
NVIDIA SIMNET™: AN AI-ACCELERATED MULTI-PHYSICS SIMULATION FRAMEWORK

PREPRINT- DECEMBER 16, 2020

Oliver Hennigh
ohennigh@nvidia.com

Susheela Narasimhan
susheelan@nvidia.com

Mohammad Amin Nabian
mnabian@nvidia.com

Akshay Subramaniam
asubramaniam@nvidia.com

Kaustubh Tangsali
ktangsali@nvidia.com

Max Rietmann
mrietmann@nvidia.com

Jose del Aguila Ferrandis
jaguila@mit.edu

Wonmin Byeon
wbyeon@nvidia.com

Zhiwei Fang
zhiweif@nvidia.com

Sanjay Choudhry*
schoudhry@nvidia.com

NVIDIA
developer.nvidia.com/simnet

ABSTRACT

We present SimNet, an AI-driven multi-physics simulation framework, to accelerate simulations across a wide range of disciplines in science and engineering. Compared to traditional numerical solvers, SimNet addresses a wide range of use cases - coupled forward simulations without any training data, inverse and data assimilation problems. SimNet offers fast turnaround time by enabling parameterized system representation that solves for multiple configurations simultaneously, as opposed to the traditional solvers that solve for one configuration at a time. SimNet is integrated with parameterized constructive solid geometry as well as STL modules to generate point clouds. Furthermore, it is customizable with APIs that enable user extensions to geometry, physics and network architecture. It has advanced network architectures that are optimized for high-performance GPU computing, and offers scalable performance for multi-GPU and multi-Node implementation with accelerated linear algebra as well as FP32, FP64 and TF32 computations. In this paper we review the neural network solver methodology, the SimNet architecture, and the various features that are needed for effective solution of the PDEs. We present real-world use cases that range from challenging forward multi-physics simulations with turbulence and complex 3D geometries, to industrial design optimization and inverse problems that are not addressed efficiently by the traditional solvers. Extensive comparisons of SimNet results with open source and commercial solvers show good correlation.

1 Introduction

Simulations are pervasive in every domain of science and engineering. However, they become computationally expensive as more geometry details are included and as model size, the complexity of physics or the number of design evaluations increases. Since large simulations can sometimes take hours to days to complete a single run, the speed of iteration in the workflow is constrained by the speed of simulations. Although deep learning [1] offers a path to overcome this constraint, supervised learning techniques are used most often in the form of traditional data driven neural networks (e.g., [2–4]). However, generating data can be an expensive and time consuming process. Furthermore, these models may not obey the governing physics of the problem, involve extrapolation and generalization errors, and provide unreliable results.

*Corresponding author

In comparison with the traditional solvers, neural network solvers [5–7] can not only do parametrized simulations in a single run, but also address problems not solvable using traditional solvers, such as inverse or data assimilation problems and real time simulation. They can also be embedded in the traditional solvers to improve the predictive capability of the solvers. In its most basic form, a neural network solver consists of a standard fully-connected neural network or multi-layer perceptron. The loss functions used to train these networks are augmented by Partial Differential Equations (PDEs) describing a physical process, and require computing the derivatives of the outputs with respect to the inputs. Initial and boundary conditions can be imposed as hard constraints by changing the network architecture or more generally, used in the loss function to fully specify the physical constraints. Training of neural network forward solvers can be supervised based on the governing laws of a physics only, and thus, unlike the data-driven deep learning models, neural network solvers do not require any training data. However, for data assimilation or inverse problems, data constraints are introduced in the loss function. Automatic differentiation is used to compute the derivatives required for the residuals of the PDEs. Gradients of the loss function are then back-propagated through the entire network for training using a stochastic gradient descent optimizer.

Rapid evolution of GPU architecture suited for AI and HPC, as well as introduction of open source frameworks like Tensorflow have motivated researchers to develop novel algorithms for solving PDEs (e.g., [8–20]). Recently, a number of neural network solver libraries are being developed aiming at making these solvers more accessible to the academia and researchers. Among those are Tensorflow-based DeepXDE [21], Keras-based SciANN [22], and Julia-based NeuralPDE.jl [23]. Although the existing research studies and libraries played a crucial role in advancing the neural network solvers, the attempted examples are mostly limited to simple 1D or 2D domains with straightforward governing physics, and the neural network solvers in their current form still struggle to forward solve real-world applications that involve complex 3D geometries and multi-physics systems. In this paper we present SimNet, a new framework for academia as well as industry, that aims to address the current computational challenges with neural network solvers. As an example, SimNet enables design optimization of a FPGA heat sink (see Figure 1) through a single network training without any training data. In contrast, the traditional solvers are not capable of simulating geometries with several design parameters in a single run.

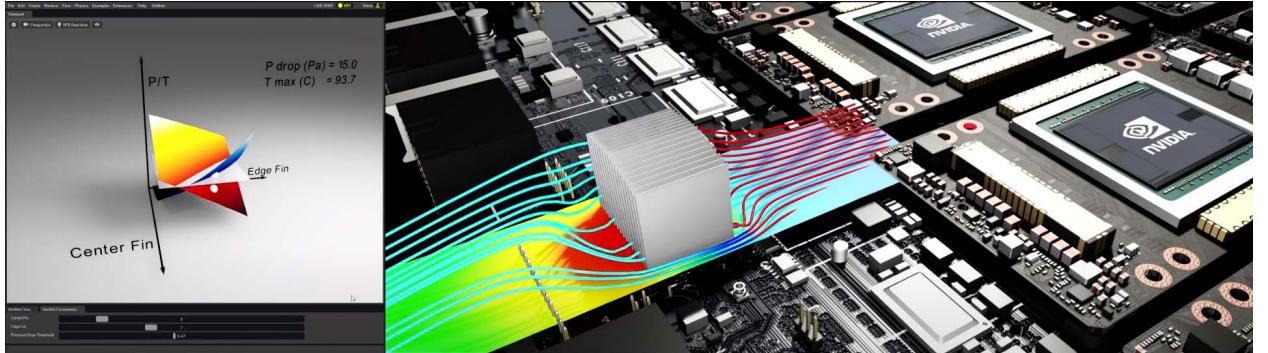


Figure 1: Design optimization of an FPGA heat sink using SimNet. The center and side fin heights are the two design variables, with a linear transition for the height of the intermediate fins.

Our Contributions: Several research studies have recently been published demonstrating solution of PDEs using neural networks. However, our experience has shown that they do not converge well when used as forward solvers for industrial problems due to the gradients, singularities and discontinuities introduced by complex geometries or physics. Our main contributions in this paper are to offer several novel features to address these challenges - Signed Distance Functions (SDFs) for loss weighting, integral continuity planes for flow simulation, advanced neural network architectures, point cloud generation for real world geometries using constructive geometry module as well as STL module and finally parameterization of both geometry and physics. Additionally, for the first time to our knowledge, we solve high Reynolds number flows using zero-equation turbulence model in industrial applications without using any data.

2 AI driven Simulations

Neural network solvers are capable of addressing the various areas in the sciences and engineering as shown in Figure 2. Other areas such as uncertainty quantification and sensitivity analysis could also benefit from the neural network solvers.

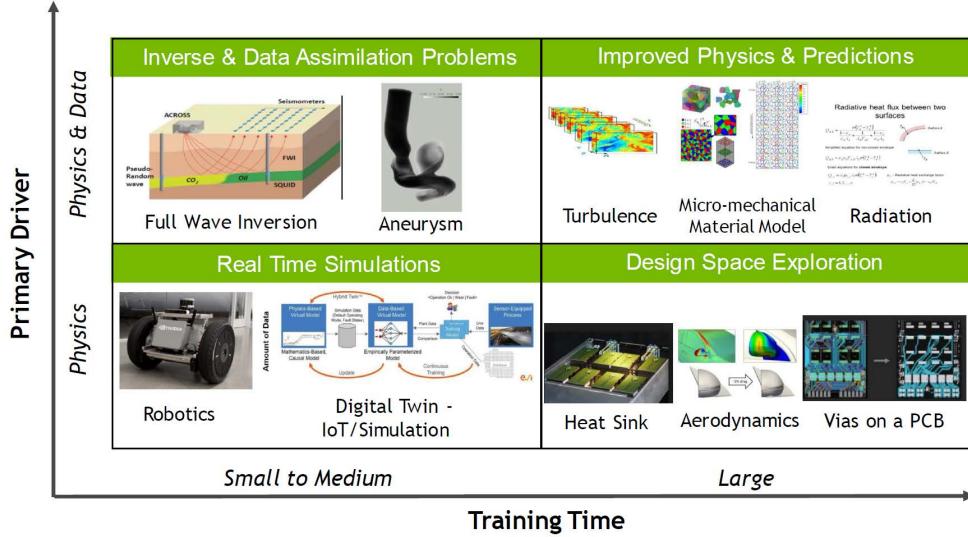


Figure 2: Four major areas in computational science and engineering addressed by SimNet.

Design Space Exploration and Optimization: Accurate physical simulation of products, processes and systems saves time and cost in product development by providing engineers with feedback on their designs without building and testing physical prototypes. In a fast-paced competitive world where manufacturers are constantly trying to reduce the time-to-market and cost while optimizing the product behavior and manufacturing processes, companies are turning increasingly towards simulations. However, due to time and computational resource constraints, only a limited number of simulations can be performed during a design cycle. Neural network solvers provide a fast simulation surrogate to traditional simulations that allows fast design space exploration due to ability to work with several different geometries simultaneously as well as parameterized design for a single geometry.

Improved physics and predictions: Traditional solvers use various models to represent physical phenomenon. These models can be replaced by neural network solvers trained on highly accurate physics results or experiments. For example, viscosity and turbulence models can be learned using the neural networks on high fidelity results (e.g. DNS) and experiments. These learnt models can then be called from a computationally inexpensive framework (e.g. RANS) from within a solver to get a higher fidelity at a lower computational expense. Besides fluid mechanics, examples of other physics areas that can benefit from such modeling are radiation in heat transfer and micromechanics of materials in solids.

Inverse Modeling: Neural network solvers are a natural choice to be used for inverse problems where the underlying physical behavior is discovered by assimilating data from observations. The physics can range from acoustics with applications in oil and gas exploration (e.g., using recorded sound data to find the sub-surface velocity model) to fluid mechanics with applications in medical imaging (e.g. given a 4D CT or MR images of intracranial aneurysm or stenosis in the artery, one can back calculate various physics quantities such as velocities and pressure using a passive scalar). Neural network solvers also present an attractive alternative to traditional simulations where the right boundary conditions are not easily available.

Real-Time Simulations: Several situations require nearly instantaneous feedback where simulations would provide valuable information but are not possible due to the real-time nature of the situation. Examples include robots, digital twins, autonomous driving and real time evaluation of aerodynamics to influence driving decisions on race-tracks. Neural network solvers provide the capability for real-time simulations in these systems. A pre-trained neural network solver can provide real-time feedback in a matter of seconds, and thus, can make a huge impact in the areas that real-time simulations are crucial.

The rest of the paper is structured as follows. In Section 3, we provide a brief overview on the theoretical aspects of the neural network solvers. The architecture of the SimNet is then introduced next in Section 4. The capability of SimNet in solving complex real-world problems involving turbulent multi-physics simulations, complex geometries, design optimization, and inverse simulations is illustrated in Section 5. We discuss performance improvements in Section 6, and conclude with a summary of the main contributions of SimNet.

3 Neural Network Solvers

In this section we provide an introduction to neural network solvers. Briefly, a neural network solver approximates the solution to a given PDE and a set of boundary and initial constraints using a feed-forward fully-connected neural network. The model is trained by constructing a loss function for how well the neural network is satisfying the PDE and constraints. If the network is able to minimize this loss function then it will in effect, solve the given PDE. A schematic of the structure of a neural network solver is shown in Figure 3.

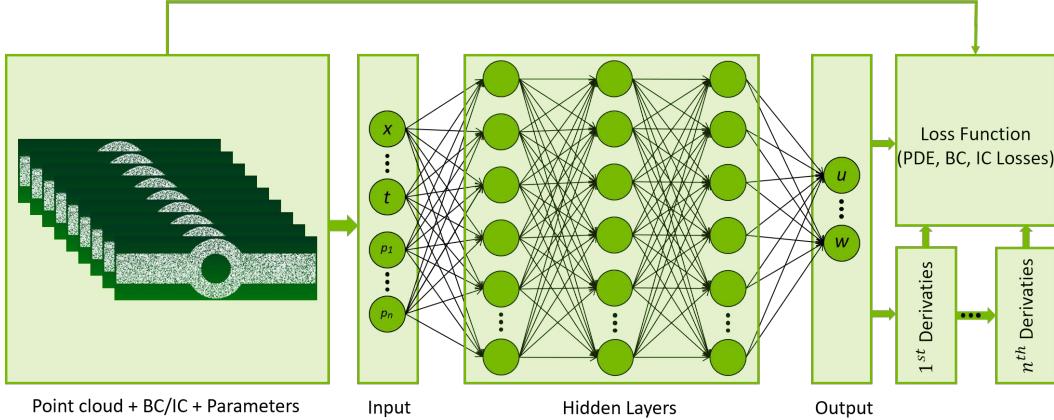


Figure 3: A schematic of the structure of a neural network solver. The inputs to the network are the spatial coordinates of a point cloud, realizations of time (if applicable), and realizations from the parametric space (if applicable). The inputs are mapped to the quantities of interest via a fully-connected network with nonlinear activation functions. To train this network, a loss function is considered which consists of the derivatives of the output w.r.t inputs (computed using automatic differentiation), and initial and boundary condition information.

Let us consider the following general form of a PDE:

$$\begin{aligned} \mathcal{N}_i[u](\mathbf{x}) &= f_i(\mathbf{x}), \quad \forall i \in \{1, \dots, N_{\mathcal{N}}\}, \mathbf{x} \in \mathcal{D}, \\ \mathcal{C}_j[u](\mathbf{x}) &= g_j(\mathbf{x}), \quad \forall j \in \{1, \dots, N_{\mathcal{C}}\}, \mathbf{x} \in \partial\mathcal{D}, \end{aligned} \quad (1)$$

where \mathcal{N}_i 's are general differential operators, \mathbf{x} is the set of independent variables defined over a bounded continuous domain $\mathcal{D} \subseteq \mathbb{R}^D$, $D \in \{1, 2, 3, \dots\}$, and $u(\mathbf{x})$ is the solution to the PDE. \mathcal{C}_j 's denote the constraint operators that may consist of differential, linear, and nonlinear terms and usually cover the boundary and initial conditions. $\partial\mathcal{D}$ also denotes a subset of the domain boundary that is required for defining the constraints. We seek to approximate the solution $u(\mathbf{x})$ by a neural network $u_{\text{net}}(\mathbf{x})$ that, in its most simple form, takes the following form:

$$u_{\text{net}}(\mathbf{x}; \theta) = \mathbf{W}_n \{ \phi_{n-1} \circ \phi_{n-2} \circ \dots \circ \phi_1 \circ \phi_E \}(\mathbf{x}) + \mathbf{b}_n, \quad \phi_i(\mathbf{x}_i) = \sigma(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i), \quad (2)$$

where $\mathbf{x} \in \mathbb{R}^{d_0}$ is the input to network, $\phi_i \in \mathbb{R}^{d_i}$ is the i^{th} layer of the network, $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $\mathbf{b}_i \in \mathbb{R}^{d_i}$ are the weight and bias of the i^{th} layer, θ denotes the set of network's trainable parameters, i.e., $\theta = \{\mathbf{W}_1, \mathbf{b}_1, \dots, \mathbf{b}_n, \mathbf{W}_n\}$, n is the number of layers, and σ is the activation function. We suppose that this neural network is infinitely differentiable, i.e. $u_{\text{net}} \in C^\infty$. ϕ_E is an input encoding layer, and by setting that to identity function, we arrive at the standard feed-forward fully-connected architecture, which is the most widely used architecture in neural network solvers. More advanced architectures will be introduced in Section 4.3.

In order to train this neural network, we construct a loss function that penalizes over the divergence of the approximate solution $u_{\text{net}}(\theta)$ from the PDE in equation 1, and such that the constraints are encoded as penalty terms. To this end, we define the following residuals by using u_{net} as the approximate solution to the PDE:

$$\begin{aligned} r_{\mathcal{N}}^{(i)}(\mathbf{x}; u_{\text{net}}(\theta)) &= \mathcal{N}_i[u_{\text{net}}(\theta)](\mathbf{x}) - f_i(\mathbf{x}), \\ r_{\mathcal{C}}^{(j)}(\mathbf{x}; u_{\text{net}}(\theta)) &= \mathcal{C}_j[u_{\text{net}}(\theta)](\mathbf{x}) - g_j(\mathbf{x}), \end{aligned} \quad (3)$$

where $r_{\mathcal{N}}^{(i)}$ and $r_{\mathcal{C}}^{(j)}$ are the PDE and constraint residuals, respectively. The loss function then takes the following form:

$$\mathcal{L}_{res}(\theta) = \sum_{i=1}^{N_{\mathcal{N}}} \int_{\mathcal{D}} \lambda_{\mathcal{N}}^{(i)}(\mathbf{x}) \left\| r_{\mathcal{N}}^{(i)}(\mathbf{x}; u_{net}(\theta)) \right\|_p d\mathbf{x} + \sum_{j=1}^{N_{\mathcal{C}}} \int_{\partial\mathcal{D}} \lambda_{\mathcal{C}}^{(j)}(\mathbf{x}) \left\| r_{\mathcal{C}}^{(j)}(\mathbf{x}; u_{net}(\theta)) \right\|_p d\mathbf{x}, \quad (4)$$

where $\|\cdot\|_p$ denotes the p-norm, and $\lambda_{\mathcal{N}}^{(i)}, \lambda_{\mathcal{C}}^{(j)}$ are weight functions that control the loss interplay between within and across different terms. The majority of the research studies focus on finding dynamic weight to scale the different loss terms (e.g., [8, 9, 19, 24]). Currently, SimNet supports three loss weighting algorithms, that are global learning rate annealing as proposed in [9], a new local variant of this algorithm, and also signed distance weighting that will be explained in Section 4.1.

Finally, to train the approximate solution $u_{net}(\theta)$, the network parameters θ are optimized iteratively using variants of the stochastic gradient descent method, such as the Adam optimizer [25]. At each iteration, the integral terms in the loss function are approximated using a regular or Quasi-Monte Carlo method, and using a batch of samples from the independent variables \mathbf{x} . Automatic differentiation is commonly used to compute the required gradients in $\nabla \mathcal{L}_{res}(\theta)$.

4 SimNet Overview

SimNet is a Tensorflow based neural network solver [26] with source code available at: developer.nvidia.com/simnet. It offers various APIs that enable the user to leverage the existing functionality to build their own applications on the existing modules. An overview of SimNet architecture is presented in Figure 4. The geometry modules, PDE module, and data are used to fully specify the physical system. The user also specifies the network architecture, optimizer and learning rate schedule. SimNet then constructs the neural network solver, forms the loss function, and unrolls the graph efficiently to compute the gradients. The SimNet solver then starts the training or inference procedure using TensorFlow’s built-in functions on a single or cluster of GPUs. The outputs are saved in form of CSV or VTK files and can be visualized using TensorBoard and ParaView. In the remainder of this section, we present more detail on the SimNet’s geometry and PDE modules as well as the available network architectures.

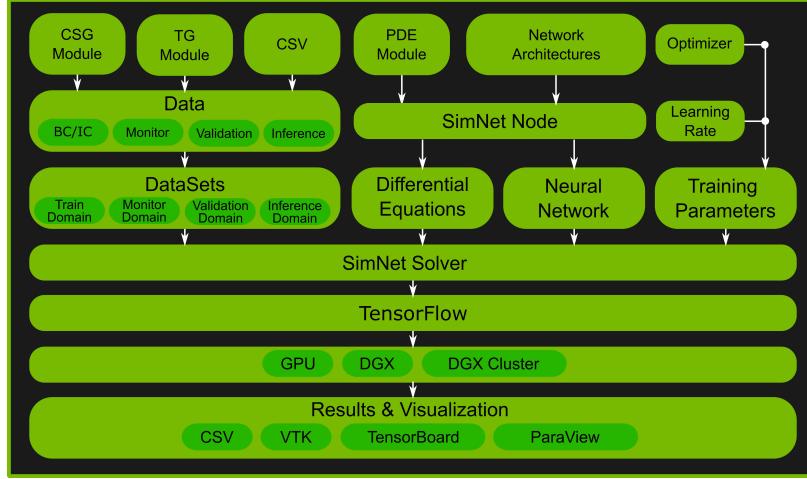


Figure 4: SimNet structure.

4.1 Geometry modules

CSG and TG modules: SimNet currently carries two geometry modules, namely the Constructive Solid Geometry (CSG) and Tessellated Geometry (TG) modules. With SimNet’s CSG module, constructive geometry primitives can be easily defined and Boolean operations can be performed. This will allow creation and parameterization of a wide range of geometries and enables definition of various shapes that can be parameterized and used for design optimization. The TG module in SimNet imports STL, OBJ, and other tessellated geometries to work with complex geometries.

Signed Distance Functions (SDFs): One area of considerable interest is how to weight the loss terms in the overall loss function, as defined in equation 4. SimNet offers spatial loss weighting, where each weight parameter can be a

function of the spatial inputs. In many cases we use the Signed Distance Function (SDF) for this weighting. Assuming \mathcal{D}_x represents the spatial subset of the input domain \mathcal{D} with boundaries $\partial\mathcal{D}_x \subseteq \partial\mathcal{D}$, the SDF-based weight function is defined as

$$\lambda(\mathbf{x}_s) = \begin{cases} d(\mathbf{x}_s, \partial\mathcal{D}_x) & \mathbf{x}_s \in \mathcal{D}_x, \\ -d(\mathbf{x}_s, \partial\mathcal{D}_x) & \mathbf{x}_s \in \mathcal{D}_x^c. \end{cases} \quad (5)$$

Here, \mathbf{x}_s is the spatial inputs, and $d(\mathbf{x}_s, \partial\mathcal{D}_x)$ represents the Euclidean distance between \mathbf{x}_s and its nearest neighbor on \mathcal{D}_x . In general, we have found it beneficial to weight losses lower on sharp gradients or discontinuous areas of the domain. For example, if there are discontinuities in the boundary conditions we may have the loss decay set to zero on these discontinuities. Equation residuals can also be weighted by the SDF of the geometries. If the geometry has sharp corners this often results in sharp gradients in the solution of the PDE. Weighting by the SDF tends to mitigate the deleterious effects of sharp local gradients, and often results in a convergence speed increase as well as improved accuracy in some cases. To accelerate the computation of the SDF on tessellated meshes of complex geometries, we developed a custom library that leverages NVIDIA's OptiX for both inside/outside (sign) testing and distance computation. The sign test uses ray intersection and triangle normal alignment (via dot product). The distance testing is done by using the bounded volume hierarchy (BVH) interface provided by OptiX, which yields excellent performance and accuracy for distance computations.

Point cloud generation: Training points in SimNet are generated according to a uniform distribution by default, which then enable regular Monte Carlo approximation of the loss function in equation 4. Alternatively, SimNet's CSG module also enables quasi-random point cloud generation using the generalized Halton sequences, which provides the means to generate training points with a low level of discrepancy across the domain to perform quasi-Monte Carlo approximation of the loss function. SimNet also has a continuous point sampling algorithm which achieves similar accuracy as the fixed point cloud without the cost of generation.

4.2 PDE module

The PDE module in SimNet contains of a variety of common differential equations including the Navier-Stokes, diffusion, advection-diffusion, wave equations, and linear elasticity equations. In order to make the PDE module extensible for the user to easily define their own differential equations, SimNet uses symbolic mathematics enabled by SymPy [27]. The PDE module in SimNet also provides implementations of turbulence and exact continuity in the Navier-Stokes equations:

Zero-equation turbulence model: Currently, SimNet adopts the zero equation turbulence model for turbulent physics simulation. The zero-equation turbulence model is defined as:

$$\mu_t(\mathbf{x}) = \rho l_m^2 (G(\mathbf{x}))^{\frac{1}{2}}, \quad (6)$$

$$l_m(\mathbf{x}) = \min(0.419d(\mathbf{x}, \partial\mathcal{D}_x), 0.09d_{max}), \quad (7)$$

$$G(\mathbf{x}) = 2 \left[\left(\frac{\partial u}{\partial x}(\mathbf{x}) \right)^2 + \left(\frac{\partial v}{\partial y}(\mathbf{x}) \right)^2 + \left(\frac{\partial w}{\partial z}(\mathbf{x}) \right)^2 \right] + \left(\frac{\partial u}{\partial y}(\mathbf{x}) + \frac{\partial v}{\partial x}(\mathbf{x}) \right)^2 + \left(\frac{\partial u}{\partial z}(\mathbf{x}) + \frac{\partial w}{\partial x}(\mathbf{x}) \right)^2 + \left(\frac{\partial v}{\partial z}(\mathbf{x}) + \frac{\partial w}{\partial y}(\mathbf{x}) \right)^2. \quad (8)$$

Here, $\mu_t(\mathbf{x}) = \rho\nu_t(\mathbf{x})$, l_m is the mixing length, G is the modulus of mean squared rate of strain tensor, d represents the normal distance from wall, and d_{max} is maximum normal distance. The zero-equation turbulence model requires normal distance and its spatial derivatives from no slip walls to compute the turbulent viscosity, and is computed using the SDF from the CSG or TG modules. More advanced turbulence models can be implemented in the current framework.

Exact continuity and integral continuity for incompressible Navier-Stokes equations: Velocity-pressure formulations are the most widely used formulations of the Navier-Stokes equations. Alternatively, one can ensure exact mass conservation using the velocity field from a vector potential [28], defined as:

$$\vec{V} = \nabla \times \vec{\psi} = \left(\frac{\partial \psi_z}{\partial y} - \frac{\partial \psi_y}{\partial z}, \frac{\partial \psi_x}{\partial z} - \frac{\partial \psi_z}{\partial x}, \frac{\partial \psi_y}{\partial x} - \frac{\partial \psi_x}{\partial y} \right)^T, \quad (9)$$

where $\vec{\psi} = (\psi_x, \psi_y, \psi_z)$. This definition of the velocity field ensures that it is divergence-free and that it satisfies continuity, that is:

$$\nabla \cdot \vec{V} = \nabla \cdot (\nabla \times \vec{\psi}) = 0. \quad (10)$$

Although exact continuity is effective in improving the convergence and accuracy, it increases the model training time as it will introduce third-order differential terms. Alternatively, in addition to solving the Navier-Stokes equations in differential form, specifying the volumetric flow rate through some integral continuity planes that are located on the outlet and across the channel significantly speeds up the convergence rate and improves accuracy.

4.3 Network architectures

In addition to the feed-forward, fully connected networks, SimNet offers a number of more advanced architectures, out of which three of the most effective ones are introduced here. Moreover, SimNet offers a large array of activation functions, including the adaptive activation functions proposed in [11].

Fourier feature networks: Neural networks are generally biased toward low-frequency solutions, a phenomenon that is known as "spectral bias" [29]. This can adversely affect the training convergence and accuracy of the model. One approach to alleviate this issue is to perform input encoding, that is, to transform the inputs to a higher-dimensional feature space via high-frequency functions [29–31]. The Fourier feature network in SimNet is a variation of the one proposed in [31] with trainable encoding, and takes the form in equation 12 with the following encoding

$$\phi_E = [\sin(2\pi\mathbf{f} \times \mathbf{x}); \cos(2\pi\mathbf{f} \times \mathbf{x})]^T, \quad (11)$$

where $\mathbf{f} \in \mathbb{R}^{n_f \times d_0}$ is the trainable frequency matrix and n_f is the number of frequency sets.

Modified Fourier feature networks: The modified Fourier feature network is SimNet's novel architecture, where two transformation layers are introduced to project the Fourier features to another learned feature space, and are then used to update the hidden layers through element-wise multiplications, similar to its standard fully connected counterpart in [9]. It is shown in the next section that this multiplicative interaction can improve the training convergence and accuracy. The hidden layers in this architecture take the following form

$$\phi_i(\mathbf{x}_i) = (1 - \sigma(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i)) \odot \sigma(\mathbf{W}_{T_1} \phi_E + \mathbf{b}_{T_1}) + \sigma(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i) \odot \sigma(\mathbf{W}_{T_2} \phi_E + \mathbf{b}_{T_2}), \quad (12)$$

where $i > 1$ and $\{\mathbf{W}_{T_1}, \mathbf{b}_{T_1}\}, \{\mathbf{W}_{T_2}, \mathbf{b}_{T_2}\}$ are the parameters for the two transformation layers. The multiplicative interactions between the Fourier features and hidden layers can potentially improve the model's convergence and accuracy, although at the cost of slightly increasing the training time per iteration.

SiReNs: The authors in [32] proposed a neural network using Sin activation functions dubbed Sinusoidal Representation Networks or SiReNs. This network has similarities to the Fourier feature networks above because using a Sin activation function has the same effect as the input encoding for the first layer of the network. A key component of this network architecture is the initialization scheme. The weight matrices of the network are drawn from a uniform distribution $\mathbf{W}_i \sim U(-\sqrt{6/d_{i-1}}, \sqrt{6/d_{i-1}})$. The input of each Sin activation has a Gauss-Normal distribution and the output of each Sin activation, an arcSin distribution. This preserves the distribution of activations allowing deep architectures to be constructed and trained effectively [32]. The first layer of the network is scaled by a factor ω (with a default value of 30) to span multiple periods of the Sin function and empirically shown to give good performance.

5 Use Cases

In this section, we present four use cases for SimNet to illustrate its capabilities. These use cases are turbulent and multi-physics simulation, simulation with complex geometries, design optimization for a multi-physics system, and an inverse problem. Although, SimNet is capable of simulating transient flows using the continuous-time sampling approach [5], the use cases presented here are time-independent. A more efficient and accurate approach based on the convolutional LSTMs for transient simulations is under development.

For the entire neural network solvers in this section, the architectures consist of 6 layers, each with 512 units, and Swish [33] nonlinearities. For the simulations presented in Sections 5.2 to 5.4, the standard fully connected architecture is used. We use the Adam optimizer with an initial learning rate of 10^{-4} , an exponential decay, and the TensorFlow's

default values for the other optimizer parameters. In training our neural network solvers, we use the regular Monte Carlo integration scheme for the loss function in equation 4. Moreover, for the 3D channel flow problems, we use integral continuity planes. For the simulations in use cases 5.1 to 5.3, we use the SDF for weighting the PDE residuals. It must be noted that use cases 5.1 to 5.3, are solved in the forward manner without using any training data.

5.1 Turbulent and multi-physics simulations

In this part, using an FPGA heat sink example, we demonstrate the SimNet’s capability in accurately solving multi-physics problems involving high Reynolds number flows. The geometry of the FPGA heat sink placed inside a channel is depicted in Figure 5, and the details of the problem setup are reported in Appendix A. This particular geometry is challenging to simulate due to thin fin spacing that causes sharp gradients which are particularly difficult to learn for a neural network solver. Using the zero-equation turbulence model, we will solve a conjugate heat transfer problem to simulate flow over the heat sink placed inside a channel at $Re = 13,239.6$. Since we are dealing with an incompressible flow, there is a one-way coupling between the heat and flow equations, and it is possible to train the temperature field after the flow field is trained. To this end, we train two separate neural network solvers, one for the flow field and the other one for the temperature field. This approach is useful for one-way coupled multi-physics problems as it is possible to achieve significant speed-up, and also simulate cases with same flow boundary conditions but different thermal boundary conditions. For instance, one can easily use the same flow field as in input to train for different thermal boundary conditions.

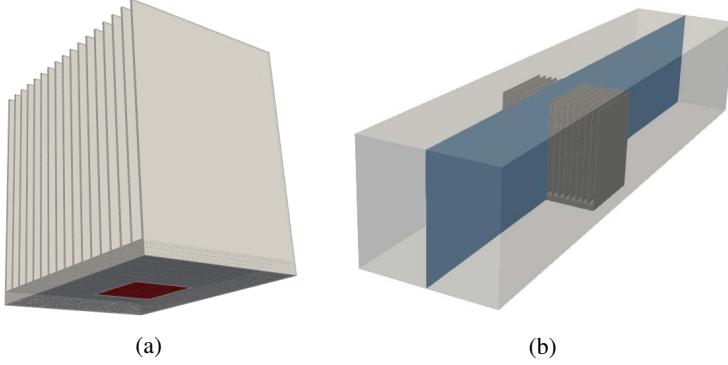


Figure 5: FPGA heat sink example. (a) The FPGA heat sink geometry; (b) the simulation domain. The blue plane represents the plane of symmetry.

We leverage symmetry of the problem to reduce the computational domain, accelerate the training, reduce the memory usage, and potentially, improve the accuracy. To this end, we utilize the following symmetry boundary conditions at the plane of symmetry, as shown in Figure 5b: (1) zero value for the physical variables that are odd functions w.r.t. the plane of symmetry, and (2) zero normal gradient for physical variables that are even functions w.r.t. the plane of symmetry.

Using SimNet, we simulate this conjugate heat transfer problem with Fourier feature network, modified Fourier feature network, and SiReN on the full geometry, and also with Fourier feature network on the half geometry using symmetry boundary conditions. Results for the loss curves are reported in Figure 6, and the pressure drop and peak temperature obtained from various runs are reported in Table 1. Pressure drop and peak temperature are the two quantities that are critical for the heat sink design optimization. The Fourier and modified Fourier feature networks show better convergence behavior compared to the SiReNs as shown in figure 6. This figure also includes the flow convergence results for a Fourier feature model without SDF loss weighting and a standard fully connected model, showing that they fail to provide a reasonable convergence and highlighting the importance of SDF loss weighting and more advanced architectures. The streamlines and temperature profile obtained from the SimNet model with modified Fourier feature network are also shown in Figure 7. A comparison between the SimNet and OpenFoam results for flow and temperature fields is also presented in Figure 8.

5.2 Blood flow in an Intracranial Aneurysm

We demonstrate the ability of SimNet to work with STL geometries from a CAD system. Using the SimNet’s TG module, we simulate the flow inside a patient specific geometry of an aneurysm depicted in Figure 9a. Problem setup details are provided in Appendix B. The SimNet results for the distribution of velocity magnitude and pressure

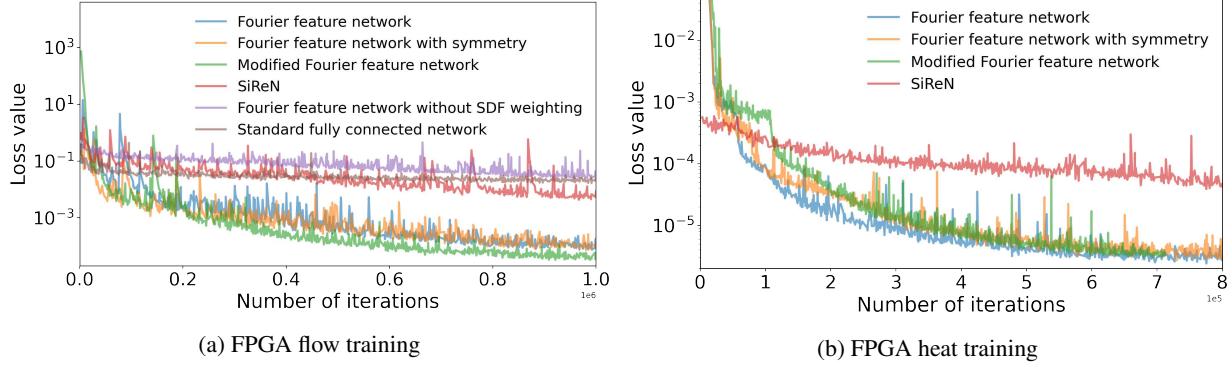


Figure 6: Loss curves for the training of the FPGA conjugate heat transfer problem using different network architectures.

Table 1: A comparison for the pressure drop and peak temperature obtained from various models.

Case Description	P_{drop} (Pa)	T_{peak} (°C)
SimNet: Fourier network (axis spectrum)	25.47	73.01
SimNet: Fourier network (partial spectrum) with symmetry	29.03	72.36
SimNet: Modified Fourier network	29.17	72.52
SimNet: SiReN	29.70	72.00
OpenFOAM Solver	27.82	56.54
Commercial Solver	24.04	72.44

developed inside the aneurysm are shown in Figures 9c and 9d, respectively. Using the same geometry, the authors in [15] solve this as an inverse problem using concentration data from the spectral/hp-element solver Nektar. We solve this problem as a forward problem without any data. When solving the forward CFD problem with non-trivial geometries, one of the key challenges is getting the flow to develop correctly, especially inside the aneurysm sac. The streamline plot in Figure 9b shows that SimNet successfully captures the flow field very accurately.

5.3 Design optimization for multi-physics industrial systems

In this example, we show that SimNet can solve several, simultaneous design configurations in a multi-physics, design space exploration problem much more efficiently than traditional solvers. This is possible because unlike a traditional solver, a neural network trains with multiple design parameters in a single training run. Forward solution of parameterized, complex geometry with turbulent fluid flow between thinly spaced fins and no training data makes this problem extremely challenging for the neural networks. Once the training is complete, several geometry or physical parameter combinations can be evaluated using inference as a post-processing step, without solving the forward problem again. Such throughput enables more efficient design optimization and design space exploration tasks for complex systems in science and engineering.

Here, we train a conjugate heat transfer problem over the Nvidia’s NVSwitch heat sink whose fin geometry are variable, as shown in Figure 10 (nine geometry variables in total). Following the training, we perform a design optimization to find out the most optimal fin configuration. In heat sink design, usually the objective is to minimize the peak temperature that can be reached at the source chip while satisfying a maximum pressure drop constraint. This is necessary to meet the operating temperature requirements of the chip on which the heat sink is mounted for cooling purposes. For details on the problem setup, refer to Appendix C.

The fluid and heat neural networks in this example consist of 12 variables, i.e. three spatial variables and nine geometry parameter variables. Using SimNet, we train these two parameterized neural networks, and then use the trained models to compute the pressure drops and peak temperatures corresponding various combinations using the minimum, maximum and median of these 12 parameters, resulting in $3^{12} = 531,441$ different heat sink designs. The optimized design is the one that minimizes the peak temperature while satisfying a pressure drop constraint as the optimal design. Figure 11 shows the streamlines and temperature profile for the optimal NVSwitch geometry. Details of this geometry are reported in Appendix C.

To confirm the accuracy of the parameterized models, we take the NVSwitch base geometry and compare the SimNet results (obtained from the parameterized model) for pressure drop and peak temperature with the OpenFOAM and

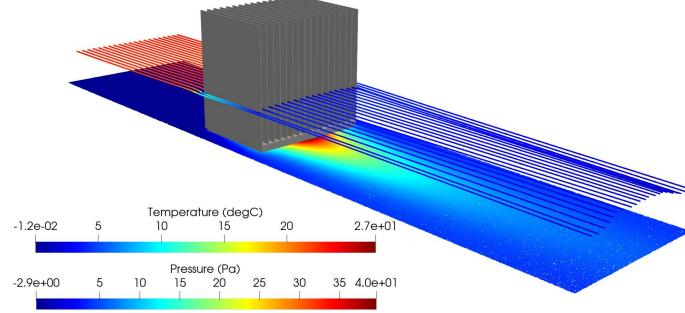


Figure 7: Streamlines and temperature profile obtained from the SimNet model for FPGA with modified Fourier feature network.

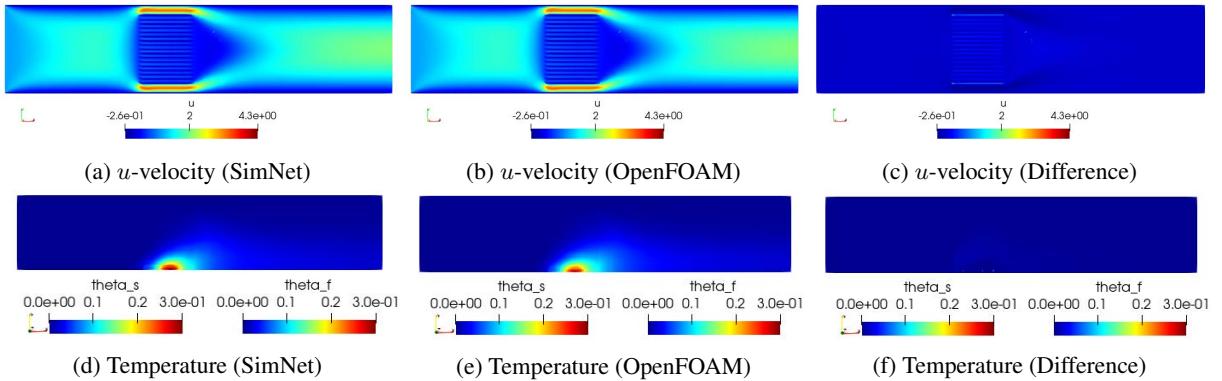


Figure 8: A comparison between the SimNet (with modified Fourier feature network) and OpenFOAM results for FPGA flow and temperature fields. Results are shown on a 2D slice of the domain.

commercial solver results, reported in Table 2.. OpenFOAM over predicts the commercial solver results by 4.5% while the SimNet results under predict the commercial solver result by about 15%.

Table 2: A comparison for the solver and SimNet results for NVSwitch pressure drop and peak temperature.

Property	OpenFOAM Single Run	Commercial Solver Single Run	SimNet Parameterized Run
Pressure Drop (Pa)	133.96	128.30	109.53
Peak Temperature ($^{\circ}C$)	41.55	43.57	39.33

By parameterizing the geometry, SimNet significantly accelerates design optimization when compared to traditional solvers, which are limited to single geometry simulations. The total compute time required by OpenFOAM, a commercial solver, and SimNet for this design optimization task is reported in Table 3. The OpenFOAM and commercial solver runs are run on 22 CPU processors, and the SimNet runs are on 8 V100 GPUs. It can be seen that SimNet can solve this design optimization problem significantly faster than traditional solvers. Increasing the number of design variables or the number of designs to be evaluated magnifies the difference in the time taken for the two approaches of neural network and traditional solvers since for the neural network, once the model is already trained, each of these computations during inference phase take only a fraction of a second. SimNet accelerates the simulation by 45,000x compared to the commercial solver and by 135,000x compared to OpenFOAM.

5.4 Inverse problems

Many applications in science and engineering involve inferring unknown system characteristics given measured data from sensors or imaging for certain dependent variables describing the behavior of the system. Such problems usually involve solving for the latent physics using the PDEs as well as the data. This is done in SimNet by combining the data with PDEs to decipher the underlying physics.

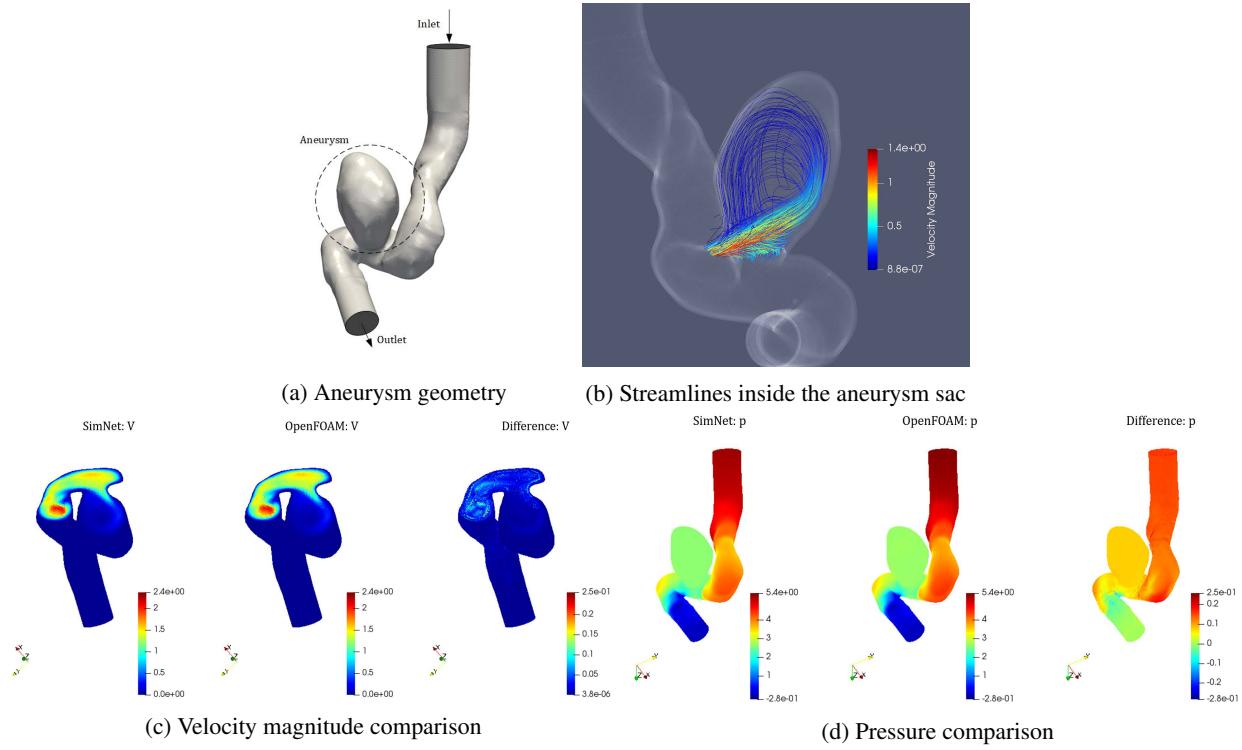


Figure 9: SimNet results for the aneurysm problem, and a comparison between the SimNet and OpenFOAM results for the velocity magnitude and pressure.

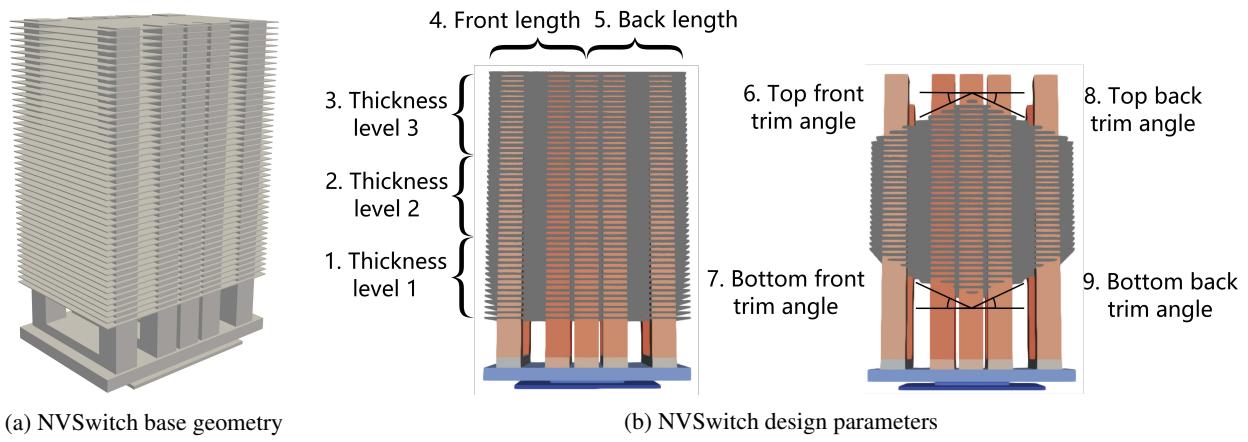


Figure 10: NVSwitch base geometry and design parameters.

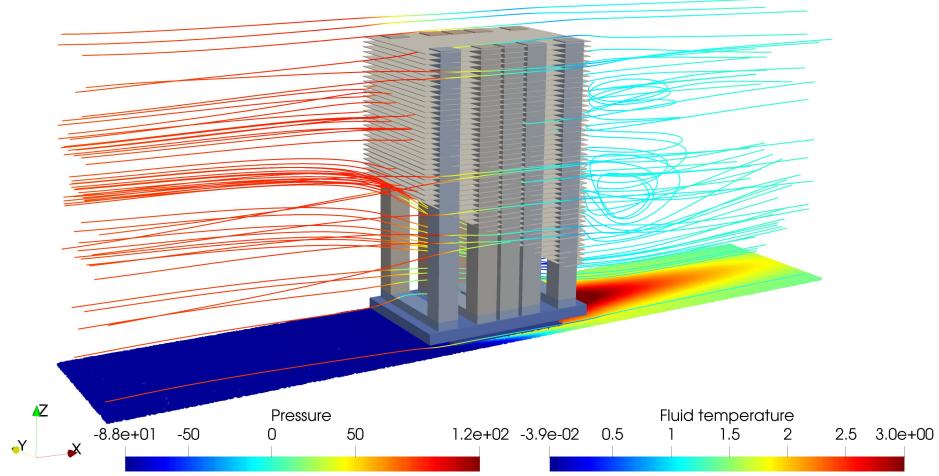


Figure 11: Streamlines colored with pressure and temperature profile in the fluid for optimal NVSwitch geometry.

Table 3: Total compute time needed for different solvers for the NVSwitch heat sink design optimization

Solver	OpenFOAM	Commercial Solver	SimNet
Compute Time (x 1000 hrs.)	405935	137494	3

Here, we demonstrate the ability of SimNet to solve data assimilation and inverse problems on a 2D cross-section of the 3-fin heat sink example. Given the data consisting of flow velocities, pressure and temperature, all of which can be measured, the task is to infer the flow characteristics such as flow viscosity and thermal diffusivity. In reality, the data is collected using measurements but for the purpose of this example, synthetic data generated by OpenFOAM is used. As the majority of diffusion of temperature occurs in the wake of the heat sink (Figure 12), we sample the training points only from this wake zone.

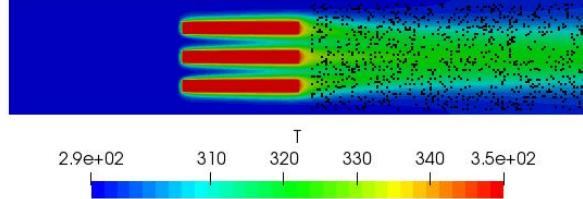


Figure 12: A batch of training points sampled from the OpenFOAM data.

Next, we construct a neural network model with a hybrid data and physics-driven loss function. Specifically, we require the neural network outputs (i.e. u , v , p , and T) to fit to the measurements, and also satisfy the governing laws of the system that includes the Navier-Stokes and advection-diffusion equations. Here, the quantities of interest (i.e., flow viscosity and thermal diffusivity) are also modeled as trainable variables, and are inferred by minimizing the hybrid loss function as shown in Figure 13. A comparison between the predicted SimNet values and the ground truth for flow viscosity and thermal diffusivity is reported in Table 4.

6 Performance Upgrades and Multi-GPU Training

6.1 XLA compiler

XLA (Accelerated Linear Algebra) is a domain specific compiler that allows for just-in-time compilation of TensorFlow graphs. Neural network solvers implemented in SimNet may have many peculiarities including the presence of many point-wise operations. Such operations, while being computationally inexpensive, heavily use the memory subsystem of a GPU. XLA allows for kernel fusion, so that many of these operations can be computed simultaneously in a single kernel and thereby reducing the number of memory transfers from GPU memory to the compute units. Based on our

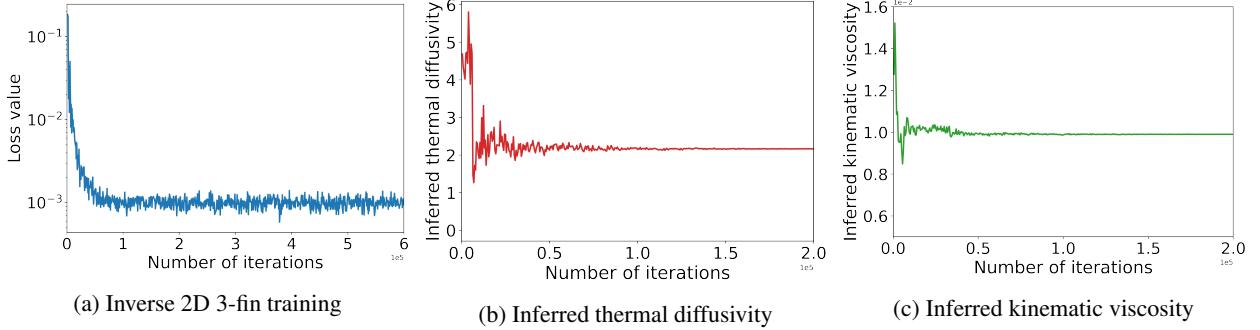


Figure 13: Solution to the 2D 3-fin inverse problem using SimNet.

Table 4: A comparison between the ground truth and the inferred flow characteristics.

Property	OpenFOAM (Ground truth)	SimNet (Predicted)
Kinematic viscosity (m^2/s)	$1.00e - 2$	$9.90e - 3$
Thermal diffusivity (m^2/s)	$2.00e - 3$	$2.16e - 3$

experiments on the FPGA problem, kernel fusion using XLA accelerates a single training iteration in SimNet by up to 3.3x.

6.2 Learning rate scaling in multi-GPU training

An effective way of reducing the time to convergence is to parallelize the training process across multiple GPUs. The most common multi-GPU parallelization strategy is data parallelism where a given global training batch is split into multiple sub-batches for each GPU. Each GPU then performs the forward and backward passes for its sub-batch and the gradients are accumulated across all the GPUs using an allreduce algorithm. This form of data parallelism is the most computationally efficient when the batch size per GPU is kept constant instead of the global batch size.

It was shown in [34] that the total time to convergence can be reduced linearly with the number of GPUs by proportionally increasing the learning rate. However, doing that naively would cause the model to diverge since the initial learning rate can be very high. An effective solution for this is to have an initial warmup period when the learning rate gradually increases from the baseline to the scaled learning rate. SimNet implements a constant and a linear learning rate warmup scheme in conjunction with an exponential decay baseline learning rate schedule, and the details are reported in Appendix D. By running a multi-GPU training, larger batch sizes can be used allowing larger models to be run without increasing the time as shown in Figure 14a. Doing so, the time per iteration remains nearly constant, as shown in Figure 14b. For the multi-GPU cases, the learning rate is gradually increased from the baseline case and this allows the model to train without diverging early on and allows the model to converge faster as a result of the increased global batch size coupled with the increased learning rate. Figure 15a shows the loss function evolution as the number of GPUs is increased from 1 to 16 for the NVSwitch heatsink case.

6.3 TF32 math mode

TensorFloat-32 (TF32) is a new math mode available on NVIDIA A100 GPUs for handling matrix math and tensor operations used during the training of a neural network. With this feature, and based on our experiments on the FPGA heat sink problem, we can obtain up to 1.6x speed-up over FP32 on A100 GPUs and up to 3.1x speed-up over FP32 on V100 GPUs. This allows us to achieve similar accuracy compared to FP32 at a reduced training time, as shown in Figure 15b.

7 Conclusion

SimNet is an end-to-end AI-driven simulation framework with unique, state-of-art architectures that enables rapid training convergence of forward, inverse problems and data assimilation problems for real world geometries and multiple physics types without any training data. SDF is used for loss weighting, which has been shown to significantly improve the training convergence in cases where the geometry has sharp corners and results in sharp gradients in the

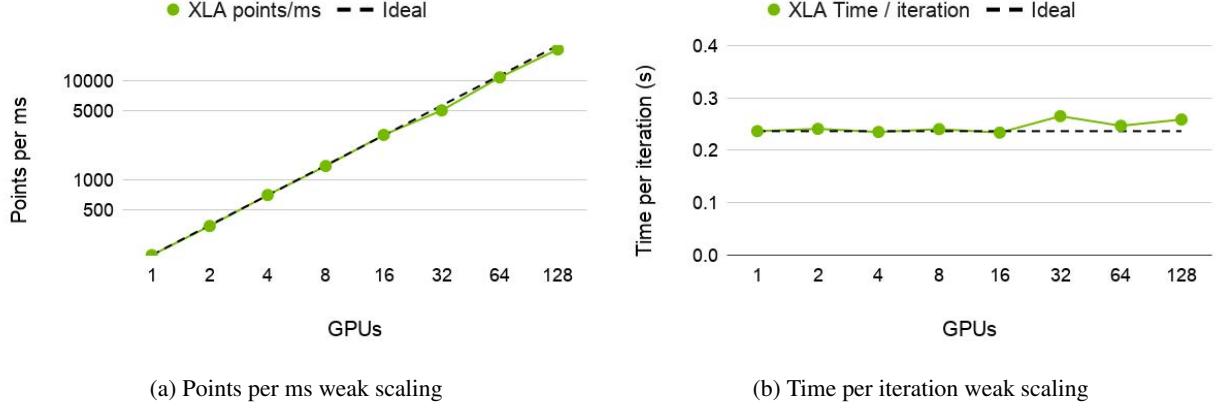


Figure 14: Weak Scaling on V100 GPUs.

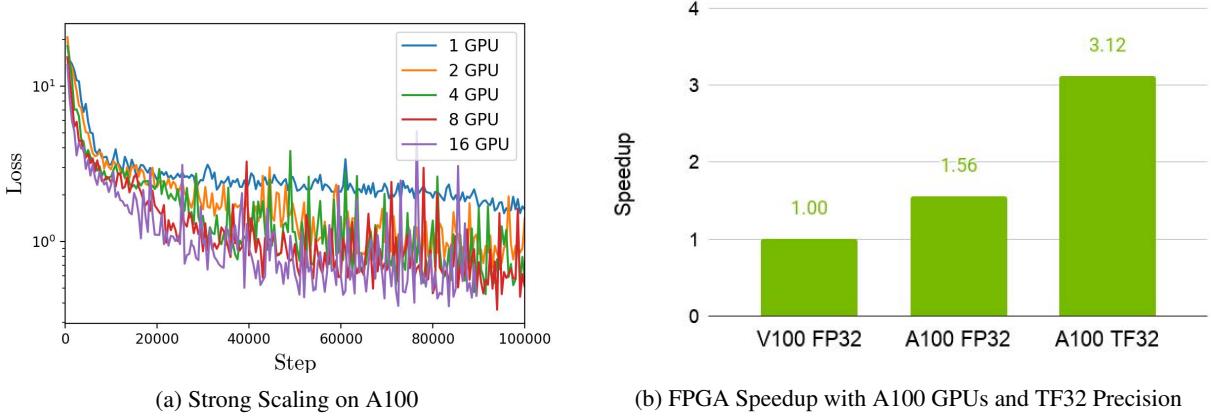


Figure 15: Accelerated training on A100 GPUs.

solution of the differential equation. SimNet’s TG module enables the users to import tessellated geometries from CAD programs. For systems governed by the Navier-Stokes equations, conservation of mass or continuity is imposed globally as well as locally in SimNet to further improve the convergence and accuracy. SimNet enables the neural network solvers to simulate, high Reynolds number turbulent flows for industrial applications. To the authors knowledge, this is the first such application of Physics driven neural networks for turbulent flows.

SimNet is designed to be flexible so that users can leverage the functionality in the existing toolkit and focus on solving their problem well rather than re-creating the tools. To this end, there are various APIs that enable the user to implement their own equations to simulate the physics, implement their own geometry primitives or importing complex tessellated geometries, or implement a variety of domains/boundary conditions. The geometry parameterization in the CSG module allows the neural network to address the entire range of all given parameters in a single training, as opposed to the traditional simulations that run one at a time. The inference for any design configuration can then be completed in real time. This accelerates the simulation with neural network solvers by orders of magnitude.

In a broader context, SimNet provides a framework that is capable of addressing major areas across the computational science and engineering. So far, extensive comparisons of SimNet results with open source and commercial solvers show good correlation. However, further applications across a wide range of use cases are required to verify the robustness, accuracy, and applicability of neural network solvers.

SimNet can be downloaded from: developer.nvidia.com/simnet

Acknowledgments

We would like to thank Doris Pan, Anshuman Bhat, Rekha Mukund, Pat Brooks, Gunter Roth, Ingo Wald, Maziar Raissi and Sukirt Thakur for their assistance and feedback in SimNet development. We also acknowledge Peter Messemer, Mathias Hummel, Tim Biedert and Kees Van Kooten for integration with Omniverse.

References

- [1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [2] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490, 2016.
- [3] Junhyuk Kim and Changhoon Lee. Prediction of turbulent heat transfer using convolutional neural networks. *Journal of Fluid Mechanics*, 882, 2020.
- [4] Oliver Hennigh. Lat-net: compressing lattice boltzmann flow simulations using deep neural networks. *arXiv preprint arXiv:1705.09036*, 2017.
- [5] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [6] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.
- [7] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.
- [8] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *arXiv preprint arXiv:2007.14527*, 2020.
- [9] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv preprint arXiv:2001.04536*, 2020.
- [10] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.
- [11] Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.
- [12] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *arXiv preprint arXiv:2003.05385*, 2020.
- [13] Wei Peng, Weien Zhou, Jun Zhang, and Wen Yao. Accelerating physics-informed neural network training with prior dictionaries. *arXiv preprint arXiv:2004.08151*, 2020.
- [14] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, page 109409, 2020.
- [15] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.
- [16] Craig Michoski, Miloš Milosavljević, Todd Oliver, and David R Hatch. Solving differential equations using deep neural networks. *Neurocomputing*, 2020.
- [17] Ehsan Haghighat, Maziar Raissi, Adrian Moure, Hector Gomez, and Ruben Juanes. A deep learning framework for solution and discovery in solid mechanics: linear elasticity. *arXiv preprint arXiv:2003.02751*, 2020.
- [18] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.
- [19] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *arXiv preprint arXiv:2003.06496*, 2020.
- [20] Yeonjong Shin, Jerome Darbon, and George Em Karniadakis. On the convergence and generalization of physics informed neural networks. *arXiv preprint arXiv:2004.01806*, 2020.
- [21] Lu Lu, Xuhui Meng, Zhiping Mao, and George E Karniadakis. Deepxde: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.
- [22] Ehsan Haghighat and Ruben Juanes. Sciam: A keras wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *arXiv preprint arXiv:2005.08803*, 2020.
- [23] Christopher Rackauckas and Qing Nie. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1), 2017. Exported from <https://app.dimensions.ai> on 2019/05/05.

- [24] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pages 794–803. PMLR, 2018.
- [25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
- [26] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.
- [27] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.
- [28] DL Young, CH Tsai, and CS Wu. A novel vector potential formulation of 3d navier–stokes equations with through-flow boundaries by a local meshless method. *Journal of Computational Physics*, 300:219–240, 2015.
- [29] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310, 2019.
- [30] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*, 2020.
- [31] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.
- [32] Vincent Sitzmann, Julien NP Martel, Alexander W Bergman, David B Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *arXiv preprint arXiv:2006.09661*, 2020.
- [33] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.
- [34] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

A FPGA Problem Description

The dimensions of the FPGA geometry are summarized in the Table 5. Compared to the real FPGA dimensions, these dimensions are normalized such that the channel height is set to 1. The channel walls are treated as adiabatic and the interface boundary conditions are applied at the fluid-solid interface.

Table 5: FPGA Dimensions

Dimension	Value (m)
Heat Sink Base ($\ell \times w \times h$)	$0.65 \times 0.875 \times 0.05$
Fin Dimension ($\ell \times w \times h$)	$0.65 \times 0.0075 \times 0.8625$
Heat Source ($\ell \times w$)	0.25×0.25
Channel ($\ell \times w \times h$)	$5.0 \times 1.125 \times 1.0$

B Aneurysm Problem Description

For the aneurysm simulation, a no-slip boundary condition is applied on the walls of the aneurysm, i.e., $u, v, w = 0$. A parabolic flow is used at the inlet, where the flow moves in the normal direction of the inlet and has a peak velocity of 1.5. The outlet also has a zero pressure condition. The kinematic viscosity of the fluid is set to 0.025, and the fluid density is a constant and is set to 1.0. A total number of about 20M training points are used to solve this problem.

C NVSwitch Problem Description

The inlet velocity to the channel is at 5.7 m/s . The pressure at the outlet is specified as 0 Pa . All the other surfaces of the geometry are treated as no-slip walls. The inlet is at 273.15 K . The channel walls are adiabatic. The heat sink has a heat source of $0.0190 \times 0.0077 \text{ m}^2$ at the bottom of the heat sink situated centrally on the bottom surface. The heat source generates heat such that the temperature gradient on the source surface is 6828.75 K/m in the normal direction. Conjugate heat transfer takes place between the fluid-solid contact surface.

For this problem, we will vary the fin thickness, fin length at front and back, and fin trim angles, as shown in Figure 10. The ranges of variation for these geometry parameters as well as their optimal values are given in Table 6. The maximum allowable pressure drop for this design optimization is assumed to be 103.77 Pa .

Table 6: Range of variability for the NVSwitch fin design variables and their optimal values

Design variable	Thickness level 1	Thickness level 2	Thickness level 3	Front length	Back length	Top front trim angle	Bottom front trim angle	Top back trim angle	Bottom back trim angle
Range	(0.0025, 0.0075)	(0.0025, 0.0075)	(0.0025, 0.0075)	(0.5325, 0.6075)	(0.5325, 0.6075)	(0, $\pi/6$)	(0, $\pi/6$)	(0, $\pi/6$)	(0, $\pi/6$)
Optimal value	0.0067	0.0061	0.0038	0.5951	0.5414	0.0	0.15 π	0.0	0.0

D Learning Rate Linear Warm-up

For the linear warmup scheme, the learning rate η at step s when run with n_g GPUs is given by

$$\eta = \min \{ \eta_w(n_g), \eta_b \}, \quad (13)$$

where η_b is the baseline exponential decay learning rate schedule given by

$$\eta_b = \eta_{0,b} r^{s/s_d} + \eta_{1,b}, \quad (14)$$

and $\eta_w(n_g)$ is the warmup learning rate given by

$$\eta_w(n_g) = \eta_{0,b} \left[1 + (n-1) \frac{s}{s_w} \right]. \quad (15)$$

Here, s_d is the baseline learning rate decay steps, r is the decay rate, $\eta_{0,b}$ and $\eta_{1,b}$ determine the start and end learning rates, and s_w is the number of warmup steps.