame 20.



(e) Frame 25.

Figure 25: Comparison of predicted (top) and ground truth (bottom) trajectories in ParaView.