An End-to-End AI-Driven Simulation Framework

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

Sanjay Choudhry
schoudhry@nvidia.com

Abstract

We present SimNet, an AI-driven multi-physics simulation framework aiming to accelerate simulations across a wide range of disciplines in science and engineering. Compared to traditional numerical solvers, SimNet offers a wider range of use case addressability including coupled forward, inverse, and data assimilation problems, and is generalizable to multiple configurations enabled through network parameterization. It offers fast turnaround time by enabling parameterized system representation that solves for multiple configurations simultaneously, as opposed to the traditional solvers that need to solve for one configuration at a time. SimNet is highly developer friendly with customizable APIs, and is optimized for high-performance GPU computing with scalable performance for multi-GPU and multi-Node implementation with both FP32 and TF32 computations. In this paper, we introduce SimNet and the various contributions it offers to push the state-of-the-art for neural network solvers. The SimNet source code is available at https://developer.nvidia.com/simnet.

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. Neural network solvers [1–3] not only accelerate these simulations, but also simplify simulation setup and address problems not solvable using traditional solvers, such as inverse problems. Training of these solvers can be supervised only based on the governing laws of a physical process without any training data. Rapid evolution of GPU architecture suited for AI and HPC, and introduction of open source frameworks have motivated researchers to develop novel algorithms (e.g., [4–12]) and libraries [13–15] for neural network solvers. Although the existing research studies and libraries played a crucial role in advancing these solvers, the attempted examples are mostly limited to simple 1D or 2D domains with straightforward governing physics, and these solvers still struggle in addressing real-world applications that involve complex 3D geometries and multi-physics systems. We present SimNet, a new AI-accelerated multi-physics simulation framework based on neural network solvers, that can efficiently and accurately solve real-world coupled forward and inverse problems and is generalizable to parameterized configurations.

Our Contribution. While numerous research studies currently exist to develop methods for solving PDEs using neural networks, these methods do not show success when applied to industrial problems due to gradients and discontinuities introduced by complex geometries or physics. Our main research contribution in this paper is to offer a framework that tackles these challenges by introducing the

Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020), Vancouver, Canada.

use of Signed Distance Functions (SDFs) for loss weighting, integral continuity planes for flow simulation, and advanced neural network solver architectures. Moreover, we adopt the Zero-equation turbulence model to simulate, for the first time, high Reynolds number flows in industrial applications using neural network solvers.

2 SimNet overview

SimNet is an advanced neural network solver built on top of the TensorFlow [16]. It approximates the solution to a PDE by a neural network $u_{net}(\mathbf{x})$ that takes the following form

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

where $\mathbf{x} \in \mathbb{R}^{d_0}$ is the input to the network with n layers, $\phi_i \in \mathbb{R}^{d_i}$ is the output of i^{th} layer, $\mathbf{W}_i \in \mathbb{R}^{d_i\times d_{i-1}}, \mathbf{b}_i \in \mathbb{R}^{d_i}$ are the weight and bias of the i^{th} layer, θ denotes the set of trainable parameters, and σ is the activation function. SimNet offers a large array of activations, including the adaptive ones proposed in [6]. ϕ_E is an input encoding layer and is set to identity function for the standard fully connected architecture, which is the dominant architecture in neural network solvers. To train this neural network, a loss function is constructed that penalizes over the PDE residual of the approximate solution $u_{net}(\theta)$ with constraints encoded as penalty terms, as follows

$$\mathcal{L}_{res}(\theta) = \sum_{i=1}^{N_{\mathcal{N}}}\int_{\mathcal{D}} \lambda_{\mathcal{N}}^{(i)}(\mathbf{x})\left\|r_{\mathcal{N}}^{(i)}\big(\mathbf{x};u_{net}(\theta)\big)\right\|_p d\mathbf{x} + \sum_{j=1}^{N_{\mathcal{C}}}\int_{\partial\mathcal{D}} \lambda_{\mathcal{C}}^{(j)}(\mathbf{x})\left\|r_{\mathcal{C}}^{(j)}\big(\mathbf{x};u_{net}(\theta)\big)\right\|_p d\mathbf{x}, \quad (2)$$

where $N_{\mathcal{N}}, N_{\mathcal{C}}$ are the number of PDEs and constraints, respectively, and $r_{\mathcal{N}}^{(i)}, r_{\mathcal{C}}^{(j)}$ are the PDE and constraint residuals. $\lambda_{\mathcal{N}}^{(i)}, \lambda_{\mathcal{C}}^{(j)}$ are weight functions that control the loss interplay between, within and across different terms. SimNet supports three loss weighting algorithms, that are global learning rate annealing as proposed in [4], a novel local variant of this algorithm that computes the annealing parameters for each network parameter independently, and also a novel SDF weighting which allows for spatial weighting of loss. If the geometry has sharp corners, this often results in sharp gradients in the PDE solution and thus, weighting by the SDF tends to weight these sharp gradients lower. This is found crucial in mitigating the deleterious effects of sharp local gradients and in improving accuracy. The neural network parameters are optimized iteratively using the Adam optimizer [17], and the required gradients are computed via automatic differentiation. At each iteration, the integral terms in the loss function are approximated using a regular or Quasi-Monte Carlo method using a batch of points that is continuously generated at each iteration, or is subsampled from a large initially-generated point cloud. Therefore, the notion of resolution as exists in the traditional solvers does not hold in neural network solvers.

Various features are introduced in SimNet that focus on dramatically reducing the training time and improving the training accuracy, expanding the use case addressability, and promoting the ease-of-use for developers. SimNet offers API-driven functionality, enabling the user to leverage the existing functionality to build their own applications on the existing modules. In SimNet, the geometry and PDE modules are used to fully specify the physical system. The user also specifies the network architecture, optimizer and learning rate schedule. SimNet constructs the neural network solver, forms the loss function, and unrolls the graph efficiently to compute the gradients. 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 the form of CSV or VTK files. SimNet consists of various modules, out of which a few are described in the following.

Geometry modules: SimNet contains Constructive Solid Geometry (CSG) and Tessellated Geometry (TG) modules. With SimNet's CSG module, constructive geometry primitives can be defined and Boolean operations performed. This allows the creation and parameterization of a wide range of geometries. The TG module uses tesselated geometries in the form of STL or OBJ files to work with complex geometries. Both of these geometry modules allow for the SDF and its spatial derivatives to be computed. CSG uses SDF functions to implicitly define the geometry and in the TG module, the SDF is computed using a ray tracing method proposed in [18]. The SDF is needed for loss weighting and for the zero-equation tubulence model considered below.

PDE module: The PDE module in SimNet consists of a variety of differential equations including the Navier-Stokes, diffusion, advection-diffusion, wave, and elasticity equations. To make this module extensible for the user to easily define their own PDEs, SimNet uses symbolic mathematics enabled by SymPy [19]. A novel contribution of SimNet is the adoption of the zero-equation turbulence model, and this is the first time a neural network solver is made capable of simulating flows with high Reynolds numbers, as shown in the next section. Moreover, for fluid flow simulation, we propose the use of integral continuity planes. In addition to solving the Navier-Stokes equations in differential form, specifying the mass flow (for compressible flows) or volumetric flow rate (for incompressible flows) through some integral continuity planes that are located on the outlet and across the channel significantly speeds up the convergence rate and improves accuracy.

Network architectures: In addition to the standard fully connected networks, SimNet offers more advanced architectures, including the Fourier feature and Modified Fourier feature networks, and Sinusoidal Representation Networks (SiReNs) to alleviate the spectral bias [20] in neural networks and improve convergence. The Fourier feature network in SimNet is a variation of the one proposed in [21] with trainable encoding, and takes the form in equation 4 with the following encoding

$$\phi_E = \left[\sin\left(2\pi \mathbf{f} \times \mathbf{x}\right); \cos\left(2\pi \mathbf{f} \times \mathbf{x}\right) \right]^T, \tag{3}$$

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. 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 [4]. 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

$$\phi_i(\mathbf{x}_i) = \left(1 - \sigma\left(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i\right)\right) \odot \sigma\left(\mathbf{W}_{T_1} \phi_E + \mathbf{b}_{T_1}\right) + \sigma\left(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i\right) \odot \sigma\left(\mathbf{W}_{T_2} \phi_E + \mathbf{b}_{T_2}\right), \tag{4}$$

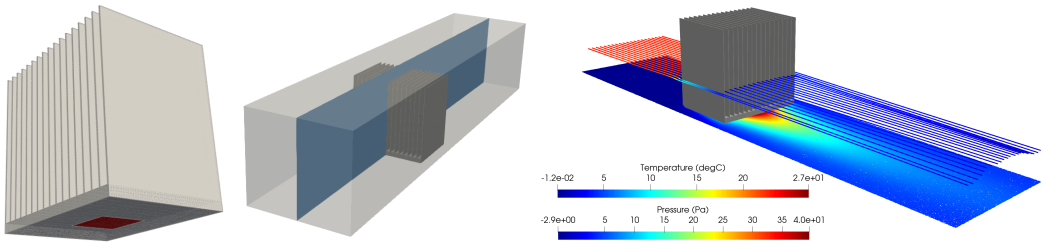
where $i > 1$ and $\{\mathbf{W}_{T_1}, \mathbf{b}_{T_1}\}, \{\mathbf{W}_{T_2}, \mathbf{b}_{T_2}\}$ are the parameters for the two transformation layers.

3 Sample applications

This section illustrates the capability of SimNet in solving real-world problems involving turbulent multi-physics simulations and complex geometries. Two other use cases involving design optimization and inverse simulation are discussed in Appendix A.

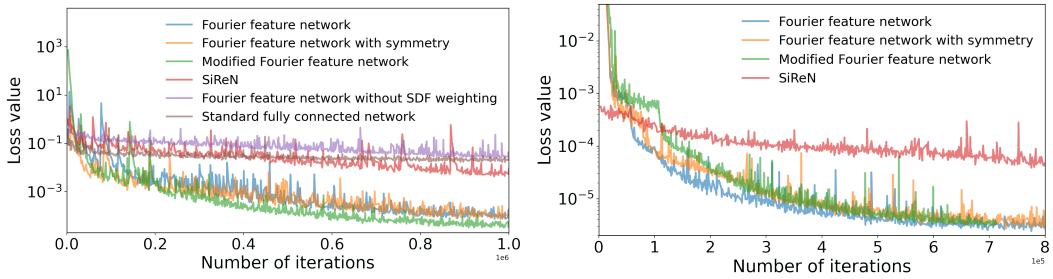
Turbulent & multi-physics simulations: Using an FPGA heat sink example, we demonstrate the SimNet's capability in accurately solving multi-physics problems involving high Reynolds number flows. The heat sink geometry placed inside a channel is depicted in Figures 1a, 1b. This particular geometry is challenging to simulate due to thin fin spacing that causes sharp gradients that are difficult to learn for a neural network solver. Using the zero-equation turbulence model, we solve a conjugate heat transfer problem with a flow at $Re = 13,239$. Generally, simulation of high-Re flows are particularly difficult due to the chaotic fluctuations of the flow field properties that are caused by instabilities in the shear layer. Due to the one-way coupling between the heat and incompressible flow equations, two separate neural networks are trained for flow (trained first) and the temperature (trained next) fields. This approach is useful for one-way coupled multi-physics problems to achieve significant speed-up. Using SimNet, we simulate this conjugate heat transfer problem with different architectures and also with symmetry boundary conditions. Loss curves can be found in Figure 2. Our proposed SDF loss weighting and integral continuity planes are used by default. Figure 2 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 model with modified Fourier feature network are shown in Figure 1c. A comparison between the SimNet and OpenFoam results for flow and temperature fields is also presented in Figure 3.

Blood flow in an Intracranial Aneurysm To demonstrate the capability of SimNet in dealing with real world geometries, using the TG module, we simulate the flow inside a patient-specific geometry of an aneurysm depicted in Figure 4a. Results for the distribution of velocity magnitude



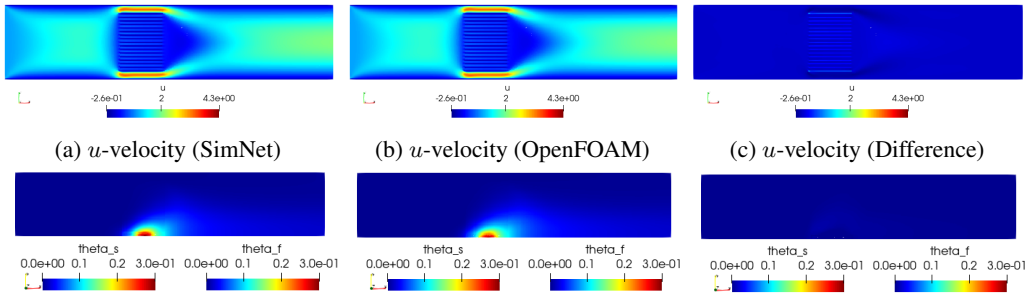
(a) (b) (c)

Figure 1: FPGA heat sink example. (a) FPGA heat sink geometry; (b) Simulation domain (blue plane represents the symmetry plane); (c) SimNet results for streamlines and temperature profile.



(a) FPGA flow training (b) FPGA heat training

Figure 2: Loss curves for FPGA conjugate heat transfer training using different architectures.



(a) u-velocity (SimNet) (b) u-velocity (OpenFOAM) (c) u-velocity (Difference)

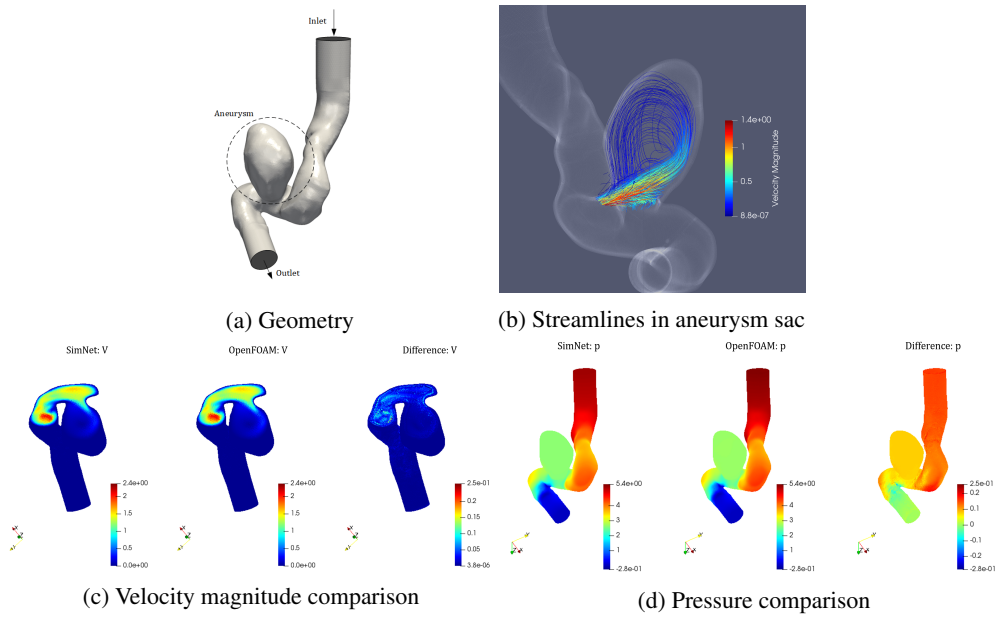
(d) Temperature (SimNet) (e) Temperature (OpenFOAM) (f) Temperature (Difference)

Figure 3: 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.

and pressure developed inside the aneurysm are shown in Figures 4c and 4d, respectively. Using the same geometry, the authors in [9] 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 4b shows that SimNet successfully captures the flow field.

4 Performance upgrades and multi-GPU/multi-node training

SimNet supports multi-GPU/multi-node scaling to enable larger batch sizes while time per iteration remains nearly constant, as shown in Figure 5a. Therefore, the total time to convergence can be reduced by scaling the learning rate linearly with the number of GPUs, as suggested in [22]. Doing so without a warmup would cause the model to diverge since the initial learning rate can be very large. SimNet implements constant and linear learning rate warmup schemes in conjunction with an exponential decay schedule. SimNet also supports TensorFloat-32 (TF32), a new math mode



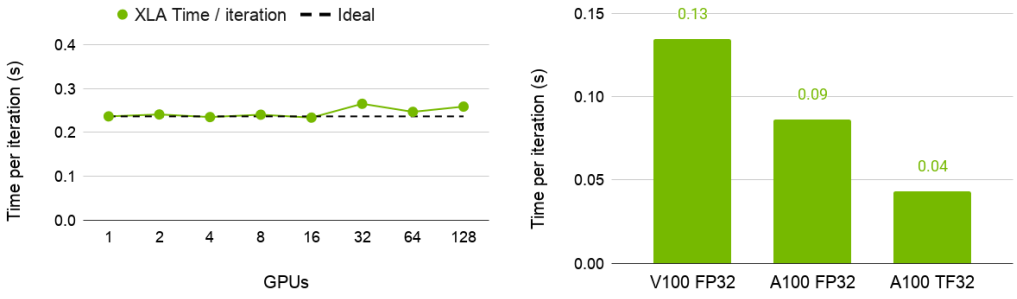
(a) Geometry

(b) Streamlines in aneurysm sac

(c) Velocity magnitude comparison

(d) Pressure comparison

Figure 4: SimNet simulation results for the aneurysm problem.



(a) Time per iteration weak scaling

(b) FP32/TF32 Time per iteration

Figure 5: SimNet performance upgrades using multi-GPU/multi-node training and TF32.

available on NVIDIA A100 GPUs. Based on our experiments on the FPGA problem, using TF32 provides up to 1.6x and 3.1x speed-up over FP32 on A100 and V100 GPUs, respectively (Figure 5b).

5 Conclusion

We presented SimNet, an end-to-end AI-driven simulation framework with unique, state-of-art architectures to enable accelerated training convergence of forward and inverse problems for real-world geometries and multi-physics compared to standard neural network solvers. We introduced SDF for loss weighting, which significantly improves the convergence for cases with sharp gradients in the solution. We also introduced the use of integral continuity planes to improve the convergence of fluid flow training. Additionally, SimNet enables the neural network solvers to simulate, for the first time, high Reynolds number flows. A novel modified Fourier feature network was also introduced that outperformed other considered fully connected architectures in solving a multi-physics problem. Although SimNet is capable of simulating transient flows using the continuous-time sampling approach [2], a more efficient and accurate approach based on the convolutional LSTMs will be developed and integrated with SimNet in a future work.

Broader impact

In a broader context, SimNet provides a framework that is capable of addressing major areas across the computational science and engineering, including design space exploration and optimization, improved multi-physics simulations and predictions, inverse modeling and uncertainty quantification, and real-time simulations. Although the area of neural network solvers has seen significant development and progress in the past four years, it is still a new area with major universities and research centers continuing to advance the technology. More experiments across a wide range of applications are required to verify the robustness, accuracy, and applicability of neural network solvers.

Acknowledgements

We would like to thank Doris Pan, Anshuman Bhat, Rekha Mukund, Pat Brooks, Gunter Roth, Maziar Raissi and SukirtThakur 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] 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.

[2] 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.

[3] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[4] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv preprint arXiv:2001.04536*, 2020.

[5] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[6] 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.

[7] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *arXiv preprint arXiv:2003.05385*, 2020.

[8] 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.

[9] 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.

[10] Craig Michoski, Miloš Milosavljević, Todd Oliver, and David R Hatch. Solving differential equations using deep neural networks. *Neurocomputing*, 2020.

[11] 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.

[12] 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.

[13] 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.

[14] Ehsan Haghighat and Ruben Juanes. Sciann: A keras wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *arXiv preprint arXiv:2005.08803*, 2020.

[15] 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.

[16] 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.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[18] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003.

[19] 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.

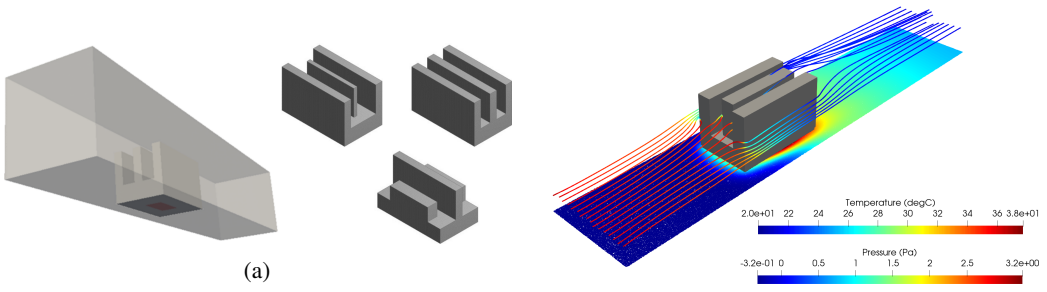
[20] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310, 2019.

[21] 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.

[22] 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 Other sample applications

Design optimization for multi-physics systems: 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. This is done by randomly sampling from the design parameter space at each iteration. Therefore, once the training is complete, several parameter combinations can be evaluated without solving the forward problem again. Such throughput enables more efficient design optimization and space exploration tasks for complex systems in science and engineering. Here, we train a conjugate heat transfer problem over a 3-fin heat sink whose six fin geometry parameters are variable. Following the training, we perform a design optimization to find out the most optimal fin structure by computing the flow and temperature fields given each design parameter on a grid of size 4^6. Figure 6a shows the geometry of a 3-fin heat sink placed inside a channel and some examples of the 3-fin geometries. Figure 6b shows the streamlines and temperature profile for the optimal 3-fin geometry. The total compute time required by OpenFOAM for this design optimization task (for 4096 simulations) is about 190x the SimNet's total compute time.

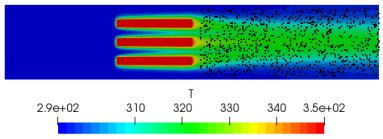


(a) (b)

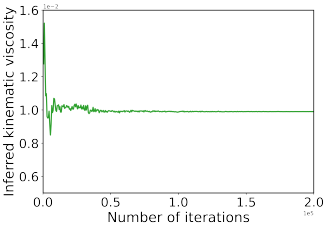
Figure 6: (a) 3-fin geometry; (b) Streamlines and temperature profile for the optimal 3-fin geometry.

Inverse problems: Inverse problems usually involve solving for the latent physics using the PDEs as well as the data. Here, we demonstrate the capability of SimNet in solving 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 that can be measured, the task is to infer

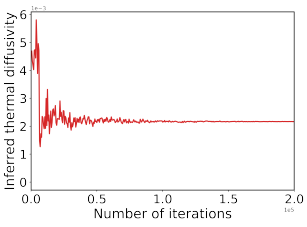
flow viscosity and thermal diffusivity. In reality, the data is collected using measurements but for the purpose of this example, synthetic data generated by OpenFOAM is used, as shown in Figure 7a. The flow viscosity and thermal diffusivity used for the OpenFOAM simulation are 0.01 and 0.002, respectively. We solve this problem by constructing a neural network model with a hybrid data and physics-driven loss function. The quantities of interest are also modeled as trainable variables, and their convergence is shown in Figures 7c,7b. The relative errors for the final inferred values are less than 8%.



(a) A batch of training points



(b) Inferred kinematic viscosity



(c) Inferred thermal diffusivity

Figure 7: Solution to the 2D 3-fin inverse problem using SimNet.