Surrogate-Based Differentiable Pipeline for Shape Optimization

Andrin Rehmann **Nolan Black** **Josiah Bjorgaard** **Alessandro Angioi**
Andrei Paleyes **Niklas Heim** **Dion Häfner** **Alexander Lavin**
Pasteur Labs
`{firstname.lastname}@simulation.science`

Abstract

Gradient-based optimization of engineering designs is limited by non-differentiable components in the typical computer-aided engineering (CAE) workflow, which calculates performance metrics from design parameters. While gradient-based methods could provide noticeable speed-ups in high-dimensional design spaces, codes for meshing, physical simulations, and other common components are not differentiable even if the math or physics underneath them is. We propose replacing non-differentiable pipeline components with surrogate models which are inherently differentiable. Using a toy example of aerodynamic shape optimization, we demonstrate an end-to-end differentiable pipeline where a 3D U-Net full-field surrogate replaces both meshing and simulation steps by training it on the mapping between the signed distance field (SDF) of the shape and the fields of interest. This approach enables gradient-based shape optimization without the need for differentiable solvers, which can be useful in situations where adjoint methods are unavailable and/or hard to implement.

1 Introduction

Many computer-aided engineering (CAE) problems revolve around finding an optimal geometry. Examples include structural optimization (designing lightweight yet robust structures), aerodynamic shape optimization (maximizing lift-to-drag ratio), heat exchanger design (improving heat dissipation), and acoustic optimization (reducing resonance). The underlying challenge is common across domains: given a parametrized family of shapes and a performance metric, find the design parameters that maximize (or minimize) the quantity of interest.

Traditionally, such optimization tasks rely heavily on manual iteration: an engineer proposes a design based on physical intuition, generates a mesh, runs a simulation, evaluates quantities of interest (QoIs), and iteratively modifies the *design parameters*. This workflow can be summarized as a repeated application of the following pipeline:

$$\texttt{Design Parameters} \rightarrow \texttt{3D geometry} \rightarrow \texttt{Meshing} \rightarrow \texttt{Simulation} \rightarrow \texttt{QoI}$$

Due to the high-dimensional design spaces typical of engineering applications, optimization methods informed by intuition rather than gradients do not scale well. Gradient-based methods promise higher efficiency but require the entire pipeline to be differentiable. Commercial and open-source design-space software like computer aided design (CAD) tools, meshing tools [1], and simulation software [2] are often not natively differentiable. Recovering adjoint sensitivities from solvers is often time-consuming and error-prone; meanwhile, newer projects leveraging automatic differentiation offer flexibility. In many cases, however, gradients can only be obtained for mesh quantities but not for the mesh structure itself. Gradient-based geometry optimizations have been done, but they are one-off solutions that do not generalize to new software components [3], or they apply to a subset of the workflow steps described above [4, 5]. It is imperative to automate the full chain of steps in order to avoid manual set-up in the design space software, which would drastically increase the step time of algorithmic optimization.

By training surrogates on parts of the workflow, we can replace non-differentiable components, with data-driven approximations. The benefits of surrogates are threefold: (i) they are inherently differentiable; (ii) the input geometry representation can be chosen freely; and (iii) the neural networks can provide fast approximation of the results, which are often sufficient to compute gradients for optimization [4].

Contributions. In this work, we demonstrate this approach on a concrete aerodynamic shape optimization problem. While the example is simplified, the methodology is general:

- We show how to construct an end-to-end differentiable optimization pipeline by replacing OpenFOAM [2] with a U-Net surrogate.
- We provide a modular pipeline for shape optimization where each component can be independently substituted depending on the problem of interest.
- We validate the Tesseract framework's [6] ability to compose pipeline components (differentiable and non-differentiable) into end-to-end workflows for data-generation, model training, and optimization.

The Tesseract framework mentioned above provides a modular architecture for composing computational pipelines from independent components. Crucially, Tesseracts expose gradient information at the component level, making it easy to compose them via AD frameworks like JAX or PyTorch. Each of the components listed below is implemented as a Tesseract.

2 Methodology

In this section we give an overview of our end-to-end differentiable pipeline, and explain key design choices we made while building it. The pipeline is split into three parts: data generation, training, and optimization — each consisting of multiple components.

Data generation. The data generation pipeline consists of four components. First, `geometry` generates 3D geometries, both a signed distance field (SDF) on a regular grid and a surface mesh, from design parameters. The geometries are truncated cones with hemispherical caps, arbitrarily rotated in space. Next, `mesh` converts the surface mesh into a volumetric tetrahedral mesh using GMSH [1], with adaptive refinement near the shape's boundary. Then, the `openfoam` component uses OpenFOAM to solve the incompressible Navier-Stokes equations to obtain flow fields around the shape. Finally, `interpolate-to-grid` maps the volumetric mesh data into a regular grid suitable for surrogate training. Full details on geometry representation, meshing, and computational fluid dynamics (CFD) simulation are provided in the appendix.

Surrogate training. To replace the computationally expensive and non-differentiable CFD simulation, we trained a surrogate model to predict flow fields directly from the SDF representation of the geometry. The architecture chosen for the surrogate model is a U-Net[7] leveraging the implementation in the PhysicsNemo library[8] with additional modifications. The network receives the SDF as input, as well as enriched input representations: positional encoding via sine and cosine functions of normalized spatial coordinates, and obstacle mask distinguishing interior from exterior regions (more details on input can be found in Appendix C.1). The `unet-trainer` component trains a U-Net to learn a direct mapping from the enriched input to the flow fields.

Optimization. The optimization workflow chains `geometry`, `interpolate-to-grid`, and `unet-inference` to predict flow fields from design parameters. The loss is computed, and its gradient with respect to the design parameters is then derived through backpropagation.

The modularity of this formulation means that individual components can be swapped or upgraded independently. For example, the geometry representation could be changed from rounded cones to parametric airfoils, or the U-Net could be replaced with a different architecture. This approach enables practitioners to selectively introduce differentiability where it provides the most value, rather than having to build monolithic differentiable solvers (often from scratch).

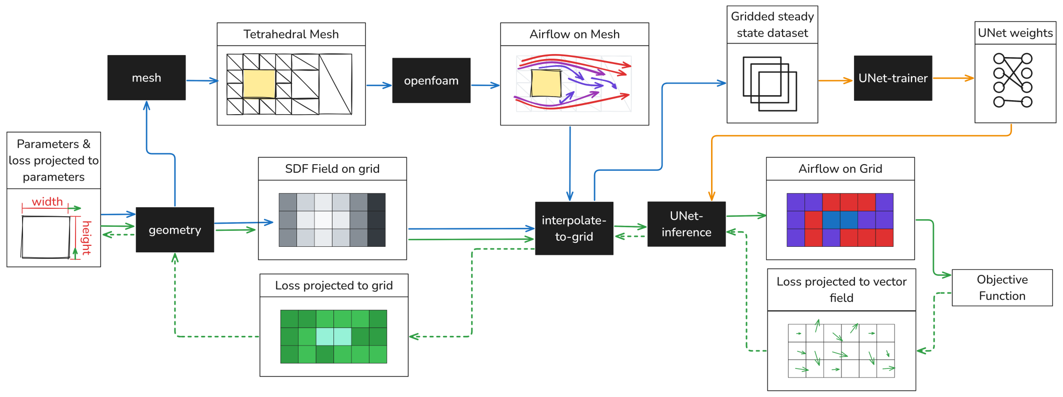

Figure 1: Pipeline architecture for surrogate-based shape optimization. Arrow colors indicate workflow stages: blue for data generation, orange for surrogate training, green for optimization. Solid arrows represent forward primal values, while dashed arrows represent backward gradient flow. All components that are written as Tesseracts are displayed as black boxes.

3 Demonstration

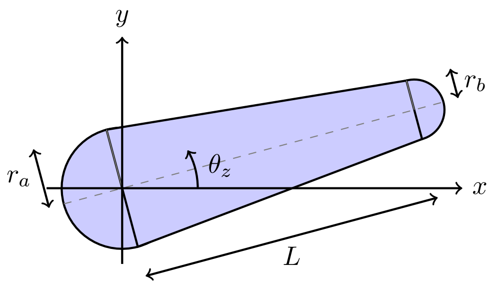

Figure 2: Example of the primitives we use for shape optimization

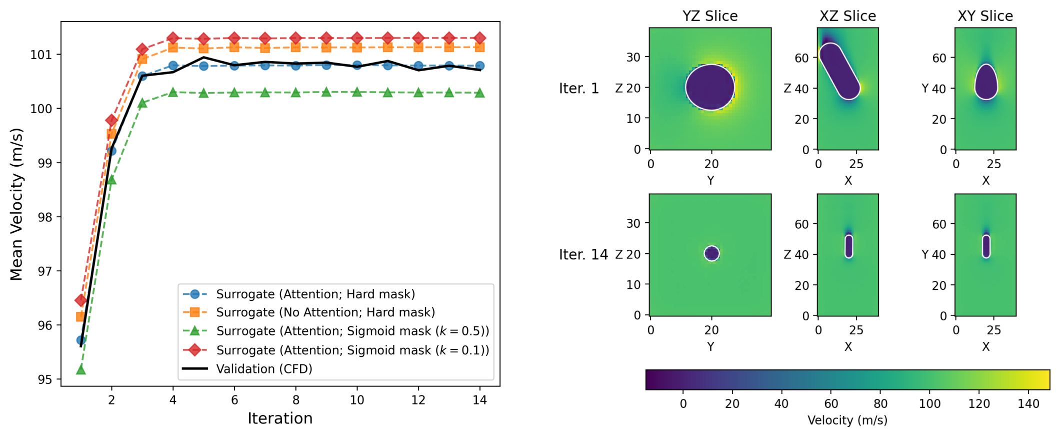

Figure 3: Optimization results: (left) convergence of the objective function over 14 iterations; (right) design evolution showing the shape aligning with flow and reducing volume.

To demonstrate the effectiveness of our proposed optimization pipeline, we consider the design presented in Figure 2, which describes a 3D rounded cone. It is parametrized by two radii r_a, r_b, a length, L and the angle θ_z. The geometry and the parameter ranges used for data generation are described in more detail in A.1. We consider this shape as being immersed in a fluid flow, and the target for optimization is chosen to maximize the mean x-component of the flow velocity. This effectively minimizes the drag created by the shape.

We first generated a dataset using the data generation pipeline (see Figure 1) to create 896 samples that were split randomly into 768 and 128 samples for test and validation. We then trained a U-Net surrogate model to predict fluid flow around the solid object. The model predicts the 3-component velocity vector field with min-max normalization to a magnitude range of [0, 128.6749]. Training was performed on gridded data with spatial resolution of $80 \times 40 \times 40$ (after slicing from full domain), which centers a box of this size on the solid object and neglects the external boundary. The target data is a 3-component velocity field. The model hyperparameters are detailed in Appendix C.2. We performed ablation studies for four distinct configurations to determine the effects of attention and obstacle mask smoothing as detailed in Appendix C.3.

The optimization workflow is made differentiable through a system of Tesseracts. It is summarized as:

$$\begin{aligned}&\texttt{design-params} \leftrightarrow \texttt{geometry} \leftrightarrow \texttt{interpolate-to-grid}\\ &\quad \leftrightarrow \texttt{unet-inference} \leftrightarrow \texttt{QoI},\end{aligned}$$

where `QoI` is defined as the scalar objective function $\Theta = \mathrm{mean}(U_x)$, and double arrows indicate forward propagation of primals and backward propagation of gradients. Gradient-based optimization of Θ is performed using the MMA Optimizer [9]. The design parameters $\{r_a, r_b, L, \theta_z\}$ are initialized as $r_a^0 = r_b^0 = 1.5$, $L^0 = 5$ and $\theta_z^0 = 0.5$ and bounded according to $r_a \in (0.5, 1.5)$, $r_b \in (0.5, 1.5)$, $L \in (2.0, 5.0)$, and $\theta_z \in (-0.5, 0.5)$, then optimization was performed through the iterative maximization of Θ for 20 iterations or until the relative change in design variables is under 1%.

The optimization convergence behavior and the design evolution are illustrated in Figure 3. As expected, the optimization alters the design so that the solid object creates as little drag as possible to maximize the average flow velocity by minimizing the frontal area of the object. The optimized result, therefore, has aligned the inclusion with the flow and decreased its volume to the design limits. Although the design space is rather limited, the gradient recovery was successful, and we observe similar optimization trajectories for all surrogate model variants; convergence was nearly monotonic and reached the desired tolerance in less than 14 iterations. The demonstrated objective has a straightforward solution, and future work will explore more complex geometries, different objectives, and generalization performance outside of the training regime. This study demonstrates that the Tesseract workflow enabled the recovery of gradients from a previously non-differentiable workflow, and those gradients were sufficient to explore the parameterized design space.

4 Conclusions

We demonstrated a surrogate-based approach to gradient-guided shape optimization that does not require a differentiable physics solver by wrapping computational units into Tesseracts. While the specific problem we investigated uses a limited design space, the modularity of this pipeline makes it extensible with relative ease to more interesting problems, especially ones with more complex geometries and loss functions. It is worth noticing, however, that this strategy requires substantial upfront investment in data generation and training, and introduces model risk from approximation errors that necessitate some form of validation of optimized designs with high-fidelity simulations.

References

[1] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11): 1309–1331, 2009. doi: https://doi.org/10.1002/nme.2579. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2579`.

[2] Hrvoje Jasak, Aleksandar Jemcov, and Željko Tuković. OpenFOAM: A C++ library for complex physics simulations. *International workshop on coupled methods in numerical dynamics*, 2007. URL `https://api.semanticscholar.org/CorpusID:35226827`.

[3] Mathias Wintzer and Irian Ordaz. Under-track CFD-based shape optimization for a low-boom demonstrator concept. In *33rd AIAA Applied Aerodynamics Conference*, page 2260, 2015.

[4] Benoit Guillard, Edoardo Remelli, Artem Lukoianov, Pierre Yvernay, Stephan R Richter, Timur Bagautdinov, Pierre Baque, and Pascal Fua. DeepMesh: Differentiable iso-surface extraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(11):7072–7087, 2024.

[5] Tianju Xue, Shuheng Liao, Zhengtao Gan, Chanwook Park, Xiaoyu Xie, Wing Kam Liu, and Jian Cao. JAX-FEM: A differentiable GPU-accelerated 3D finite element solver for automatic inverse design and mechanistic data science. *Computer Physics Communications*, 291:108802, 2023.

[6] Dion Häfner and Alexander Lavin. Tesseract Core: Universal, autodiff-native software components for Simulation Intelligence. *Journal of Open Source Software*, 10(111):8385, 2025. doi: 10.21105/joss.08385. URL `https://doi.org/10.21105/joss.08385`.

[7] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention U-Net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.

[8] PhysicsNeMo Contributors. NVIDIA PhysicsNeMo: An open-source framework for physics-based deep learning in science and engineering, February 2023. URL `https://github.com/NVIDIA/physicsnemo`.

[9] Krister Svanberg. The method of moving asymptotes—a new method for structural optimization. *International journal for numerical methods in engineering*, 24(2):359–373, 1987.

[10] Thomas Lewiner, Hélio Lopes, Antonio Vieira, and Geovan Tavares. Efficient implementation of Marching cubes' cases with topological guarantees. *Journal of Graphics Tools*, 8, 04 2012. doi: 10.1080/10867651.2003.10487582.

[11] Bane Sullivan and Alexander Kaszynski. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, May 2019. doi: 10.21105/joss.01450. URL `https://doi.org/10.21105/joss.01450`.

Appendix A Geometry representation and parametrization

A.1 `geometry` Tesseract

In order to generate a family of candidate shapes, we utilized a differentiable signed distance function (SDF) representation. For the sake of definiteness, we use truncated cones with hemispherical caps as the primitive for both data generation and optimization.

The primitives are parameterized by six design variables, and we generated 1000 geometries with the following ranges:

1. Two radii (r_a, r_b) in radians controlling both the cone's cross-sectional dimensions and the curvature of the hemispherical caps at the base and tip. Uniformly sampled from the range [0.5, 1.5].
2. A scalar length (L) defining the cone's extent. Uniformly sampled from the range [2.0, 5.0].
3. Three Euler angles (θ_x, θ_y, θ_z) in radians specifying the cone's spatial orientation. The cone's axis is initially aligned with the x-axis and then rotated according to the specified Euler angles. Here, only the angle θ_y is sampled from the range [-0,5 0.5], where the x and y components are set to zero.

The geometries are exported as both SDFs on a regular grid and surface meshes; this is because the former are more amenable to automatic differentiation, whereas the latter are needed by the physical solver.

We evaluate the SDFs for each geometry on a regular grid in a vectorized fashion using JAX. The grid spans a bounding box specified by user-defined bounds. Notice that for this part of the computation, automatic differentiation is exposed through the vector-Jacobian product interface of Tesseracts.

Surface meshes are extracted from the SDF field using the marching cubes algorithm with the Lewiner [10] variant, which ensures topological correctness. Finally, the mesh is post-processed using Laplacian smoothing to reduce staircase artifacts from the discrete grid representation while preserving the overall geometry.

A.2 `mesh` Tesseract

The volumetric mesh generation component converts the parameterized surface geometry into a domain suitable for CFD analysis. This process uses GMSH to create a tetrahedral mesh representing the fluid domain around the obstacle.

The computational domain consists of a rectangular box with the optimized shape positioned as an interior void. The box is centered at $(L_x/4, 0, 0)$, where L_x is the projected length of the shape on the x axis.

The domain is constructed using GMSH's built-in geometry kernel. To ensure high-quality volume mesh generation, the input STL surface undergoes optional reparameterization. This process uses GMSH's `classifySurfaces` function with a tolerance angle (20°) to identify distinct geometric features and create smooth surface patches. Curves are split when they deflect beyond a specified angle threshold (180°).

Mesh refinement is controlled through a hierarchy of size parameters:

- Global bounds constrain element sizes throughout the domain
- A coarse target size is applied to far-field regions
- A fine target size is enforced on the obstacle surface

This approach concentrates the mesh's resolution near the shape boundary where more detail is needed, while maintaining efficiency in the far field.

The resulting tetrahedral mesh is exported in GMSH MSH 2.2 format. Physical groups are assigned to all boundary surfaces and the fluid volume to facilitate boundary condition specification in the CFD solver.

Appendix B Computational Fluid Dynamics Simulation

The CFD simulation component uses OpenFOAM to solve the incompressible Navier-Stokes equations. This module accepts the tetrahedral mesh from the meshing stage and returns flow field data for subsequent computation.

The simulation workflow is orchestrated through a bash script that executes the standard OpenFOAM preprocessing, solving, and postprocessing pipeline. The GMSH mesh is first converted to OpenFOAM's native format using the `gmshToFoam` utility, which preserves the physical group tags assigned during mesh generation.

We used the `incompressibleFluid` solver within OpenFOAM's `foamRun` framework to solve the incompressible Navier-Stokes equations. The flow is characterized by a density $\rho = 1\,\mathrm{kg/m^3}$ and kinematic viscosity

$\nu = 10^{-5}\mathrm{m}^2/\mathrm{s}$. In this regime, the Reynold's number remains on the order of 10^7 regardless of the inclusion geometry, so Reynolds-averaged Navier–Stokes (RANS) equations are used to describe the fluid flow.

The computational domain employs the following boundary conditions:

- Freestream velocity conditions set to the uniform flow velocity $(100, 0, 0)$ m/s in both the inlet and the outlet.
- No-slip conditions enforcing zero velocity at the channel walls
- No-slip condition representing the solid boundary at the obstacle surface.

The solver employs an adaptive time-stepping scheme with a maximum Courant number of 5 and an initial time step of $\Delta t = 0.001$ s. The simulations progress from $t = 0$ to $t = 0.15$ s, with the output written at the initial and final times. This transient formulation allows the flow to develop from the uniform initial conditions to a quasi-steady state suitable for force evaluation.

Upon completion, the simulation data is exported to VTK format using OpenFOAM's `foamToVTK` utility. The output includes separate files for each boundary patch (inlet, outlet, walls, obstacle surface) and the internal volume mesh, each containing point coordinates, cell connectivity, and field data (velocity, pressure, turbulence quantities). These VTK files are read using PyVista [11] and converted to a structured format containing both geometric information and field quantities, enabling subsequent interpolation and gradient computation in the optimization pipeline. Since some simulation runs failed or resulted in poor convergence. Therefore, we removed samples with NaN values or with mesh cells that had a velocity vector with a magnitude greater than 160.

Appendix C Surrogate Training

C.1 Input engineering

Rather than using only the SDF, the network receives an enriched input representation:

- SDF: Normalized signed distance values indicating proximity to the obstacle surface.
- Positional encoding: Sine and cosine functions of normalized spatial coordinates (x, y, z), providing explicit positional information to the translation-invariant convolutions.
- Obstacle mask: Indicator distinguishing interior (SDF < 0) from exterior regions - Hard mask: Binary threshold at SDF=0 - Sigmoid mask m with temperature parameter k where smaller k produces sharper transitions at the boundary The sigmoid mask provides a differentiable alternative to hard thresholding, enabling gradients to flow through the masking operation during backpropagation.

$$m(\mathrm{SDF}) = \sigma\left(-\frac{\mathrm{SDF}}{k}\right) = \frac{1}{1 + e^{\mathrm{SDF}/k}} \tag{1}$$

C.2 Hyperparameters

Key hyperparameters for model and optimizer were chosen through trial and error to establish stable baseline training.

- Batch size: 64
- Learning rate: $1.5 * 10^{-4}$
- Optimizer: Adam with default β parameters (β_1=0.9, β_2=0.999)
- Training epochs: 400
- Loss function: Mean squared error (MSE) on predicted velocity fields
- Encoder depth: 3 levels with progressively increasing feature maps [64, 64, 128, 128, 512, 512]
- Attention Gates: Feature maps [512, 128] with 256 intermediate channels
- Convolutional blocks: 2 consecutive blocks per level using 3×3×3 kernels with GELU activation
- Normalization: Layer Normalization
- Pooling: 3D max pooling with 2×2×2 stride between encoder levels

Training was performed on 4 NVIDIA A100 80GB PCIe GPUs with an approximate training time of 6.5 hours per run, and results are illustrated in Figure 4.

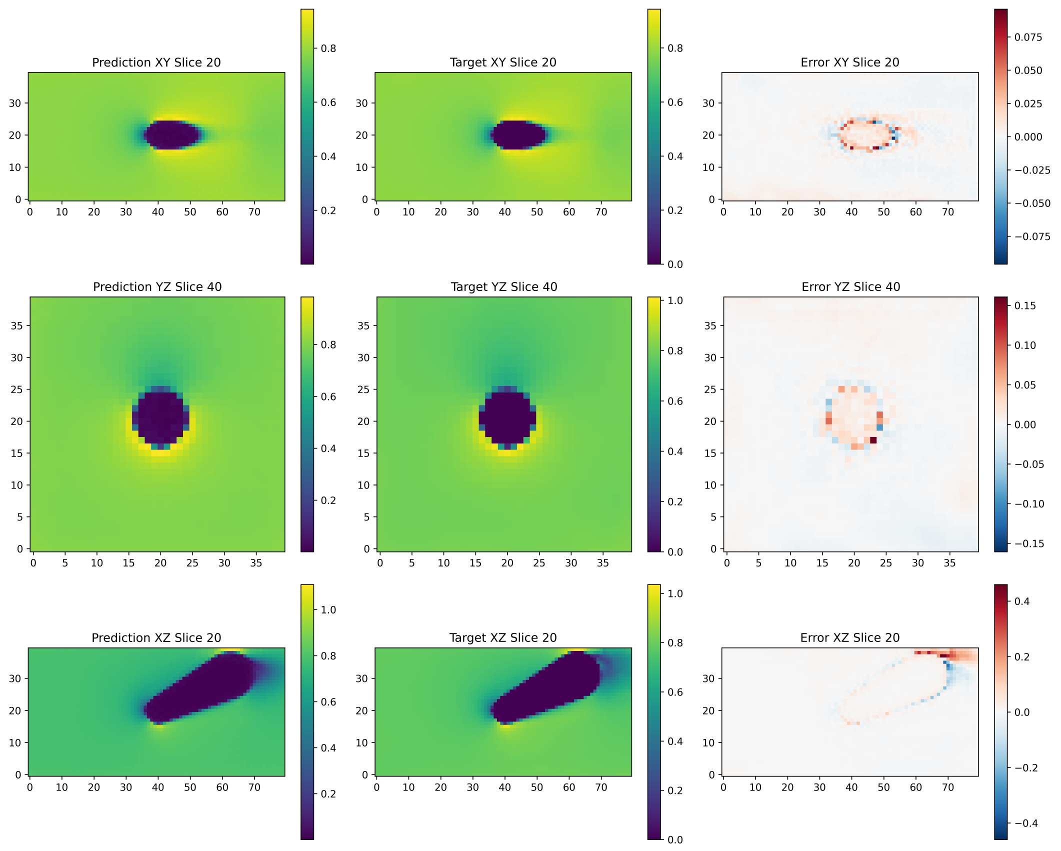

Figure 4: Example normalized target and prediction slices in xy, yz, and xz planes for trained baseline model.

C.3 Ablation Studies

We performed experiments for four distinct configurations to determine the effects of attention and obstacle mask smoothing as detailed in Table 1. In addition to MSE loss (Figure 5), we calculated the correlation between the spatial gradient of the absolute error $\epsilon = |y - f(x)|$ and the predicted value $f(x)$:

$$\text{Corr}(\nabla\epsilon, f(x)) \tag{2}$$

where $Corr$ is the Pearson correlation function, ∇ is the gradient operator (calculated by nearest neighbor finite difference). This metric assesses whether regions of fluctuating error align with regions of high velocity (Figure 6).

These results lead to the following conclusions.

- Attention did not lead to improved model performance in this particular dataset.
- Sigmoid masking resulted in slower model convergence, regardless of k, but did not strongly affect the overall performance of the trained model.
- Stronger fluctuations in model error had a slightly higher likelihood to be found in regions of low velocity than in regions of high velocity.

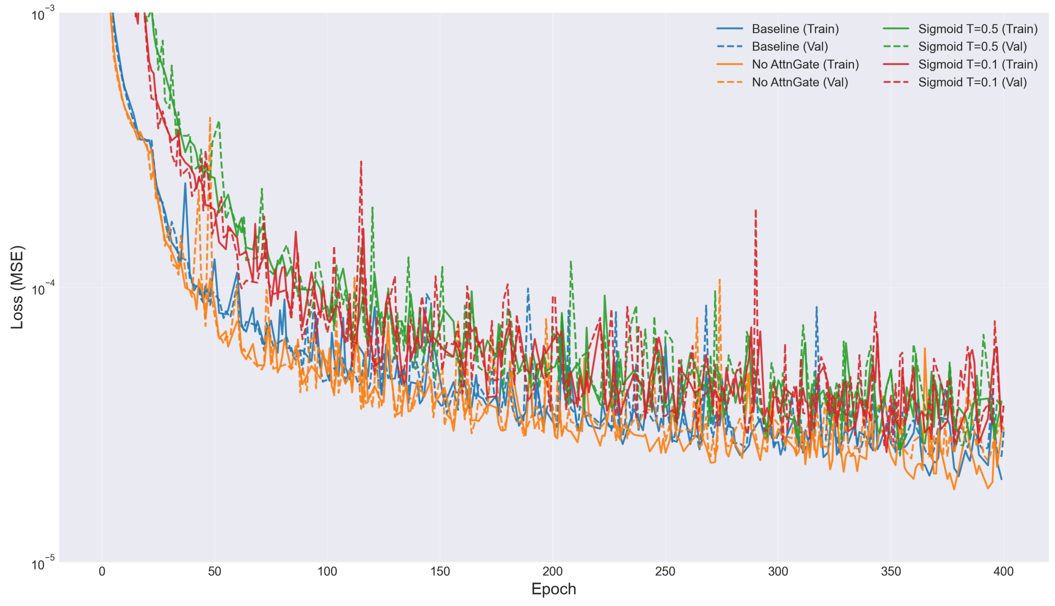

Figure 5: Training and validation loss curves for the ablation study. Solid lines indicate training loss, dashed lines indicate validation loss. All models converge to similar performance, with the baseline configuration achieving the lowest validation loss.

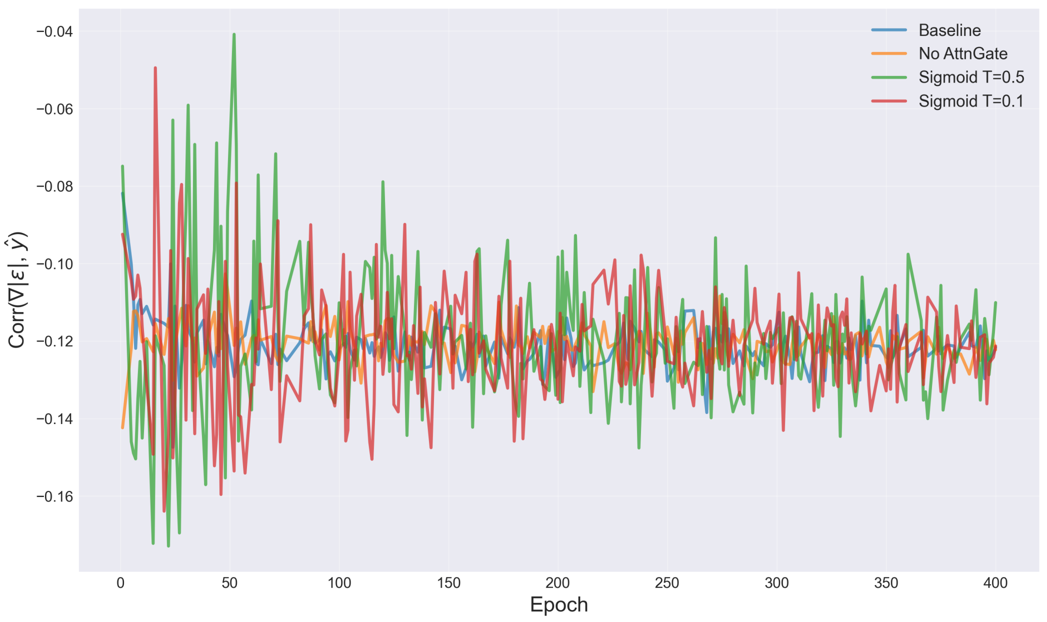

Figure 6: Error gradient correlation metric during training. The metric measures the Pearson correlation between spatial gradients of the absolute error field ($\nabla\epsilon$) and the predicted velocity values ($f(x)$). Negative values indicate that regions with rapidly varying errors tend to have lower predicted velocities, suggesting difficulty in low-velocity or boundary regions.

Table 1: Training Results: Ablation Study Comparing Attention Gates and Masking Strategies

Attn	Mask Type	Temp	Train Loss (Final) [$\times 10^{-5}$]	Val Loss (Final) [$\times 10^{-5}$]	Best Val (Train/Val, Epoch) [$\times 10^{-5}$]	Corr($\nabla\|\epsilon\|$,$\hat{y}$)
Yes	Hard	×	2.00	2.99	2.17/2.42 (368)	-0.1217
No	Hard	×	2.97	3.01	1.85/2.33 (393)	-0.1223
Yes	Sigmoid	0.5	3.84	3.69	2.61/2.44 (354)	-0.1101
Yes	Sigmoid	0.1	2.98	3.77	2.82/2.63 (378)	-0.1213