Poisson CNN: convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh

Ali Girayhan Özbay, Arash Hamzehloo, Sylvain Laizet, Panagiotis Tzirakis, Georgios Rizos, Björn Schuller

Angaben zur Veröffentlichung / Publication details:

Özbay, Ali Girayhan, Arash Hamzehloo, Sylvain Laizet, Panagiotis Tzirakis, Georgios Rizos, and Björn Schuller. 2021. "Poisson CNN: convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh." *Data-Centric Engineering* 2: e6. https://doi.org/10.1017/dce.2021.7.

Data-Centric Engineering (2021), 2: e6
doi:10.1017/dce.2021.7

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Poisson CNN: Convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh

Ali Girayhan Özbay[1], Arash Hamzehloo[1], Sylvain Laizet[1], Panagiotis Tzirakis[2], Georgios Rizos[2] and Björn Schuller[2]

[1]Department of Aeronautics, Imperial College London, South Kensington Campus, London, United Kingdom
[2]Department of Computing, Imperial College London, South Kensington Campus, London, United Kingdom
*Corresponding author. E-mail: aligirayhan.ozbay14@imperial.ac.uk

Received: 09 September 2020; **Revised:** 29 March 2021; **Accepted:** 29 May 2021

Keywords: Convolutional neural network; partial differential equations; Poisson equation

Abstract

The Poisson equation is commonly encountered in engineering, for instance, in computational fluid dynamics (CFD) where it is needed to compute corrections to the pressure field to ensure the incompressibility of the velocity field. In the present work, we propose a novel fully convolutional neural network (CNN) architecture to infer the solution of the Poisson equation on a 2D Cartesian grid with different resolutions given the right-hand side term, arbitrary boundary conditions, and grid parameters. It provides unprecedented versatility for a CNN approach dealing with partial differential equations. The boundary conditions are handled using a novel approach by decomposing the original Poisson problem into a homogeneous Poisson problem plus four inhomogeneous Laplace subproblems. The model is trained using a novel loss function approximating the continuous L^p norm between the prediction and the target. Even when predicting on grids denser than previously encountered, our model demonstrates encouraging capacity to reproduce the correct solution profile. The proposed model, which outperforms well-known neural network models, can be included in a CFD solver to help with solving the Poisson equation. Analytical test cases indicate that our CNN architecture is capable of predicting the correct solution of a Poisson problem with mean percentage errors below 10%, an improvement by comparison to the first step of conventional iterative methods. Predictions from our model, used as the initial guess to iterative algorithms like Multigrid, can reduce the root mean square error after a single iteration by more than 90% compared to a zero initial guess.

Impact Statement

The Poisson equation is one of the most computationally intensive partial differential equations to solve, requiring very expensive iterative methods. We propose a novel and flexible CNN architecture which can either serve as an alternative to existing iterative methods on Cartesian grids, or augment them. Outperforming existing NN models, the proposed framework can deal with different boundary conditions and domain aspect ratios, without the need for re-training as long as boundary condition types (Dirichlet, Neumann, etc.) are unchanged. The proposed model can enable engineers to run simulations faster, for instance in computational fluid dynamics, where a Poisson equation must to be solved at each time step of the simulation for incompressible flows.

1. Introduction

Partial differential equations (PDEs) describe complex systems in many fields of engineering and science, ranging from fluid flows to options pricing. Despite their ubiquity, they can be very costly to solve accurately. One of the most important PDEs in engineering is the Poisson equation, expressed mathematically as

$$\nabla^2\phi = f, \tag{1}$$

where f is a forcing term and ϕ is the variable for which a solution is sought. Appearing in a diverse range of problems including heat conduction, gravitation, simulating fluid flows, and electrodynamics, the Poisson problem plays a central role in the design of many modern technologies. However, solving the Poisson equation numerically for large problems involving millions of degrees of freedom is only tractable by employing iterative methods. For example, it is well known that the treatment of incompressibility is a real difficulty to obtain solutions of the incompressible Navier–Stokes equations, the mathematical model used to describe the motions of a turbulent flow. Any algorithm must ensure a divergence-free flow field at any given time during the calculation. The unavoidable step of solving the Poisson equation, as introduced by a fractional step method per Chorin (1967, 1968, 9) can be computationally very expensive (it can represent up to 90% of the computational effort), since it is often based on the said complex iterative techniques.

One of the fastest algorithms for solving the Poisson equation iteratively, with $O(n)$ complexity per Lui (2011), is the multigrid algorithm whereby the problem is solved using a number of successively coarser grids to eliminate low wavenumber error in a few iterations. However, even the multigrid algorithm can take prohibitively long amounts of time to iterate for very large problems. Convolutional neural networks (CNNs) are well positioned to accelerate such iterative methods; they are already in use in various areas of engineering and applied mathematics for complex regression and image-to-image translation tasks, have $O(n)$ runtime complexity[1] and can be run very efficiently on graphics processing units (GPUs) and other floating point acceleration hardware.

Leveraging these strengths, our principal motivation is to develop a CNN-based Poisson equation solver that does not require re-training to perform inference on inputs with different types of boundary conditions (BCs) within a given range of grid resolutions and sizes in the context of Cartesian grids. In this work, we present a novel CNN architecture capable of handling arbitrary right-hand side (RHS) functions f and BCs of a given type on rectangular two-dimensional grids of different aspect ratios but uniform grid spacing Δ. The proposed model is also included in a CFD solver to demonstrate its potential to provide an accurate initial guess (more accurate than the first step of conventional iterative methods), so that the rate of convergence can be dramatically increased.

The outline of this paper is as follows: in Section 2, we provide a brief summary of the recent advancements and ongoing challenges in applying neural networks (NNs) to solve the Poisson equation and more broadly to solve general PDEs. Then, we propose a novel BC handling strategy in Section 3 and the related model architecture in Section 4. The details of our dataset generation method, novel loss and training process are provided in Section 5, Section 6, and Section 7, respectively. The results and performance of the new model are showcased and discussed in Section 8. Finally, conclusions are drawn and plans for future work are outlined.

2. Related Work

Interest in solving PDEs using NN-based methods has a relatively long history, beginning in the 1990s with efforts by Lee and Kang (1990), Dissanayake and Phan-Thien (1994), and Lagaris et al. (1998). A

[1] Convolutional neural networks have $O(n)$ (where n is the product of the input size across each dimension) runtime complexity, as convolution is in fact equivalent to a banded matrix–vector product, where the banded matrix has the (flattened) convolution weights in each row stored across the appropriate diagonals, which is known to be an $O(n)$ operation.

significant proportion of early works on using NNs to approximate the solutions of PDEs focus on approximating the solution ϕ given the variables it depends on, such as spatial coordinates, treating the NN as a continuous function. Training is performed to minimize the solution residuals inside the domain and on the boundaries. For example, Lagaris et al. (1998) utilize a single layer perceptron (constrained by the computational power limitations of the time) to augment trial functions known to satisfy the equation in question to solve a number of benchmark problems in numerical analysis, such as the Poisson equation subject to both (fixed) Dirichlet and Neumann BCs using various RHS functions.

This rather intuitive application of NNs greatly benefits from their differentiability via the back-propagation algorithm since it provides a very accurate method to compute the residuals. Combined with other factors such as the accurate results achieved for specific problems using relatively few parameters, the computational constraints of the era and the relative infancy of more advanced NN architectures in use today, this method became the dominant approach in the early attempts to bring the fields of PDEs and NNs together. However, it suffered from the rather severe drawback of each set of trained weights being able to handle only a specific RHS function and set of BCs.

With the increase in computation power through the 2000s, more complicated models with more parameters and multiple layers became the norm. Smaoui and Al-Enezi (2004) used the greater amounts of computational power available to investigate deeper models, utilizing multilayer perceptrons to predict the proper orthogonal decomposition modes of the one-dimensional (1D) Kuramoto–Sivashinsky equation and the two-dimensional (2D) Navier–Stokes equation. Similarly, Baymani et al. (2010) used multilayer perceptrons to compute the solution of the Stokes equation by decomposing it into multiple Poisson problems and solving the Poisson problems with a procedure similar to the works discussed above. Furthermore, research into different neural computation methods became more popular, leading to works such as those using Radial Basis Function NNs by Jianyu et al. (2003) and by Mai-Duy and Tran-Cong (2001). A more detailed survey of the methods of the era was published by Kumar and Yadav (2011).

Research into the applications of NNs to solve PDEs is still gaining significant momentum, driven by a number of factors providing a supporting ecosystem for deep learning research in general. Substantial improvements in the hardware used to run NN training and inference, discovery of better practices to train NNs, a culture of making publications available in open-access repositories and releasing source code using free and open-source licences in the research community help maintain a substantial growth rate. A brief overview of notable approaches being investigated in a modern setting is provided in the following subsections.

2.1. Modern approaches to solving PDEs using NNs

The above factors led to great strides in models using the older, "continuous" paradigm of training multilayer perceptron-style NNs to minimize solution residuals. Such models can now tackle a much greater variety of PDEs, solved in complex domains with a variety of BCs. A prominent example of advances in this area in recent years is the advent of "physics informed neural networks (PINNs)" by Raissi et al. (2017a) who demonstrated the use of such a methodology to solve the Schrödinger and Burgers equations, the latter of which is of particular note due to the presence of discontinuities. Based on that paradigm, Lu et al. (2021) developed the DeepXDE library, capable of solving a wide range of differential equations including partial- and integro-differential equations, providing a more user-friendly way of using NNs in this context. Furthermore, Meng and Karniadakis (2019) proposed a method to help PINNs predict the correct values in a problem when given a combination of abundant yet less accurate "low-fidelity" plus sparse yet more accurate "high-fidelity" data.

Another subject that attracted substantial attention in the recent years is using NNs to "discover" the underlying PDE describing (e.g., a physical system) from an existing dataset. In such a system, the NN tries to find the numerical coefficients of a previously assumed PDE form by minimizing residuals. Notable examples include works by Long et al. (2018), who adopted a convolutional methodology to first discover the coefficients for and then predict the time evolution of a 2D linear diffusion equation, and also

a number of works by Raissi et al. (2017b) and Raissi (2018), with examples of applications to, among others, the Burgers and Navier–Stokes equations. It is further noteworthy that interest in this subject extends beyond the use of NNs, as evidenced by works such as those from Rudy et al. (2016) using sparse regression.

Meanwhile, separate from the initial approaches to applying NNs to PDEs described above, fully convolutional models that capitalize on significant strides made in computer vision using CNNs began to gain traction. Utilizing common techniques in computer vision by treating the known terms and solutions of PDEs on rectangular grids as though they were images, such methods approach the task of solving a PDE as an image-to-image translation problem similar to the pix2pix method by Isola et al. (2016).

2.2. CNNs and the Poisson equation

A significant proportion of the efforts to use CNNs to solve PDEs has focused on the Poisson equation, considering its status as a well-understood benchmark problem with applications to many fields as mentioned earlier.

In the fluid mechanics community, works of Tompson et al. (2016) and Xiao et al. (2018) pioneered the usage of CNNs to solve the Poisson equation. While both use their models within the framework of a complete CFD solver to simulate the motion of smoke plumes around objects, the architectures used and training methodologies are different. The former developed a model which takes a 3D array containing the velocity divergence and geometry information on cubic $128 \times 128 \times 128$ grids while the latter adopt a multigrid-like strategy with multiple discretizations to make predictions on larger $384 \times 384 \times 384$ grids. Moreover, the former approach trains the NN to minimize the divergence of the velocity field only while the latter adopts a more direct strategy by trying to instead minimize a linear combination of the L2 norms of the velocity divergence and the discrepancy between the predicted and ground truth pressure correction values by leveraging the additional data available from the specific methodology. Building on the strategy developed in these works, Ajuria Illarramendi et al. (2020) used a CNN to handle the Poisson solver step of CFD simulations of plumes and flows around cylinders, demonstrating stable and accurate time evolution even for Richardson numbers greater than in the training data when applied in combination with several Jacobi iterations. Outside fluid mechanics, Shan et al. (2017) investigated the application of a fully convolutional NN to predict the electric potential on cubic $64 \times 64 \times 64$ grids given the charge distributions and (constant) permittivities, claiming average relative errors below 3% and speedups compared to traditional methods.

In general, all of these works attempt to predict the solution of the Poisson equation given an array containing the values of the RHS function on a grid. In the present study, we propose a fully convolutional NN architecture that can handle arbitrary BCs, on grids with different aspect ratios and uniform grid spacing. The mathematical formulation and the neural architecture of the proposed approach are discussed in the next two sections.

3. Mathematics of the Proposed NN Architecture

Since the Poisson problem does not have a unique solution when BCs are absent, a way to include and process BC information alongside the RHS is required to obtain a model that is able to solve the Poisson problem with arbitrary BCs (as opposed to training different models tailored for one specific set of BCs). Matrix-based methods such as successive over-relaxation and direct solution handle the BCs by augmenting the RHS vector. Conversely, an NN must process boundary information more explicitly by integrating it into the model architecture. Following from John (1982) and assuming arbitrary Dirichlet or Neumann BCs, the proposed methodology involves splitting the original (2D) Poisson problem into one Poisson problem with homogeneous (zero Dirichlet) BCs plus four Laplace problems where each Laplace problem has three homogeneous BCs plus one inhomogeneous Dirichlet or Neumann BC identical to one of the BCs in the original problem on the corresponding boundary.

Formally, for Dirichlet BCs, if we consider the original problem on a rectangular domain D, denoting an edge of the domain with the index k as $[\partial D]_k$ and the corresponding boundary condition on the edge as g_k,

$$\nabla^2\phi = f \wedge \phi\left([\partial D]_1\right) = g_1 \wedge \phi\left([\partial D]_2\right) = g_2 \wedge \phi\left([\partial D]_3\right) = g_3 \wedge \phi\left([\partial D]_4\right) = g_4, \tag{2}$$

we can rewrite it in the following form, exploiting the linearity of the Laplace operator

$$\nabla^2(\phi_h + \phi_{BC0} + \phi_{BC1} + \phi_{BC2} + \phi_{BC3}) = f, \tag{3}$$

such that

$$\nabla^2\phi_h = f \wedge \phi_h(\partial \mathrm{D}) = 0, \tag{4}$$

$$\nabla^2\phi_{BCk} = 0 \wedge \phi_{BCk}\left([\partial D]_j\right) = \begin{cases} g_k & j = k \\ 0 & j \neq k \end{cases}, j,k \in [0,1,2,3]. \tag{5}$$

Thus, it is possible to solve the original problem by first solving the Poisson equation with the original RHS but zero BCs to find ϕ_{h}, then solving Laplace problems for each BC to find ϕ_{BCk} and summing these results. Similarly, the proposed NN architecture will be composed of two parts—one which solves the homogeneous Poisson problem and another which solves the Laplace problem given one BC. It will be shown in Section 8.6 that this approach is superior to designing a single NN for an inhomogeneous Poisson problem. It also offers more flexibility for handling various types of BCs. Such a feature is not available in the geometry-restricted strategies seen in previous works. It means that the proposed model can tackle any Poisson problem with the appropriate BCs.

4. Proposed NN Architecture

Figure 1 provides an overview of the high-level structure of the NN architecture proposed in the current study. The NN components of the model are the blocks marked as Dirichlet BC NN (DBCNN), which approximates the solution of the Laplace equation with one inhomogeneous Dirichlet boundary, and the homogeneous Poisson NN (HPNN) which approximates the solution of the Poisson equation with homogeneous BCs. The architecture mirrors the decomposition in Equations (2)–(5). First, the DBCNN submodel makes predictions for the four Laplace problems and applies reflection and rotation operations such that the inhomogeneous boundaries align with the corresponding boundary in the original problem. Then, the HPNN model makes a prediction for the Poisson problem with homogeneous BCs. Finally,

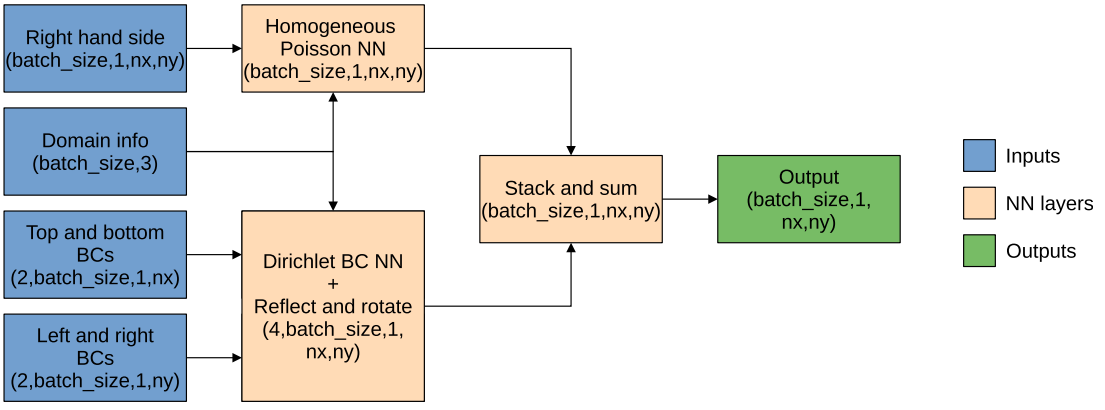

Figure 1. *High-level diagram of Poisson convolutional neural network (CNN). Variable names in parentheses in each block indicate the shape of the output of the block.*

these results are summed to obtain the final prediction for a Poisson problem with four inhomogeneous DBCs.

4.1. Homogeneous Poisson NN (HPNN)

The HPNN estimates the solution of the Poisson problem with homogeneous BCs and was inspired by the Fluidnet architecture by Tompson et al. (2016). The model has two inputs: the RHS and the domain information (grid spacing, domain sizes per spatial dimension). Figure 2 provides an overview of the model architecture. Our model expands upon the Fluidnet model by adding ResNet blocks (first proposed by He et al., 2015) after most convolutions, incorporating a substantially larger number of independent pooling operations and using dense layers as well as positional embeddings[2] to handle different grid resolutions, grid sizes, and aspect ratios.

In the present study, it was found that the additional model architecture features lead to substantial gains in accuracy for relatively little increases in runtime. A more detailed investigation of the improvements afforded by the novel model features can be found in Section 8.6 with an ablation study. Further investigations indicated that without incorporating a mechanism to handle domain and grid spacing information as with the positional embeddings and the feedforward layers, the model learns to merely reproduce the solution profile with an average peak magnitude value, substantially reducing its usefulness. The necessity to supply the grid spacing is evident from the fact that this is necessary for classical methods as well; on a grid, the finite difference approximation for the Laplacian operator depends on the Δ^2 factors in the denominator and hence this information is needed to reverse the operation. On the other hand, while there is no such similar absolute necessity to supply the domain size information (and indeed classical methods do not require this information at all), in practice it was found to greatly boost the performance of the model since it enables the model to adjust its output to different aspect ratios more easily.

When processing the RHS, first the data are passed through several convolutions. Then, the computation is split into multiple independent pooling operations, each of which applies average pooling of progressively larger pool sizes to capture lower-wavenumber modes and applies further convolutions to the pooled results. Then, the pooled results are upsampled using either a polynomial reconstruction

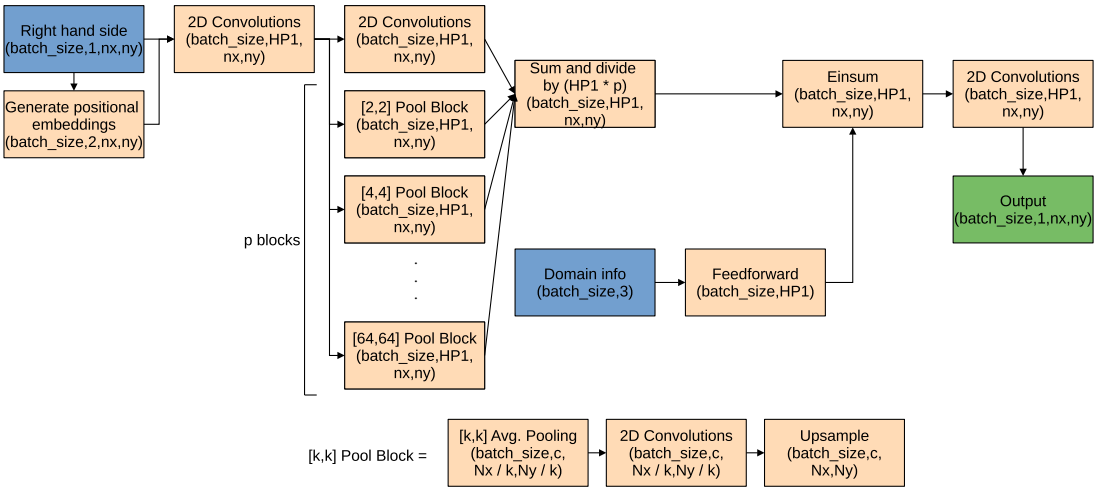

Figure 2. *Homogeneous Poisson neural network (NN) diagram. Variable names in parentheses in each block indicate the shape of the output of the block. Variables named "HP* " are hyperparameters.*

[2] Positional embeddings are created by concatenating a cosine wave $\cos(\pi x_i / L_{x_i})$, tiled to match the input grid shape, in each direction to the right hand side in the channel dimension.

method (in the case of the pooling threads with the largest two pool sizes) or transposed convolutions (for branches with smaller pool sizes). Using polynomial interpolation based upsampling methods for pooling branches with large pool sizes was observed to reduce artefacting in the output; upsampling, for example, a 2×2 input generated using 128×128 pooling from a 200×200 source image requires using a stride[3] of 128 for the transposed convolution to upsample to the original size, which is excessively large. Finally, the results from all branches are summed and are divided by the number of branches times the number of channels in each branch.

The domain information (i.e., Δ, L_x, and L_y) is processed by several dense layers. The RHS and domain information branches are subsequently merged by multiplying every channel from the RHS branch by one of the outputs of the domain information branch, expressed using tensor notation (without the implicit summation) as

$$A_{ijkl}B_{ij} = C_{ijkl}, \tag{6}$$

where index i is the batch dimension, index j is the channel dimension, and the remaining indices are for the spatial dimensions. Finally, several more convolutions are applied to the result from Equation (6), reducing the final number of channels to 1 to produce the solution.

4.2. Dirichlet boundary condition NN (DBCNN)

The DBCNN estimates the solution of the Laplace equation with one inhomogeneous DBC. As inputs, it takes one 1D array containing the BC information, the same domain information used for the HPNN and the number of grid points in the direction orthogonal to the provided BC. Figure 3 further details the operation of the model.

The difficulty of solving this problem is constructing the solution, which is a 2D scalar field, from the BC information which is only a 1D scalar field. To overcome this issue, a known mathematical property of the solutions of the Laplace equation in this setting is exploited; given homogeneous DBCs on three boundaries plus one inhomogeneous Dirichlet BC on another on a rectangular domain, the solution of the Laplace equation on the domain $[0, L_x] \times [0, L_y]$ can be written as a series of the form

$$\phi = \sum_{m=1}^{\infty} A_m \sinh\left(\frac{m\pi(x - L_x)}{L_x}\right) \sin\left(\frac{m\pi y}{L_y}\right), \tag{7}$$

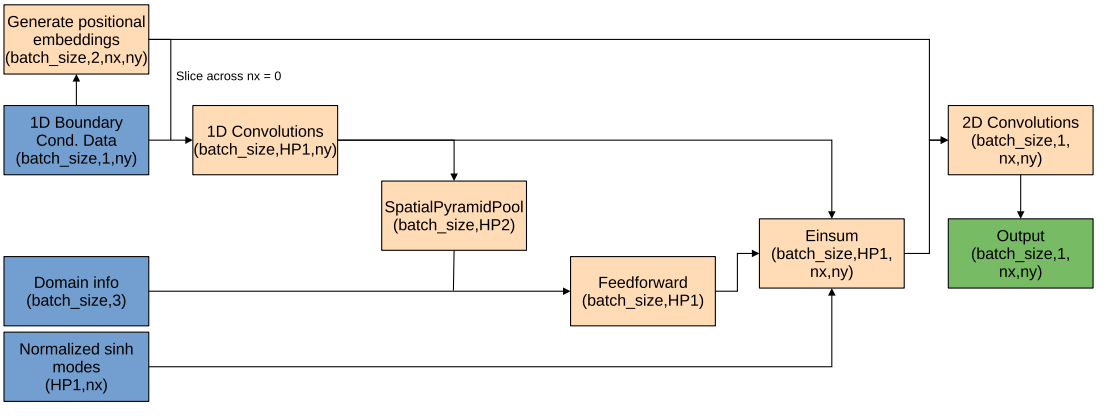

Figure 3. *Dirichlet boundary condition neural network (NN) diagram. Variable names in parentheses in each block indicate the shape of the output of the block. Variables named "HP*" are hyperparameters.*

[3] Please see Dumoulin and Visin (2016) for further details.

when the nonhomogeneous boundary is placed at $x = 0$. Thus, we can say that the solution along any constant $y = \widetilde{y}$ can be written as

$$\phi(x, \widetilde{y}) = \sum_{m=1}^{\infty} B_m \sinh\left(\frac{m\pi(x - L_x)}{L_x}\right). \tag{8}$$

Thus, to reconstruct the solution, the B_m values for each gridpoint in the y direction are required. Naturally, as is common practice for numerical algorithms, a cutoff point n_c is chosen for the series. The DBCNN model works by first performing a series of 1D convolutions on the BC data to increase the number of channels to n_c. Following that, spatial pyramid pooling as introduced by He et al. (2014) is performed on this result and the fixed-size vector from this operation is concatenated with the domain/grid information. This vector is processed through a feedforward neural network, the final layer of which has n_c units. As the third step, the sinh modes $\sinh(m\pi(x - L_x)/L_x)$ are constructed and are normalized such that the maximum absolute value in each mode is 1.0. Finally, a tensor product between the results in the first three steps is performed to create a batch of (n_c, n_x, n_y) sized tensors and a number of 2D convolutions are performed to reduce the number of "channels" n_c to 1.

It should be noted that this model returns results such that the inhomogeneous BC is always on the left boundary of the domain. However, a series of rotations and reflections are sufficient to modify the result such that any other boundary is the inhomogeneous boundary instead.

5. Dataset

The dataset plays a fundamental role when training NNs, since NNs learn to reproduce the conditional probability distribution of the data as outlined by Bishop (2006). Hence, training on a dataset that does not reflect the probability distribution of the problem in question will give inaccurate results when performing inference. This is especially important when the model will be trained on synthetic data as with our approach, since the onus is on the user to ensure the training set reflects the conditional PDF of the "real" data.

Furthermore, the Poisson problem in the most general case will have source functions and solutions that are finite yet unbounded, which present significant obstacles for NNs. Hence, there is a need to normalize the dataset in a way that does not lead to a loss of generality in the predictive performance of the model while constraining the range of the inputs and outputs to a well-defined and consistent interval. Fortunately, the linearity of the Laplace operator allows us to do this by merely scaling the RHS and the solution by some coefficient α:

$$\nabla^2\phi = f \Rightarrow \alpha\nabla^2\phi = \nabla^2(\alpha\phi) = \alpha f. \tag{9}$$

A similar argument may be made to show that scaling the DBCs for such a Poisson problem by a constant results in the solution of the corresponding Laplace problem (as shown in Equation 5) being scaled by the same constant. In this work, it was chosen to constrain the source functions and BCs such that their maximum absolute value is always 1.0. Different training data generation methods were adopted for the two sub-models, detailed below.

5.1. Training data for HPNN

The HPNN submodel was trained on an analytically generated dataset, composed of synthetically generated solution-RHS pairs based on truncated Fourier and polynomial series. First, the solutions were generated as the sum of a Fourier series and a polynomial, both with random coefficients

$$\phi = \phi_F + \phi_P. \tag{10}$$

Given homogeneous DBCs, the Poisson equation on a rectangular domain can be shown to have solutions of the form

$$\phi = \sum_{m,n} A_{mn} \sin\left(\frac{m\pi x}{L_x}\right) \sin\left(\frac{n\pi y}{L_y}\right), \tag{11}$$

where, for example, L_x indicates the domain length in the x direction. It should be noted that that the Laplacian of such a function will correspond to an RHS which is also the sum of products of sine waves. Using such a truncated Fourier series however results in RHS=0 for all grid points on the boundary—a loss of generality. The RHS need not be zero on the boundaries when solving a Poisson problem analytically as an infinite Fourier series can be fitted for a discontinuous function, Gibbs' phenomenon notwithstanding. However, since a truly infinite series cannot be computed on a computer, the solution was instead augmented by a polynomial component

$$\phi_P = \prod_i p_i(x_i) = p_x(x) p_y(y). \tag{12}$$

The multidimensional polynomial ϕ_P is generated by multiplying 1D polynomials together, benefiting from the variable separable nature of the Poisson equation. Each polynomial component was created by randomly choosing roots r within the extent of the domain in the respective dimension plus a random coefficient B, reserving two of the roots for the extrema of the domain

$$p_i(x_i) = B_i(x_i - 0)(x_i - L_i)(x_i - r_{i,0})(x_i - r_{i,1})\ldots \tag{13}$$

A similar methodology was followed for the Fourier series counterpart with random coefficients C, again utilizing the variable separability

$$\phi_F = \prod_{\mathrm{i}} s_i(x_i), \tag{14}$$

$$s_i(x_i) = \sum_k C_{k,i} \sin\left(\frac{k\pi x_i}{L_i}\right). \tag{15}$$

Subsequently, the peak magnitudes of the two components $\max(|\phi_F|)$ and $\max(|\phi_P|)$ were equalized to ensure neither component dominates the generated dataset. The corresponding right hand sides were calculated by appropriately adjusting the coefficients in the case of ϕ_F and using autodifferentiation to compute $p_i''(x_i)$ in the case of ϕ_P.

As a final step, normalization was carried out on the dataset to improve model performance. Both the right hand side and the solution are divided by $\max(|\mathrm{RHS}|)$ to set the RHS' maximum absolute value to 1. Furthermore, the solutions were divided by an additional $\max\left(\left[L_x, L_y\right]\right)^2$ factor, that is square of the longest domain extent, to ensure that both the RHS and the solution are properly nondimensionalized[4].

During the dataset synthesis process, the number of terms for each sample in the Fourier and Taylor series components were chosen randomly from a uniform distribution within a predetermined range. Overall, this methodology to generate synthetic homogeneous Poisson problems provides four parameters per spatial dimension to control the roughness of the RHS—two parameters for both series components prescribing the range to draw the number of terms from—for a sum of eight parameters for the two dimensional problems investigated in Section 8. The grid parameters were similarly chosen randomly from uniform distributions within predetermined ranges. This necessitates two parameters to choose the range of grid spacings, plus a further two parameters per spatial dimension for the maximum and minimum number of grid points across each dimension, for a total of six parameters for a 2D problem. Note that grid spacings were randomly chosen per-batch instead of per-sample.

[4] The division by $\max(|\mathrm{RHS}|)$ nondimensionalizes the RHS but since the Laplace operator modifies the dimensionality of its argument by a factor of $[L]^{-2}$, this additional normalization is necessary to ensure that the solution is also nondimensionalized. This normalization was observed to reduce loss by over 50%.

5.2. Training data generation for the DBCNN

A different approach to generating training data for the DBCNN submodel was chosen, eschewing the series based approach for the HPNN submodel in favor of a purely numerical approach. This was motivated by the fact that the peak magnitude of the sinh component in the series for the solution of the Laplace equation shown in Equation (7) grows exponentially as the series index m grows. To avoid having to artificially skew the distribution of the coefficients, instead it was chosen to randomly generate the boundary conditions which necessitates the usage of a numerical method to obtain the solutions to the Laplace problems.

However, since the boundary conditions of interest are continuous, generating, for example, 200 random values for a 200 gridpoint boundary results in extremely noisy BCs that do not reflect the typical use case for Poisson solvers and are very difficult for NNs to learn from. To rectify this issue, it was instead chosen to generate lower resolution random BCs and then upscale these to the desired resolution using cubic interpolation. Owing to the C^2 continuity of cubic upscaling, this ensures that the resulting BCs are smooth when the ratio of the final resolution to initial resolution is high enough. The drawback of this method is the addition of this lower resolution to the list of user-defined parameters. To prevent the NN from overfitting for a specific value of this parameter, this value was chosen randomly from a uniform distribution for each batch. This methodology requires $2(N_D - 1)$ user-defined parameters to choose the minimum and maximum possible quantities of control points, where N_D is the number of spatial dimensions. The number of parameters needed to control the grid size is identical to Section 5.1.

Solutions to the resulting linear systems were computed using the Python algebraic multigrid package `PyAMG` by Olson and Schroder (2018). Specifically, the "classical" Ruge–Stuben algebraic multigrid method with a tolerance of 10^{-10} was chosen. Calculations were done on a 32-core 64-thread AMD Threadripper 2990WX CPU, using 64 threads. Table 6 in Section 8.7 outlines the wall-clock runtime needed to solve single problems at various grid sizes, excluding the time needed to generate the RHS and BCs, along with a comparison to the multigrid method running on the GPU based on the `pyamgx` Python bindings by Srinath (2018) to the `AMGX` library by Nvidia Corporation (2020). In this work, the coarse grid sizes (across each dimension) were set between 2 and 10. To avoid overfitting on specific examples, the datasets are generated on-the-go before each batch is fed to the training routine.

6. Loss Function and Approximation of Loss Value

In typical regression applications, it is common to use the mean squared error (MSE) as the loss function. However, as shown in multiple studies on image-to-image translation using convolutional models such as those by Isola et al. (2016), MSE loss does not typically lead to good results. As an alternative, since the targets are strictly smooth functions, the continuous version of the L^p norm between the output and the target presents a more meaningful loss for a pair of continuous functions. The norm is defined as

$$\left[\frac{1}{A}\int_A (y-t)^p \,\mathrm{d}A\right]^{1/p}, \tag{16}$$

where y is the prediction, t is the target, and A is a finite region of $\mathbb{R}^n$. However, we know the functions' values only on a rectangular grid. In a naive approach, we could evaluate the expression in Equation (16) by using a polynomial reconstruction method and integrating the piecewise polynomial. In fact, MSE loss can be interpreted as doing this with the midpoint rule. A more accurate alternative is Gauss–Legendre quadrature. In practice, this methodology can be written down for a function $h : \mathbb{R}^n \to \mathbb{R}$ as

$$\int_V h(x_1, x_2, \ldots, x_n)\,\mathrm{d}x_1 \ldots \mathrm{d}x_n \approx \sum_{i_1=1}^{k_1} \ldots \sum_{i_n=1}^{k_n} \left(\prod_{j=1}^{n} w_{i_j}\right) h\left(\overline{x}_{1_{i_1}}, \ldots, \overline{x}_{n_{i_n}}\right), \tag{17}$$

where k_l is the order of the quadrature in the lth direction, $\overline{x}_{l_{i_l}}$ is the i_lth quadrature point (i.e., Legendre polynomial root) in the lth direction and w is a vector containing the quadrature weights (hence w_{i_j}

denotes the i_lth weight in the jth direction). Applying this formula to Equation (16) for $y, t : \mathbb{R}^2 \to \mathbb{R}$ with k weights in both directions, we can write

$$\left[\frac{1}{A}\int_A \left(y(x_1,x_2)-t\left(x_1,x_2\right)\right)^p \mathrm{d}A\right]^{1/p} \approx \left[\frac{1}{A}\sum_i^k \sum_j^k w_i w_j \left[y\left(\bar{x}_{1_i},\bar{x}_{2_j}\right)-t\left(\bar{x}_{1_i},\bar{x}_{2_j}\right)\right]^p\right]^{1/p}. \tag{18}$$

This approach, however, relies on knowing the values of y and t at the quadrature points $\bar{x}_1, \bar{x}_2$ which are not equispaced and therefore cannot be made to coincide with the grid points using a simple affine transformation as is the common practice for utilizing Gauss–Legendre quadrature for domains other than the canonical $[-1,1]$ domain. To overcome this issue, $y(\bar{x}_1,\bar{x}_2)$ and $t(\bar{x}_1,\bar{x}_2)$ were approximated using bilinear interpolation within the "cell" enclosing each quadrature point. In practice, for each quadrature point, the value is a linear combination of the four known values on the corners

$$y(\bar{x}_1,\bar{x}_2) \approx [b_{00}\ b_{01}\ b_{10}\ b_{11}] \begin{bmatrix} y_{i,j} \\ y_{i+1,j} \\ y_{i,j+1} \\ y_{i+1,j+1} \end{bmatrix} = \vec{b}^{\top}\vec{y}, \tag{19}$$

where the weight vector $\vec{b}$ depends on the coordinates of the quadrature point and the grid points. Denoting $\vec{\cdot}_{ij}$ as the vector containing the values of the variable in question for the ith quadrature point in the x_1 direction and the jth quadrature point in the x_2 direction, we can rewrite Equation (18) as

$$\left[\frac{1}{A}\int_A \left(y(x_1,x_2)-t\left(x_1,x_2\right)\right)^p \mathrm{d}A\right]^{1/p} \approx \left[\frac{1}{A}\sum_i^k \sum_j^k w_i w_j \left[\vec{b}_{ij}^{\top}\left(\vec{y}_{ij}-\vec{t}_{ij}\right)\right]^p\right]^{1/p}. \tag{20}$$

During training, it was found that augmenting the integration with a mean absolute error (MAE) loss component sped up the loss minimization. Hence, the final expression for the loss is

$$L = \lambda_1 \left[\frac{1}{A}\sum_i^k \sum_j^k w_i w_j \left[\vec{b}_{ij}^{\top}\left(\vec{y}_{ij}-\vec{t}_{ij}\right)\right]^p\right]^{1/p} + \frac{\lambda_2}{N}\sum_i^N |y_i - t_i|. \tag{21}$$

Experimentation during training indicated that values of $\lambda_1 = 0.4$, $\lambda_2 = 1.0$ and $p = 2$ provide good performance. A benchmark of training the DBCNN model with and without this novel loss function can be found in Section 8.6.

Training the model by using the root mean square (RMS) solution residual as the loss as in works by Raissi et al. (2017a, 2017b) was also tried, both on its own and as an extra term in the loss expression in Equation (21). It was found that, when used by itself, the residual loss leads to unstable training. This issue can be overcome by starting training with an MAE and/or integral loss and introducing the residual component after several epochs of training. Unfortunately this leads to higher MSE than the loss function in Equation (21), albeit resulting in smoother solutions that may in some cases look qualitatively better. However, it was chosen to focus on numerical measures of error in this work.

7. NN Training

The model was implemented using Tensorflow 2.3. The submodels were trained independently; no further end-to-end training was carried out for the full model configuration shown in Figure 1. Adam was chosen as the optimizer method, with an initial learning rate of 10^{-4} for the DBCNN and 10^{-5} for the HPNN. Learning rate was dynamically tapered to 10^{-7} as loss plateaued. To determine when to stop the training, early stopping based on the mean squared error with 20-epoch patience was used. Table 1 summarizes the information regarding the number of samples used to train each model.

Table 1. *Details of the number of samples used to train each submodel.*

Model	Batch size	Batches per epoch	Epochs	Samples
HPNN	50	200	62	620,000
DBCNN	50		49	370,000

Abbreviations: DBCNN, Dirichlet boundary conditions neural network; HPNN, homogeneous Poisson neural network.

Furthermore, a number of best practice guidelines for both training and tweaking the model architecture were established for ensuring the best model performance:

- Final layers should have small kernels to prevent artefacting near the domain edges since smaller kernels require less padding to retain the input shape, while initial layers should have large kernels to propagate information from each gridpoint to every other gridpoint[5] using fewer convolutional layers.
- Randomizing input domain shapes by independently picking the size of each dimension from a uniform distribution leads to a nonuniform distribution of aspect ratios (ARs), as the ratio of two uniform distributions is known to result in a highly nonuniform distribution per Marsaglia (1965). This leads to poor performance with ARs which are less commonly encountered in the dataset. To rectify this, instead the ARs themselves can be picked from a uniform distribution and the grid sizes generated based on the ARs.
- Using large batch sizes is critical for ensuring good model performance. In the case of the HPNN submodel, increasing the batch size from 13 to 50 led to a 45% reduction in loss with an identical number of samples and a 78% reduction at the end of training.
- Adding as many pooling levels to the HPNN as possible is crucial to allow the model to capture information at multiple scales.
- Deconvolutional upsampling in the HPNN provides superior results when reversing pooling operations with small pool sizes, while bilinear/nearest neighbor are better for very large pool sizes (e.g., 64×64).
- Increasing the number of spatial pyramid pooling (SPP) levels in the DBCNN was found to reduce prediction accuracy beyond a certain level; increasing the number of output values from the SPP layer to 123 from 58 reduced loss by 41%, but further increasing it to 228 led to a 32% increase relative to 123 outputs.
- Using Rectified Linear Unit (ReLU) activations in the intermediate layers for this task was observed to cause the dying ReLU problem. Other activations such as tanh or leaky ReLUs were observed to result in higher accuracy.
- Residual connections boost performance of both the DBCNN and the HPNN.

8. Results and Performance

In this section we show the performance of a Poisson CNN model, the submodels of which were trained independently on problems with grids containing 192–384 grid points in each direction and Δ values between 0.005 and 0.05. Sections 8.1 and 8.2 outline the test cases chosen and give greater detail regarding the performance of the model in each test case. Moving outside the "comfort zone" of the model, Section 8.3 investigates the performance of the model with problem sizes outside the training data. Section 8.4 details ways to work around artefacting that can show up in the model's predictions, and demonstrates the usage of Poisson CNN predictions as initial guesses for the multigrid algorithm. Building on that demonstration, Section 8.5 showcases the usage of the Poisson CNN within a CFD

[5] Propagating information from each gridpoint to every other gridpoint is necessary for the Poisson problem due to the elliptic nature of the PDE.

Table 2. *Summary of the number of parameters.*

	Homogeneous Poisson NN	Dirichlet BC NN	Poisson CNN
Number of parameters	5,559,108	483,878	6,042,986

Abbreviations: BC, boundary condition; CNN, convolutional neural network.

Table 3. *Summary of the results presented in this section, plus averaged figures for larger numbers of examples for each case where applicable.*

	Case	MAE	MAPE	% of gridpts within 10% of target	Normalized peak error
Examples similar to the training set (Section 8.1)	HPNN (Figure 4)	4.73×10^{-2}	13.25	56.24	1.43×10^{-1}
	HPNN (Figure 5)	3.24×10^{-2}	8.96	74.88	1.26×10^{-1}
	HPNN—Avg	3.50×10^{-2}	13.04	57.47	1.58×10^{-1}
	U-Net—Avg	6.98×10^{-1}	50.37	20.04	1.22×10^{0}
	DBCNN (Figure 6)	7.78×10^{-3}	18.29	41.16	8.01×10^{-1}
	DBCNN (Figure 7)	8.03×10^{-3}	20.66	30.73	7.56×10^{-1}
	DBCNN—Avg	1.96×10^{-2}	19.26	40.54	8.10×10^{-1}
	LSTM—Avg	1.06×10^{-1}	53.03	10.04	1.04×10^{0}
	Full model (Figure 8)	9.62×10^{-2}	9.81	66.44	8.32×10^{-1}
	Full model—Avg	6.39×10^{-2}	8.48	71.70	8.76×10^{-1}
Taylor–Green vortex (Section 8.2)	HPNN (Figure 9)	1.01×10^{-2}	12.91	63.48	6.91×10^{-2}
	DBCNN (Figure 10)	4.65×10^{-3}	15.80	43.28	3.86×10^{-2}
	Full model (Figure 11)	1.51×10^{-2}	12.08	60.98	1.31×10^{0}

Note that grid points with absolute percentage errors above 200% were excluded from the MAPE calculation due to MAPE values approaching infinity near the zero solution contours. Baseline cases are denoted in *italics.* Average figures were computed over 600 randomly generated samples. Peak error figures were normalized by the maximal absolute value of the ground truth.

simulation and compares the accuracy obtained with a fully conventional simulation. Finally, Section 8.6 explores the impact of key innovations in the model architecture on the predictive performance of the model and 8.7 compares the wall-clock runtime of the model against a conventional iterative solver algorithm. Furthermore, to demonstrate the flexibility of our approach, we present in Appendix A a gallery of results with a greater range of aspect ratios.

Table 2 outlines the number of parameters for the models with the specific choice of hyperparameters (e.g., kernel sizes and number of channels) made. It should be noted that our choice does not represent a necessarily optimal choice for these hyperparameters, as this work is meant as a feasibility study.

Table 3 gives an overview of the performance of the models in terms of the MAE, mean absolute percentage error (MAPE) and the percentage of grid points in each case for which the model made a prediction within 10% of the target. Supplementing the selected test cases, averaged results for larger sets of similar examples are also available where applicable.

In addition to our models, we provide an overview of the performance of two baseline models trained on the same datasets using identical loss functions and optimizers to illustrate the strength of our individual sub-models on their respective tasks. A four-level U-Net, proposed by Ronneberger et al. (2015), with leaky ReLU activations and 7.5 million parameters serves as the baseline case for the HPNN submodel. Meanwhile, the baseline case for the DBCNN sub-model is a stack of six bidirectional Long Short-Term Memory (LSTM) layers (see Hochreiter and Schmidhuber, 1997) with 100 units per layer and tanh activations containing approximately 440,000 parameters, plus bilinear upscaling applied to the final LSTM output to match the target shape.

Detailed results from the baseline experiments are omitted for brevity, however our models greatly outperformed the naive baseline implementations; both the HPNN and the DBCNN model achieved MAEs an order of magnitude smaller than their respective baselines. Particularly, the U-Net produced results with the kind of severe checkerboard artefacting patterns associated with transpose convolutions, in line with the observations of Odena et al. (2016), displaying the advantage of using both transpose convolutions and traditional upsampling techniques in the HPNN architecture. Meanwhile, compared to the LSTM stack, the DBCNN model's specifically tailored architecture leveraging the mathematical properties of the investigated problem produces results which exhibit substantially less high-frequency noise in the direction orthogonal to the boundary. Another striking result is the percentages of grid points for which the prediction is within 10% of the target value, which are much higher for our models when compared to the baselines.

8.1. Examples similar to the training set

It is crucial for machine learning (ML) algorithms to be able to make correct predictions on previously unseen data that comes from a dataset with the same conditional probability distribution between the inputs and outputs as the training data. In typical ML applications, this is tested for by splitting the dataset (of e.g., photos) into a training and a validation set and observing if a model that performs well on the training set does equally well on the validation set. In our approach, since each batch of training data is generated from random noise synthetically just before being fed to the training loop, analogously we can test the model's performance on new random samples generated in an identical manner. This section presents the models' performance on examples generated in the same manner as the training data first for each submodel individually in Sections 8.1.1 and 8.1.2, finally presenting an example for a Poisson problem with four inhomogeneous DBCs in Section 8.1.3.

8.1.1. Homogeneous Poisson NN

Figures 4 and 5 depict the performance of the HPNN on two random examples generated in the same manner as the training data, with different ARs. The model displays good predictive accuracy with the majority of the predictions lying within 10% of the target for the example in Figure 4, rising to over three quarters for the example in Figure 5. A substantial proportion of the predictions that lie outside this 10% band are clustered around the contours where the solution $\phi = 0$, where high percentage errors occur despite good predictions in terms of the numerical value. The most notable inaccuracies (in terms of the absolute values) lie near points where the ground truth has a local maximum or minimum due to slight mispredictions of both the location and the value of these local extrema.

The example in Figure 5, when juxtaposed against Figure 4, showcases the capability of the HPNN submodel to deal with problems involving vastly differing ARs and grid sizes without retraining. This capability is crucial for Poisson CNN's intended aim as an acceleration step for iterative algorithms, as reusability for different grid parameters greatly enhances the attractiveness for this purpose.

8.1.2. Dirichlet BC NN

As shown in Figures 6 and 7, the DBCNN reproduces the multigrid solution with a good degree of accuracy, replicating the hyperbolic sine solution profile well. The largest absolute errors occur due to some oscillatory behaviour near the left boundary, caused by the padding-related issues explained in Section 7 being magnified by the large solution magnitudes in this region. Section 8.4 discusses the application of Jacobi post-smoothing to resolve this issue. Jacobi smoothing substantially reduces the peak normalized error down to 2.49×10^{-1} and 1.96×10^{-1} for Figures 6 and 7, respectively—an almost three quarter reduction compared to the values in Table 3. The performance in terms of the percentage of grid points with predictions within 10% of the target are 41 and 31% for the two cases, slightly worse than the HPNN submodel. Conversely, the MAE for the case in Figure 6 is 84% lower than the MAE for the HPNN submodel in Figure 4 as shown in Table 3.

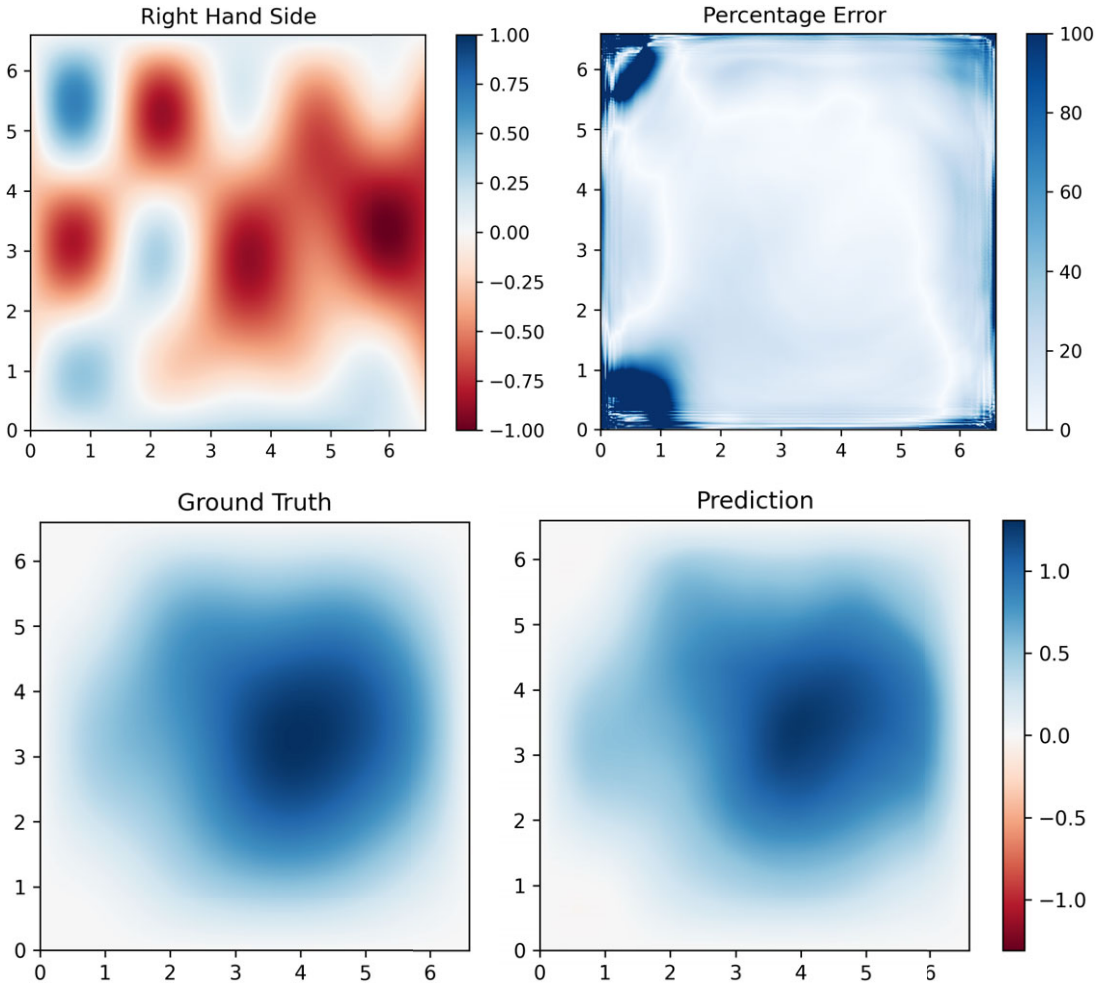

Figure 4. *Performance of the homogeneous Poisson neural network (HPNN) model on an example with grid size* 365×365 *and* $\Delta = 1.81 \times 10^{-2}$. *The HPNN submodel performs well, producing a prediction within 10% of the target for over half of the grid points.*

The seemingly contradictory MAPE and MAE values are explained by the observation that % errors increase progressively as we move in the positive horizontal direction. For the DBCNN, % errors rise toward the right boundary since the sinh-shaped solution profile in this direction rapidly reduces the solution magnitudes. Thus, absolute errors remain very small despite large relative errors. ARs further complicates this comparison as the solution decays quicker in terms of normalized horizontal coordinates x/L_x for high AR domains as seen in Figure 7. Hence, MAPEs are relatively small and MAEs are relatively large for low AR cases and the opposite is true for higher ARs.

8.1.3. Poisson CNN (full model)

Figure 8 depicts the performance of the full model on a randomly generated Poisson problem. The performance of the model in terms of MAPE is 9.81% and over two thirds of grid points have predictions within 10% of the target. The decomposition of the Poisson problem as shown in Equation (5) performs well and the Poisson CNN architecture proposed is capable of providing good estimates for solutions to Poisson problems with four inhomogeneous BCs, substantially exceeding the performance of the individual submodels on their respective subproblems.

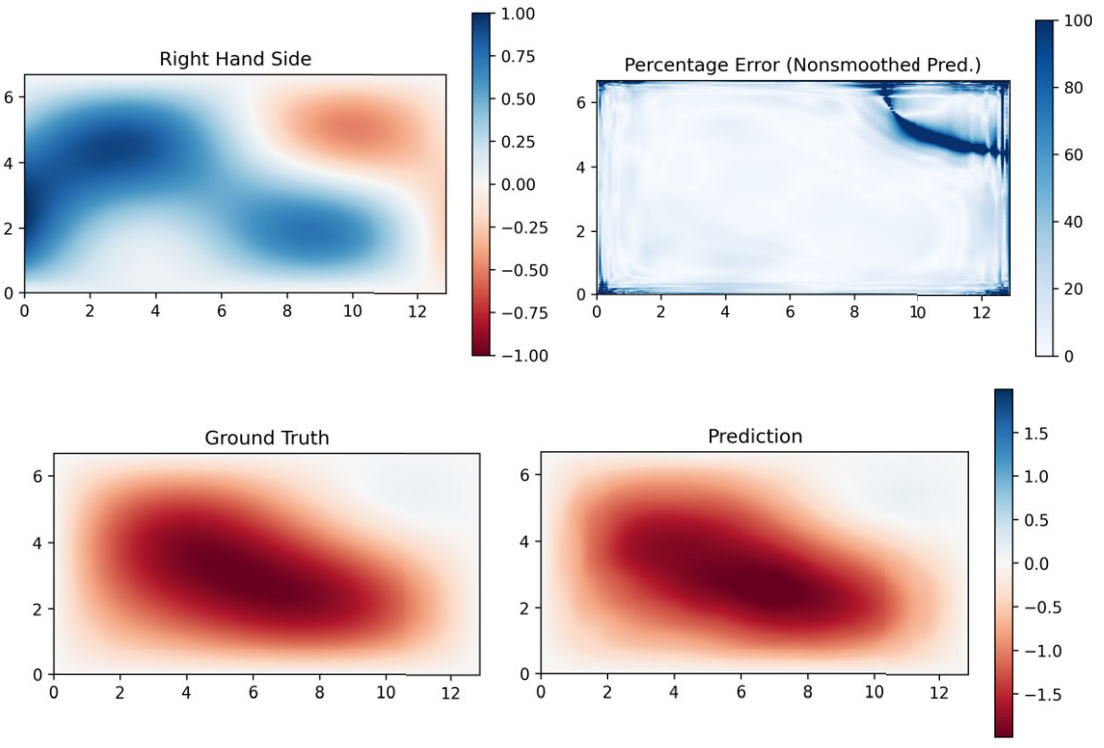

Figure 5. *Performance of the homogeneous Poisson neural network (HPNN) model on an example with grid size* 384×192 *and* $\Delta = 3.47 \times 10^{-2}$. *The HPNN submodel can handle a variety of different aspect ratios well, including in this case where it predicts values within 10% of the target for almost three-quarters of the grid points.*

8.2. Taylor–Green vortex problem

The Taylor–Green vortex (TGV) in two-dimensions is a well-known analytical solution to the Navier–Stokes equations. The pressure component of the TGV, with the time-dependent term and density set as unity for simplicity, presents an easy-to-construct analytical test case. It is a benchmark frequently used in CFD community and is representative of Poisson equations encountered in practical applications. We can construct a Poisson problem in the domain $[0,\pi] \times [0,\pi]$ by setting the solution ϕ as the TGV pressure field

$$\phi = -\frac{1}{4}(\cos(2x) + \cos(2y)) \tag{22}$$

$$\therefore \mathrm{RHS} = \nabla^2\phi = \cos(2x) + \cos(2y), \tag{23}$$

with the BC along each boundary defined as

$$b(t) = -(\cos(2t) + 1)/4, \tag{24}$$

where t is the coordinate along the boundary. As a benchmark case, we investigate the performance of the model on this problem with a grid size towards the middle of the range the model has seen during training—255×255.

Figures 9, 10, and 11 show the performance of the HPNN, DBCNN, and Poisson CNN models, respectively. The HPNN submodel performed at a level in line with the results displayed in Section 8.1.1 when predicting on the RHS function. The model achieved predictions within 10% of the target at over half of the grid points and reproduced key solution features such as symmetricity about

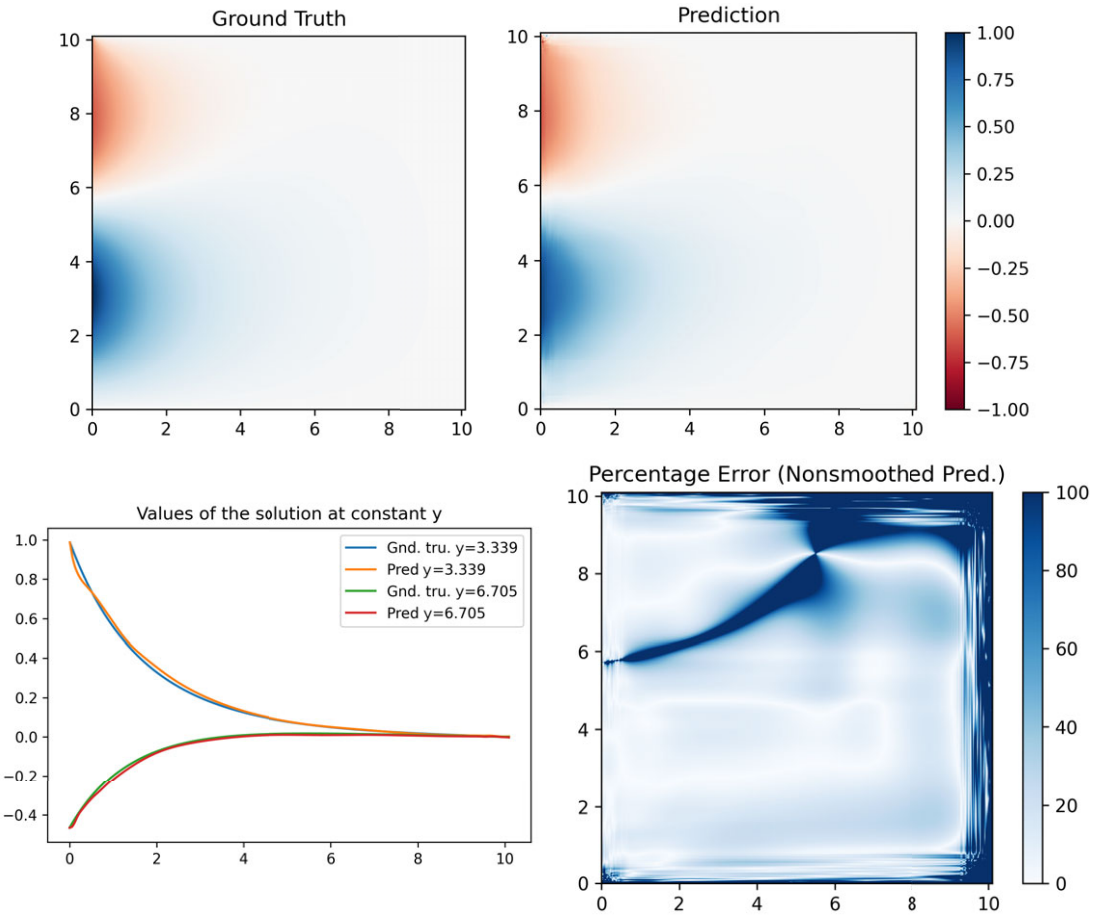

Figure 6. *Performance of the Dirichlet boundary condition neural network (DBCNN) model on an example generated in the same manner as its training data, with a grid size of* 384×384 *and* $\Delta = 2.63 \times 10^{-2}$. *The DBCNN submodel replicates the hyperbolic sine solution profile successfully.*

the $x = y$ line. Mirroring the previous cases, largest absolute errors are concentrated near local extrema, particularly near the local minimum in the middle of the domain, and largest percentage errors are near the $\phi = 0$ contours.

The DBCNN submodel performed slightly better in this case compared to the 600-sample average shown in Table 3. Although providing somewhat grainy results and underpredicting the peak magnitude of the solution along the $y = (2k+1)\pi/2$ contours where the BC value approaches 0, the predictions along the midsection of the domain are very good. Highest absolute errors are seen near the left boundary by the corners while the highest percentage errors are by the right boundary, mirroring the previous sample shown in Section 8.1.2.

The performance of the full Poisson CNN model exceeds that of the individual submodels. The MAPE of the full model lies below that of the two submodels and is just above the previous 600-sample average shown in Table 3. Important solution features such as the sharp contours along the $x+y = (2k+1)\pi/2$ lines are present. However, from a purely qualitative perspective, some artefacting is visible near the corners, stemming from the artefacting seen in the DBCNN prediction.

Overall, both submodels as well as the overall Poisson CNN model demonstrate solid performance in this analytical test case, giving further evidence that the model did not overfit on the training data.

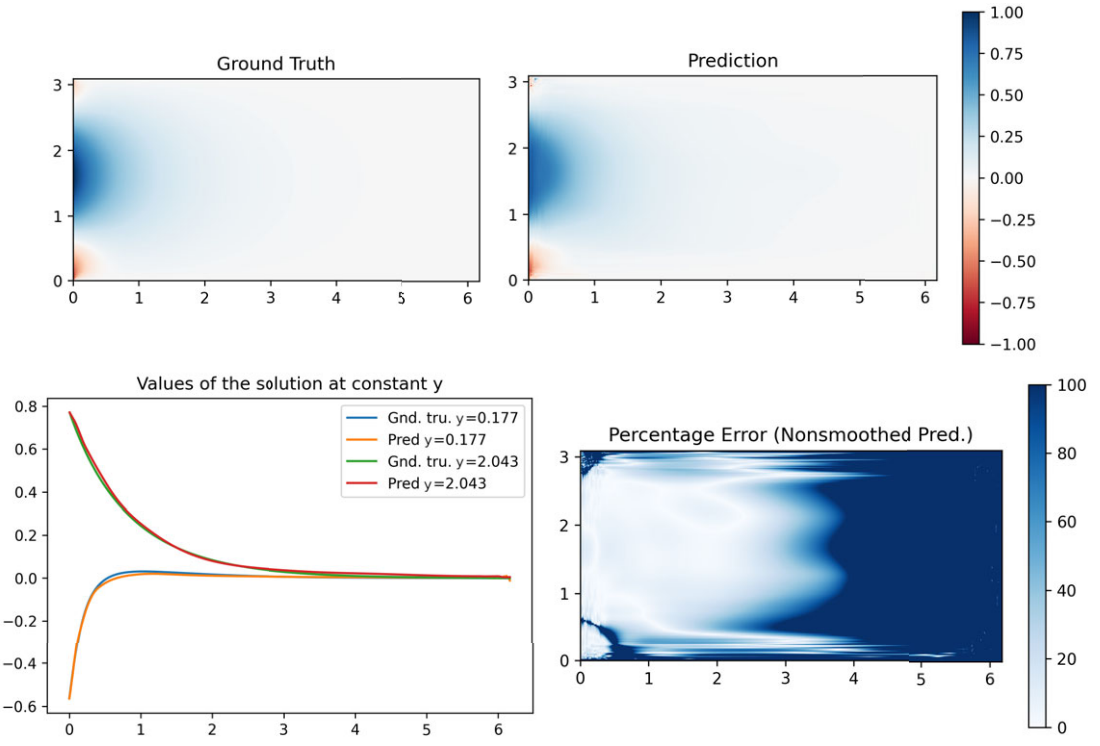

Figure 7. *Performance of the Dirichlet boundary condition neural network (DBCNN) model on an example generated in the same manner as its training data, with a grid size of* 382×197 *and* $\Delta = 1.59 \times 10^{-2}$. *Similar to the homogeneous Poisson neural network (HPNN), the DBCNN can handle different aspect ratios effectively, replicating the solution profile properly.*

8.3. Performance with previously unseen grids

An important aspect of the generalization performance of the model is whether it is able to handle grids with parameters (such as grid spacing and sizes) outside the range encountered during training. Figure 12 shows the RMS error of the model prediction relative to the analytical solution on the same TGV case shown in Section 8.2 with progressively denser grids.

The results clearly indicate that while the model is struggling to handle grids that are coarser than the training data, the quality of the predictions degrade more gradually for larger grids. In addition, even within the range seen in the training data, the NN model behaves unlike a traditional numerical algorithm where a linear decrease of the RMS error with a slope equal to the order of the method is expected.

In the 120×120 case, the model gives an answer completely dissimilar to the analytical solution due to severe mispredictions by the HPNN submodel as seen in the midsection of the domain. For the 500×500 and 750×750 cases however, the general solution profile is retained despite gradually increasing under-prediction of the local maximum at $(\pi/2, \pi/2)$ and over-prediction of the local minima at the corners.

Although the model's performance when predicting on smaller grids than the training data is substantially worse, this is a use case with far narrower use cases than the converse. The fact that the model retains the ability to reproduce the general solution profile, albeit with lower accuracy, is very promising as this can enable the model to supply initial guesses for iterative algorithms for problems with larger grids than encountered during training. The capability of the model to do that for the multigrid algorithm will be explored in Section 8.4.

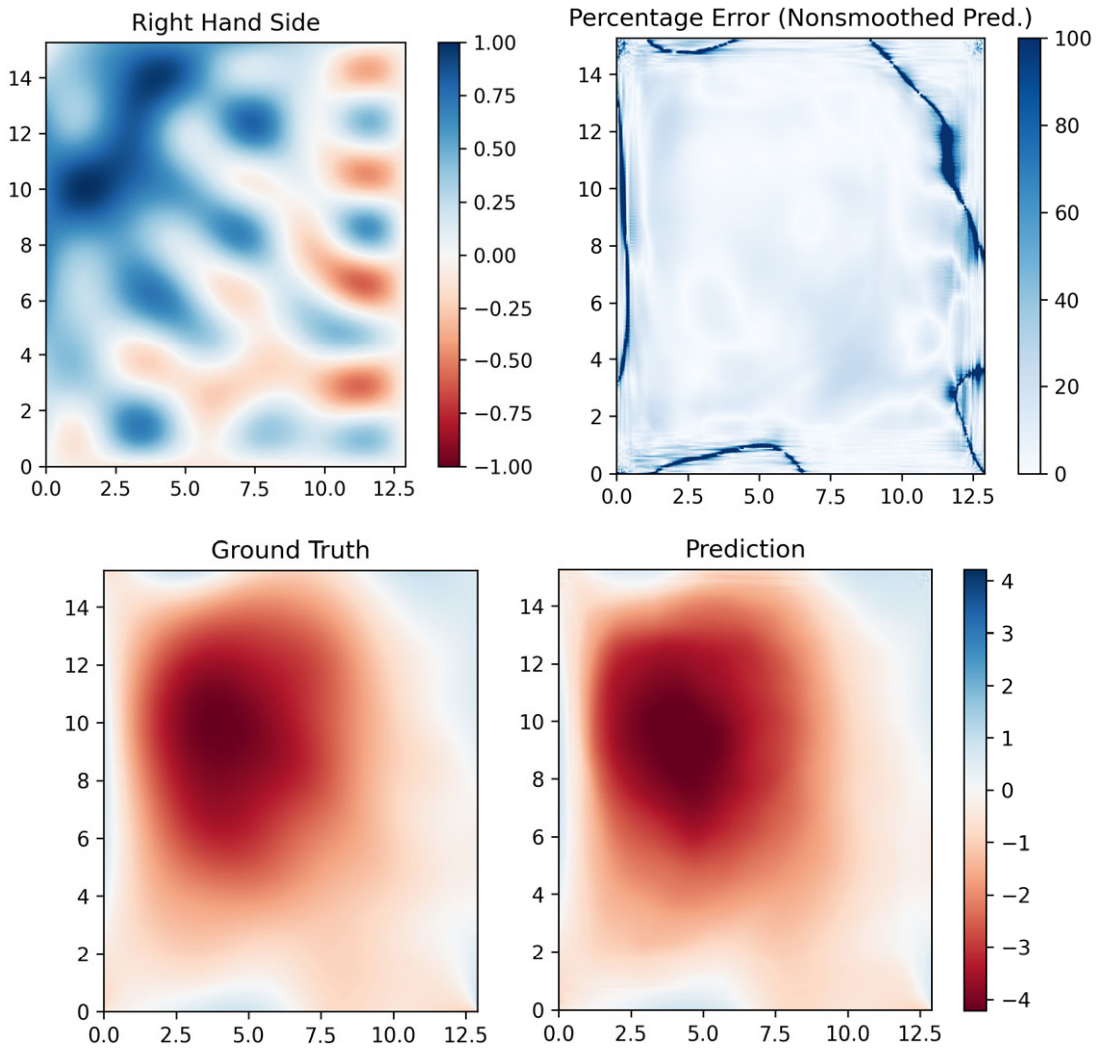

Figure 8. *Performance of the Poisson convolutional neural network (CNN) model on an example with grid size* 263×311 *and* $\Delta = 4.91 \times 10^{-2}$. *The components predicted by the submodels reconstruct the solution accurately, demonstrating the flexibility of the decomposition.*

8.4. Post-smoothing and comparison versus one multigrid cycle

Considering the primary envisioned use case for our model is accelerating iterative Poisson solvers, one problematic issue is the presence of high-frequency artefacting which can occasionally manifest itself as seen in Figure 11. Applying five Jacobi postsmoothing iterations to the predicted field eliminates most of the high frequency artefacting, greatly reducing the roughness of the produced solution surface as shown in Figure 14.

The post-smoothing applied modestly improved the MAPE of the solution, lowering it to 11.61%. With the same post-smoothing applied, the multigrid method with eight levels was able to achieve a MAPE of 21.08%. Hence, the post-smoothing resolves the issue of high-frequency oscillations created by the model near the edges as can be seen by comparing Figures 11 and 14.

The preliminary results in Figure 14 demonstrate that it should be possible to increase the convergence rate of conventional iterative methods by using the present CNN architecture as the first step of an iterative strategy. Figure 15 serves as a more detailed demonstration of that capability, juxtaposing the RMS error

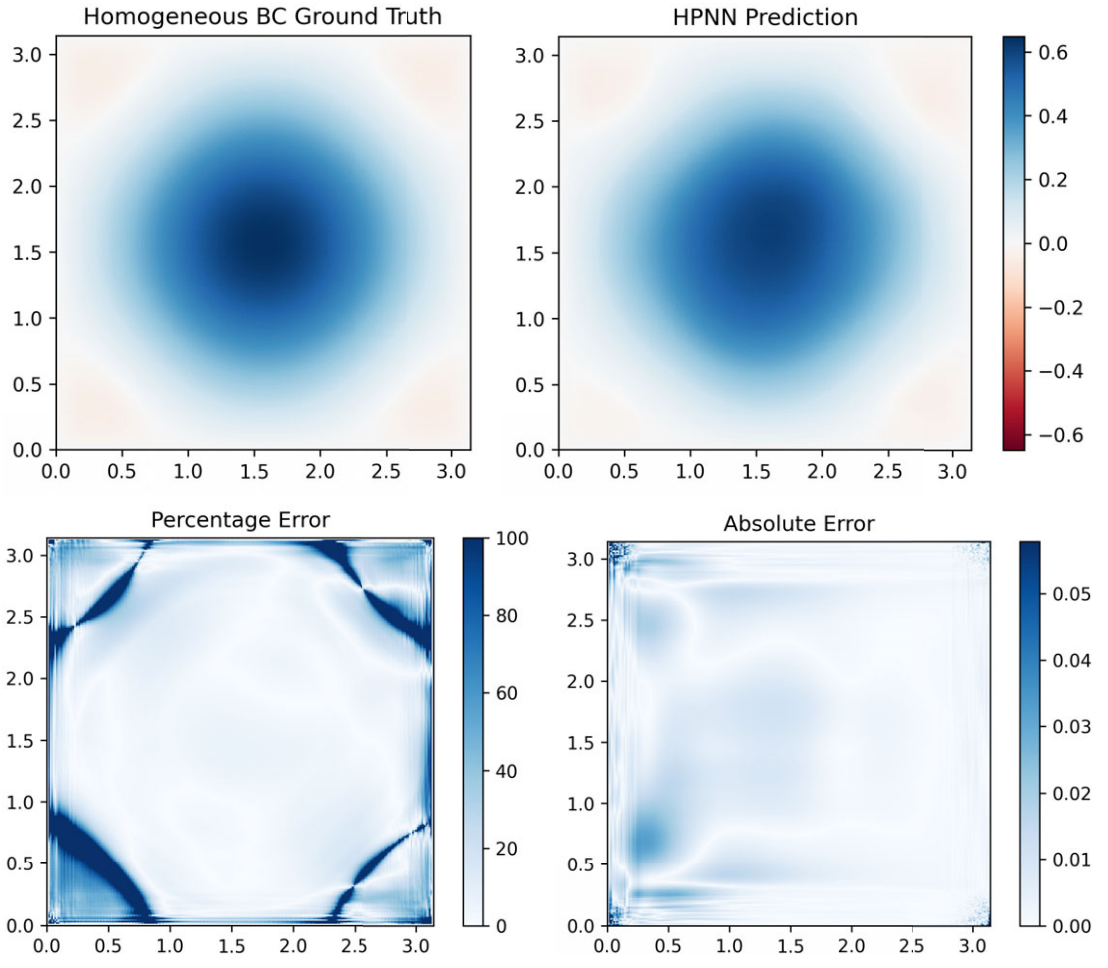

Figure 9. *Prediction of the homogeneous Poisson neural network (HPNN) submodel on the Taylor–Green vortex (TGV) case, compared to multigrid. Although the problem is materially dissimilar to the training set, the HPNN submodel performs in line with the 600 example average in Table 3.*

after a single multigrid iteration with a zero initial guess, two multigrid iterations with a zero initial guess and a single multigrid iteration with the model prediction as the initial guess for the TGV case over a number of grid resolutions, including those outside the training data range.

The model shows remarkable capability in this task, being able to reduce the RMS error after a single multigrid iteration by a massive 94% relative to an initial zero guess when utilized on a problem with a previously encountered 384×384 grid size. A reduction of 74% is achieved on a $4{,}500 \times 4{,}500$ problem, a grid over an order of magnitude larger than the largest problem in the training set. It is further noteworthy that a single multigrid iteration with the model prediction as the initial guess is able to beat two multigrid iterations in accuracy even for the same $4{,}500 \times 4{,}500$ grid with 22% lower RMS error. This suggests that the Poisson CNN model can provide very substantial increases of accuracy to the multigrid algorithm. Moreover, the model is capable of undertaking this task for problems much larger than the examples encountered during training.

This capability is especially important tn light of the results in Table 6 which clearly display that our model increases the runtime gap versus multigrid as the problem size grows. Eventually, our model catches up to even a single cycle of multigrid for large problems while beating the accuracy of the said initial cycle, providing a substantial boost to the convergence rate of the multigrid method from the time perspective as well.

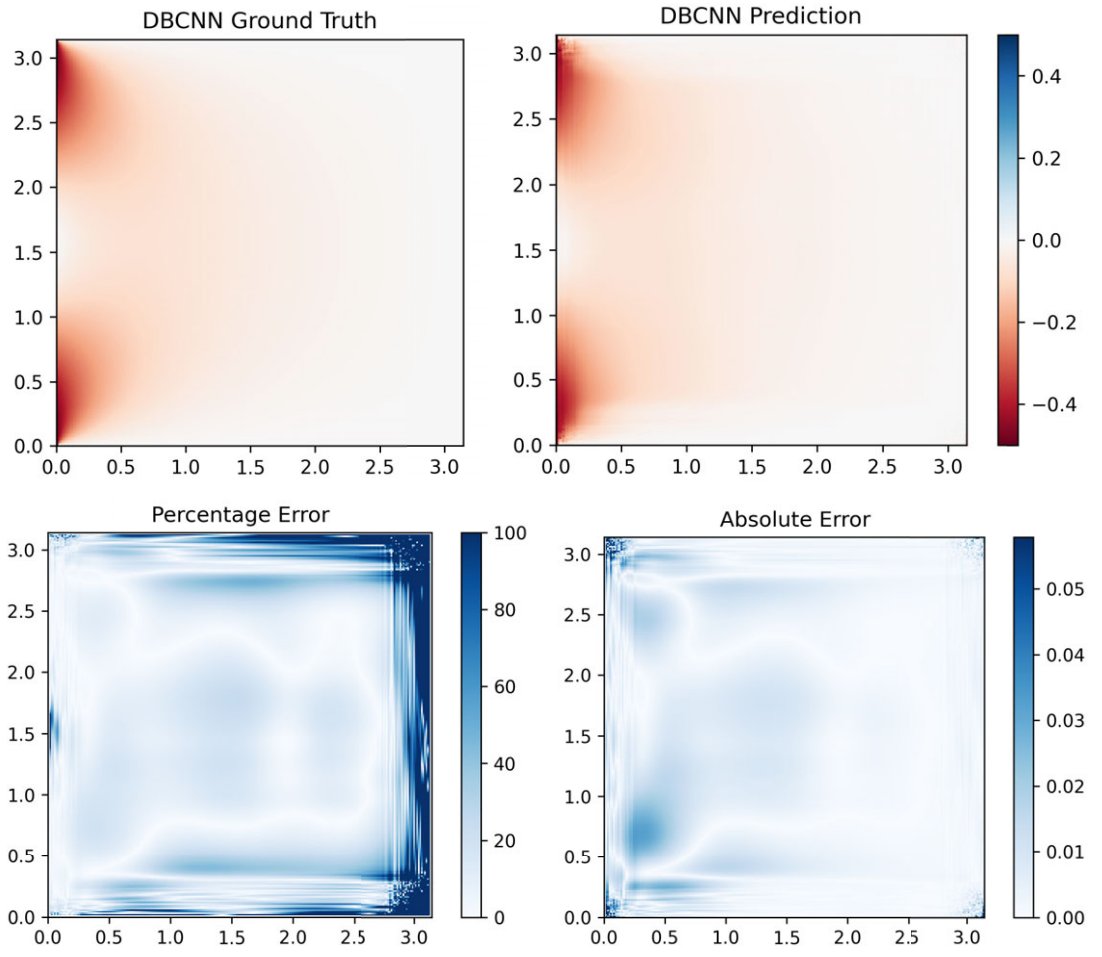

Figure 10. *Prediction of the Dirichlet boundary condition neural network (DBCNN) submodel on the Taylor–Green vortex (TGV) case, compared to multigrid. The DBCNN submodel performs well in this test case, marginally better than the 600 example average in Table 3, although mild artefacting is visible near the corners.*

8.5. NN-assisted pressure projection method

Given the solid performance of the model on a single snapshot as shown in Sections 8.2 and 8.4, we present the results from a TGV simulation, where the pressure projection step is assisted by a variant of the proposed NN. To ensure boundary condition compatibility with this CFD problem, we require a variant of the model capable of handling Neumann as opposed to DBCs. Considering the analytical solution for the TGV presented in Equation (23), sticking to the domain previously investigated in this section, we only require a model capable of handling homogeneous Neumann BCs.

Hence, the variant of our model used in this sub-section comprises of only the HPNN component, with the only change being the application of "symmetric" padding in the final convolution layer as opposed to zero padding to ensure that the boundary conditions are naturally applied by the model. The dataset used to train this model also incorporates a corresponding modification to accommodate the change in the BCs, switching to homogeneous Neumann BCs as opposed to DBCs. In effect, this is achieved by replacing the sine series in Section 5.1 with a cosine series. The training on this dataset was done in a manner similar to the model with DBCs in Sections 8.1–8.4, but with smaller grid sizes ranging between 80 and 120 on each side. In addition, the physics-informed loss from Raissi et al. (2017a) was found to reduce validation MSE

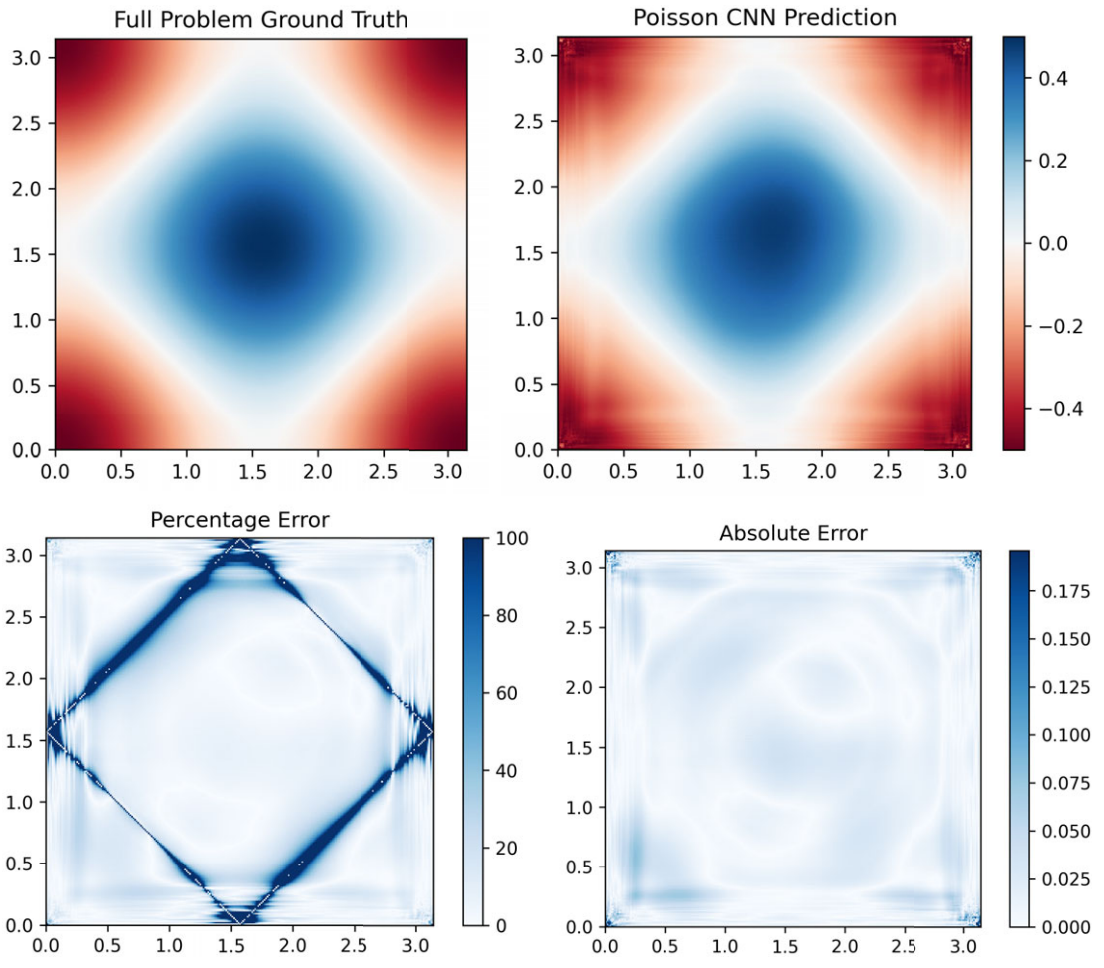

Figure 11. *Prediction of the full model on the Taylor–Green vortex (TGV) case. Overall, the Poisson convolutional neural network (CNN) exhibits good performance only slightly behind the performance shown on the problems similar to the ones encountered in the dataset, highlighting the generalization performance of the model.*

with Neumann BCs, unlike the DBC case, and thus was used as a loss function component in addition to Equation (21).

The simulation itself was performed within the framework of the MIT licensed `Navier_Stokes_2D` codebase by Roberts and Zhang (2014), a simple 2D finite-difference code incorporating a range of projection methods, including an implementation of the gauge method proposed by Weinan and Liu (2003) which was utilized in this investigation. The codebase was modified to ensure inter-operability with the HPNN model and recent versions of Python3.

The nondimensionalised Navier–Stokes equations require a choice of Reynolds number[6] $Re = 1/\nu$, where ν is the kinematic viscosity of the fluid. Additionally, an appropriate Courant–Friedrichs–Lewy number $C = \frac{\Delta t}{\Delta}(u+v)$ must be chosen to ensure that the time step Δt is sufficiently small to prevent numerical instabilities given the grid spacing Δ and the $x-$ and $y-$direction velocities u and v. The results presented in this section were obtained from a simulation run with the default parameters of $Re = 1.0$ and

[6] The Reynolds number is a nondimensional quantity which is commonly interpreted as the ratio of inertial to viscous forces in a fluid, and determines the amount of diffusion in the nondimensionalized equation system.

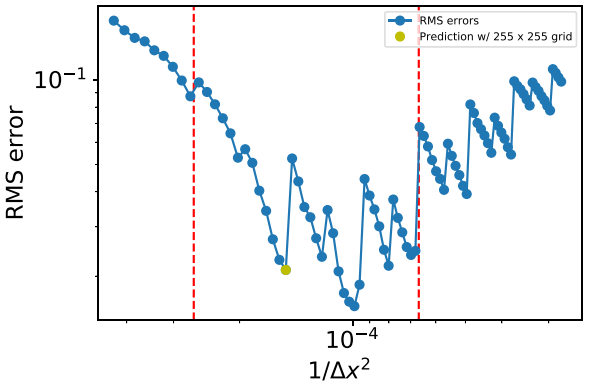

Figure 12. *Root mean square (RMS) error model output w.r.t. the analytical solution versus grid density for the Taylor–Green vortex (TGV) case. Interval of grid sizes encountered in training is marked by red lines. Yellow dot indicates the results presented in Section 8.2.*

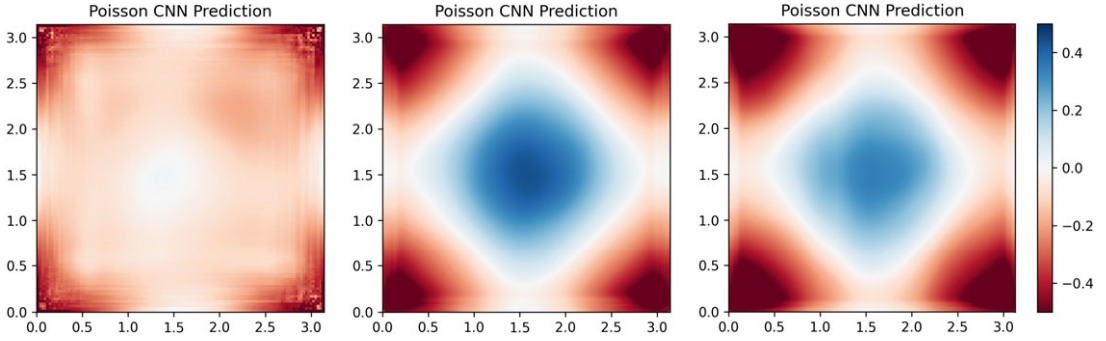

Figure 13. *Poisson convolutional neural network (CNN) predictions on* 120×120 *(left),* 500×500 *(middle) and* 750×750 *(right) grids for the Taylor–Green vortex (TGV) case. Although the model does not perform well for grids smaller than those seen during training, its predictive ability diminishes only gradually for larger grids.*

$C = 0.2$ in the `Navier_Stokes_2D` codebase. The domain was kept identical to the one in Section 8.2, but was discretised with 100×100 grid points. Similar to the "hybrid" strategy applied by Ajuria Illarramendi et al. (2020), designed to reduce the accumulation of errors during the time marching process, traditional solver iterations are applied to the output of the model.

Table 4 displays the L_2 error norms of the velocities and the pressure compared to the analytical result at the end of the simulation ($t = 1.0$). Figure 16 juxtaposes the ground truth result with the predictions of the HPNN for the pressure at $t = 1.0$, both raw and when assisted by a single iteration of the biconjugate gradient stabilized (BiCGSTAB) method.

Overall, the HPNN aided by a single BiCGSTAB prediction achieves a level of pressure error comparable to the level achieved by two iterations of BiCGSTAB with an initial zero guess, a massive L_2 error reduction compared to a single zero-initial-guess BiCGSTAB iteration. This is in line with our comparison with multigrid solvers in Section 8.4 using the Dirichlet BC variant of the Poisson CNN model, showing strong evidence that our model performs well in practical applications for acceleration of Poisson problems.

8.6. Ablation studies

The proposed NN architecture incorporates many features which improve performance for the Poisson equation task relative to other CNN architectures commonly used in image-to-image translation tasks.

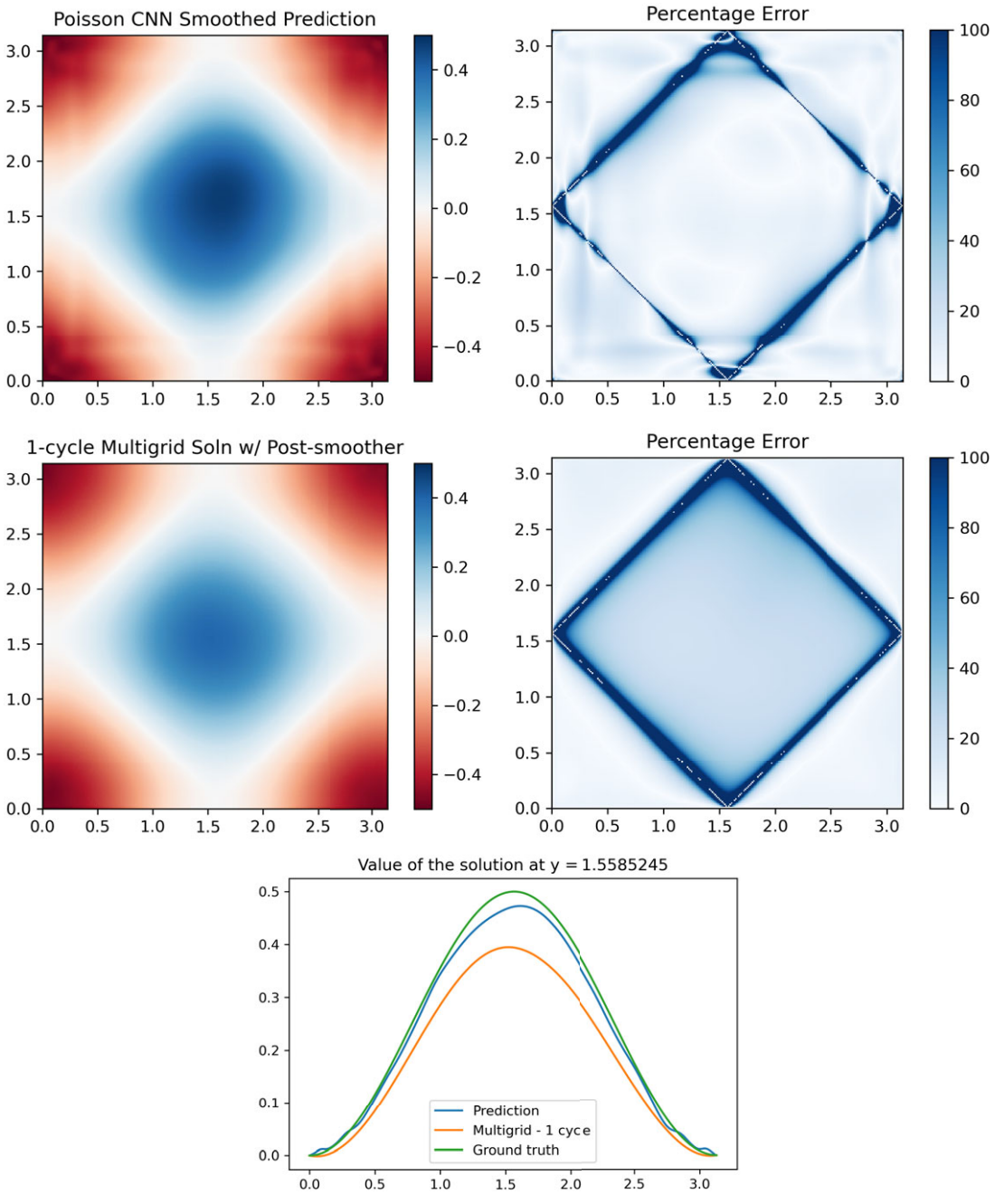

Figure 14. *Comparison of the model's performance versus multigrid with a single cycle, with five Jacobi post-smoothing iterations applied to each. The Poisson convolutional neural network (CNN) prediction clearly outperforms the single-cycle multigrid prediction.*

This is clearly demonstrated by the order-of-magnitude difference in validation MAE achieved on various cases relative to the baseline models in Table 3. To open a path for future advances in developing CNN architectures for this task, it is important to understand the contribution of each architectural feature to the performance of the model. One of the most common methods to investigate this in machine learning is via

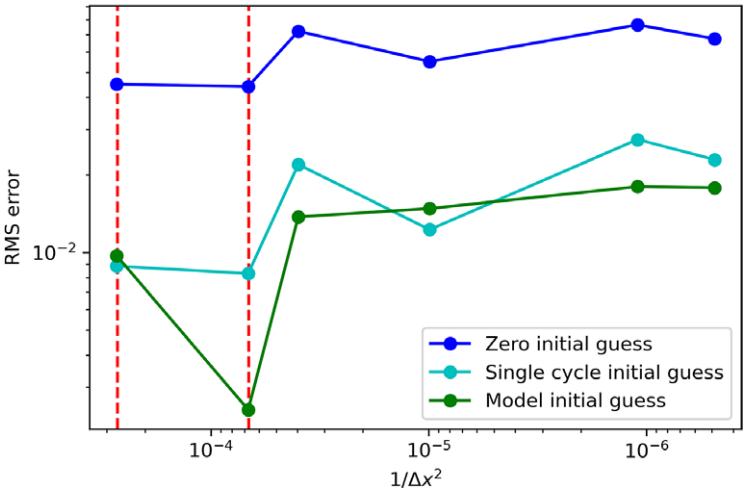

Figure 15. *Root mean square (RMS) error comparison for multigrid iterations with zero, single cycle multigrid and Poisson convolutional neural network (CNN) prediction initial guesses on the Taylor–Green vortex (TGV) snapshot case. Red lines indicate the grid parameter range seen by the model during training.*

Table 4. *L_2 error norms for the velocities (u,v) and the pressure p at the end of the Navier–Stokes simulation ($t = 1.0$).*

Case	$\|\text{err}(u)\|_2$	$\|\text{err}(v)\|_2$	$\|\text{err}(p)\|_2$	% reduction in $\|\text{err}(p)\|_2$
Zero initial pred. + 1 iter.	3.66×10^{-5}	3.53×10^{-5}	7.03×10^{-3}	99.3%
HPNN initial pred. + 1 iter.	4.69×10^{-6}	4.73×10^{-6}	4.80×10^{-5}	
Zero initial pred. + 2 iter.	3.93×10^{-7}	3.69×10^{-7}	1.56×10^{-5}	52.2%
HPNN initial pred. + 2 iter.	2.59×10^{-7}	2.52×10^{-7}	7.45×10^{-6}	

an ablation study, whereby certain features of the model under investigation are disabled in isolation and the performance is compared against the full model. Table 5 presents the results of our ablation study conducted with five key improvements of our model compared to previous works by considering the rise in validation MSE as well as change in the inference speed of the model.

The ablation study clearly highlights the magnitude of the improvements Poisson CNN offers compared to previous architectures. Most significant advantage of the proposed model is the utilization of the decomposition approach presented in Section 3, which enables it to achieve a more than six-fold reduction in validation MSE at zero cost in terms of inference time. Next in terms of importance comes the almost halving of validation MSE enabled by the inclusion of positional embeddings, a feature critical to enable good performance when handling variable input shapes to the model. Then, we see the approximately one-third reduction in validation MSE afforded by employing our novel loss function, the L^p integral loss, and the addition of more pooling blocks compared to the Fluidnet architecture by Tompson et al. (2016). Finally, the inclusion of residual connections in convolutional layers enables a more modest further 5% reduction in MSE values. Combined, these features enable our model to tackle the generalized form of the Poisson problem effectively, enabling a marked improvement over models previously employed on the Poisson problem or in more general image-to-image translation tasks.

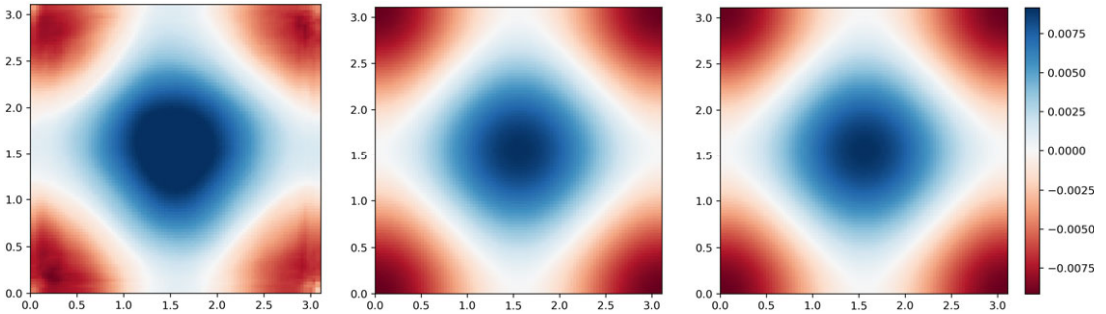

Figure 16. *Raw Neumann boundary condition (BC) homogeneous Poisson neural network (HPNN) prediction (left), Poisson convolutional neural network (CNN) prediction with one traditional solver iteration (middle) and ground truth solution (right) of the Taylor–Green vortex (TGV) simulation at* $t = 1.0$ *(time step 637). The raw HPNN prediction displays artefacting near the edges and overshoot at local extrema, however, a single traditional solver iteration aids it to achieve a high degree of accuracy.*

Table 5. *Percentage change to the validation MSE and inference speed of the Poisson CNN when key model architecture features are removed.*

Change to model architecture	Validation MSE change	Inference runtime change
No residual connections in HPNN	+5.56%	−1.26%
MAE loss only DBCNN training	+51.16%	N/A
3 fewer pooling blocks in HPNN	+56.81%	−2.85%
No positional embeddings in HPNN	+89.21%	−0.25%
End-to-end training for nonhomogeneous problems	+544%	N/A

Abbreviations: CNN, convolutional neural network; DBCNN, Dirichlet boundary condition neural network; HPNN, homogeneous Poisson neural network; MSE, mean squared error.

Table 6. *Wall clock runtimes of the multigrid solver versus the DBCNN, HPNN, and Poisson CNN models (in seconds).*

Grid size	Multigrid GPU	Multigrid CPU	DBCNN (4×)	HPNN	Poisson CNN
100	0.2193	0.0361	0.4419	0.1028	0.5811
200	0.3149	0.1101	0.4651	0.1208	0.6276
384	0.6974	0.5469	0.4897	0.1634	0.6936
500	0.8932	0.8580	0.5543	0.2777	0.8744
1,000	4.3026	3.3701	0.7686	0.7917	1.6382
3,000	21.2910	45.6587	2.9436	6.3047	9.6978
4,500	45.4543	106.2516	6.2163	14.2176	21.4086

Abbreviations: CNN, convolutional neural network; CPU, central processing unit; DBCNN, Dirichlet boundary condition neural network; GPU, graphics processing unit; HPNN, homogeneous Poisson neural network.

8.7. Wall-clock runtime

The practical applicability of Poisson CNN's ability to boost the accuracy of multigrid iterations is contingent on its wall-clock runtime. Table 6 outlines the wall-clock runtime of both the sub-models and

the full model using single precision run on an Nvidia V100 GPU, versus multigrid run on 64 threads on a 32-core/64-thread AMD Threadripper 2990WX CPU and the same GPU. Note that the model architecture and weights were not changed compared to the results presented above and the accuracy of the model was not evaluated.

The Poisson CNN model begins to outperform both CPU and GPU multigrid around grid sizes approaching 500×500, eventually building up to five times the speed of CPU multigrid at $4{,}500 \times 4{,}500$, and over twice that of GPU multigrid. While in the case of CPU multigrid there likely exists a severe memory bandwidth bottleneck[7] for such large problems, as the performance of the multigrid algorithm was demonstrated to be memory bandwidth limited by various authors such as Goodnight et al. (2005), our model performs favorably runtime-wise against GPU multigrid as well.

These results show that the NN model proposed in the current study has the potential to offer substantial speedups to solve the Poisson equation for practical applicatons especially in light of the results shown in Sections 8.4 and 8.5.

It is noteworthy that no hyperparameter optimization efforts were conducted for our model. It is likely that by tuning the number of hyperparameters (by e.g., reducing the number of channels in the intermediate layers or the number convolution layers altogether) substantial speedups could be achieved while making little or no compromise in terms of the predictive accuracy of the model, albeit larger model sizes may be necessary for larger problems. The extreme similarity of many of the feature maps in the initial layers of the model is supportive of this possibility. Finally, further very substantial speedups are possible by taking full advantage of quickly developing deep learning acceleration hardware. For example, the "Tensor Cores" on the V100 GPU (which require half-precision operations and specific programming to properly utilize) theoretically offer up to 125TFlops of performance as opposed to 15.2 TFlops of standard single-precision performance as explained in Nvidia Corporation (2019).

9. Conclusion and Future Work

A CNN model to estimate the solution of the 2D Poisson equation with DBCs was developed, based on splitting the problem into a homogeneous Poisson problem and four Laplace problems with one inhomogeneous BC each, with envisioned practical applications to accelerate iterative Poisson solvers by providing initial guesses that are both faster and more accurate than a single iteration of multigrid. Training was done on synthetic, random datasets generated to replicate the smooth functions a Poisson solver would be expected to handle in real world use cases. In order to achieve good convergence during training, the novel L^p integral loss was developed and found to be superior for this task to the MSE.

The model developed can estimate solutions with pointwise deviations below 10% even when given inputs that are materially different from both training and validation data, based on analytical test cases. The predictions are accurate enough such that the model can be used in the pressure projection step of a Navier–Stokes simulation in conjunction with a traditional solver iteration, beating the accuracy of a classical simulation with more iterations per projection step. Comparison of the runtime performance of the model indicates that the runtime at the grid sizes used to generate results for this work is similar to that of multigrid, though with superior accuracy compared to a single cycle. Theoretical speedups up to $5\times$ were observed when using the model with larger inputs. Hence, our model provides a solid foundation to substantially accelerate the solution of the Poisson equation.

Based on the successes showcased in this work, in future work we intend to improve our model by incorporating the following features:

Predicting on larger, previously unseen grids. By fixing the scaling issue demonstrated in the predictions made by the model when encountering larger grids than previously seen, the need for training on a specific range of grid sizes may be completely eliminated.

[7] For comparison, the Nvidia V100 GPU reports over 800GB/s of memory bandwidth, compared to the 100GB/s supplied by the quad channel DDR4 3200MT/s memory available to the CPU.

BC combinations. Many applications require solving the Poisson equation with different combinations of BCs. Though all-Dirichlet and all-Neumann cases were investigated in this work, one possible approach for tackling more complicated combinations can be adopting a strategy similar to the one outlined in Section 3, but since the BCs must retain their respective types in the homogeneous problem as well, having multiple models to solve the homogeneous problem with each combination of BCs possible. The number of separate models can be minimized by grouping BC combinations that are identical under rotations/reflections.

3D grids. The work presented focused on less memory- and time-intensive 2D problems, but most problems of interest in CFD are 3D. It is not expected that vast architectural changes will be necessary for a 3D version of this work since most convolutions etc. can be easily made 3D, however the greater number of parameters and much larger inputs can make training difficult and more memory intensive, necessitating work on model parallelism across GPUs.

Domain adaptation/transfer learning for different domain sizes. As seen in Figure 12, currently the Poisson CNN model's predictions degrade in quality for grids larger than those encountered in training. This problem is analogous to the domain mismatch problem encountered in many deep learning applications such as computer vision, as explored in a survey by Wang and Deng (2018). Adapting transfer learning techniques can be a way to overcome this weakness by training a Poisson CNN model on a very diverse range of data and then fine-tuning for more specific domain ranges as necessary for increased accuracy.

Objects inside domain. Currently our model works only with BCs imposed on the edges of a rectangular domain. Objects inside the domain are commonly encountered in CFD and electrodynamics, requiring imposition of BCs on the surface of these objects. A possible way to tackle this problem is adding a "mask" channel to the inputs with 1.0 values for points lying inside the objects placed in the domain and 0.0 values for points outside, similar to the approach in Tompson et al. (2016).

Hyperparameter optimization. The *hyperparameters* of an NN model such as kernel sizes and number of convolution layer channels play an important role in the accuracy, inference speed and training time of deep NNs. Choosing an optimal size for the model will be crucial to obtain maximum performance especially in comparison to multigrid on GPUs. Gaussian Process based Bayesian hyperparameter search or tree-structured Parzen estimators as explored by Bergstra et al. (2011) are two systematic approaches common in literature for this task.

Funding Statement. This work was supported by a PhD studentship funded by the Department of Aeronautics, Imperial College London.

Competing Interests. The authors declare no competing interests exist.

Data Availability Statement. Readers interested in replicating the results in this work may find the code available on Github, which has been tested to work on AMD64 and IBM POWER9 systems with Nvidia GPUs. We strongly recommend running the code by building a Docker image using the Dockerfile appropriate for your system, under the `docker`/directory in the `development` branch.

Author Contributions. Conceptualization, A.G.Ö., A.H., S.L., B.S., G.R., and P.T.; Formal analysis, A.G.Ö.; Investigation, A.G.Ö.; Methodology, A.G.Ö., A.H., B.S., G.R., and P.T.; Validation, A.G.Ö.; Visualization, A.G.Ö.; Software, A.G.Ö.; Data Curation, A.G.Ö.; Resources, S.L.; Supervision, S.L. and B.S.; Project administration, S.L.; Funding acquisition, S.L.; Writing-original draft, A.G.Ö., S.L., and G.R.; Writing-review & editing, A.G.Ö., A.H., S.L., and G.R.

References

Ajuria Illarramendi E, **Alguacil A**, **Bauerheim M**, **Misdariis A**, **Cuenot B and Benazera E** (2020) Towards an hybrid computational strategy based on deep learning for incompressible flows. In *AIAA AVIATION 2020 FORUM*, p. 3058.

Baymani M, **Kerayechian A and Effati S** (2010) Artificial neural networks approach for solving stokes problem. *Applied Mathematics 1*, 288–292.

Bergstra JS, **Bardenet R**, **Bengio Y and Kégl B** (2011) Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, p. 2546–2554.

Bishop CM (2006) *Pattern Recognition and Machine Learning*. Berlin, Heidelberg: Springer-Verlag.

Chorin AJ (1967) The numerical solution of the navier-stokes equations for an incompressible fluid. *Bulletin of the American Mathematical Society 73*(6), 928–931.

Chorin AJ (1968) Numerical solution of the navier-stokes equations. *Mathematics of computation 22*(104), 745–762.

Dissanayake MWMG and Phan-Thien N (1994) Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering 10*(3), 195–201.

Dumoulin V and Visin F (2016) A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285. 2016 Mar 23.

Goodnight N, **Woolley C**, **Lewin G**, **Luebke D and Humphreys G** (2005) A multigrid solver for boundary value problems using programmable graphics hardware. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH 05. New York, NY: Association for Computing Machinery, p. 193-es.

He K, **Zhang X**, **Ren S and Sun J** (2014) Spatial pyramid pooling in deep convolutional networks for visual recognition. In Fleet D, Pajdla T, Schiele B and Tuytelaars T (eds), *Computer Vision—ECCV 2014*. Cham: Springer International Publishing, pp. 346–361.

He K, **Zhang X**, **Ren S and Sun J** (2015) Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition 2016* (pp. 770–778).

Hochreiter S and Schmidhuber J (1997) Long short-term memory. *Neural Computation 9*(8), 1735–1780.

Isola P, **Zhu J-Y.**, **Zhou T and Efros AA** (2016) Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition 2017* (pp. 1125–1134).

Jianyu L, **Siwei L**, **Yingjian Q and Yaping H** (2003) Numerical solution of elliptic partial differential equation using radial basis function neural networks. *Neural Networks 16*(5), 729–734.

John F (1982) *Partial Differential Equations*. Berlin: Springer.

Kumar M and Yadav N (2011) Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: A survey. *Computers & Mathematics with Applications 62*(10), 3796–3811.

Lagaris IE, **Likas A and Fotiadis DI** (1998) Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks 9*, 987–1000.

Lee H and Kang IS (1990) Neural algorithm for solving differential equations. *Journal of Computational Physics 91*(1), 110–131.

Long Z, **Lu Y**, **Ma X and Dong B** (2018) Pde-net: Learning pdes from data. In *International Conference on Machine Learning 2018 Jul 3* (pp. 3208–3216). PMLR.

Lu L, **Meng X**, **Mao Z and Karniadakis GE** (2021) Deepxde: A deep learning library for solving differential equations. *SIAM Review 63*(1), 208–228.

Lui SH (2011) *Numerical Analysis of Partial Differential Equations*. Hoboken, NJ: John Wiley & Sons, Inc.

Mai-Duy N and Tran-Cong T (2001) Numerical solution of differential equations using multiquadric radial basis function networks. *Neural Networks 14*(2), 185–199.

Marsaglia G (1965) Ratios of normal variables and ratios of sums of uniform variables. *Journal of the American Statistical Association 60*(309), 193–204.

Meng X and Karniadakis GE (2019) A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems. *Journal of Computational Physics 401*, 109020.

Nvidia Corporation (2019) *NVIDIA V100 | NVIDIA.*

Nvidia Corporation (2020) Algebraic multigrid solver (amgx) library. Available at https://github.com/NVIDIA/AMGX.

Odena A, **Dumoulin V and Olah C** (2016) Deconvolution and checkerboard artifacts. *Distill 1*.

Olson LN and Schroder JB (2018) PyAMG: Algebraic multigrid solvers in Python v4.0. Release 4.0.

Raissi M (2018) Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, *19*(1), 932–955.

Raissi M, **Perdikaris P and Karniadakis GE** (2017a) Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. arXiv preprint arXiv:1711.10561.

Raissi M, **Perdikaris P and Karniadakis GE** (2017b) Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. arXiv e-prints, p. arXiv preprint arXiv:1711.10566.

Roberts S and Zhang H (2014) Navier_stokes_2d. Available at https://github.com/acer8/Navier_Stokes_2D.

Ronneberger O, **Fischer P and Brox T** (2015) U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention 2015 Oct 5* (pp. 234–241). Springer, Cham.

Rudy S, **Brunton S**, **Proctor J and Kutz J** (2016) Data-driven discovery of partial differential equations. *Science Advances*, *3*.

Shan T, **Tang W**, **Dang X**, **Li M**, **Yang F**, **Xu S and Wu J** (2017) Study on a poisson's equation solver based on deep learning technique. In *2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS) 2017 Dec 14* (pp. 1–3). IEEE.

Smaoui N and Al-Enezi S (2004) Modelling the dynamics of nonlinear partial differential equations using neural networks. *Journal of Computational and Applied Mathematics 170*(1), 27–58.

Srinath A (2018) pyamgx: Python interface to nvidia's amgx library. Available at https://github.com/shwina/pyamgx.

Tompson J, **Schlachter K**, **Sprechmann P and Perlin K** (2016) Accelerating eulerian fluid simulation with convolutional networks. *ArXiv e-prints.*

Wang M and Deng W (2018) Deep visual domain adaptation: A survey. *Neurocomputing 312*, 135–153.

Weinan E, **Liu J-G** (2003) Gauge method for viscous incompressible flows. *Communications in Mathematical Sciences 1*(2), 317–332.

Xiao X, **Zhou Y**, **Wang H and Yang X** (2018) A novel cnn-based poisson solver for fluid simulation. In *IEEE Transactions on Visualization and Computer Graphics*, p. 1.

Appendix. Higher Aspect Ratio Examples

Below are further examples obtained from a HPNN model, trained with a range of aspect ratios ranging from 0.25 to 4.0. Despite slightly lower performance compared to the models the results of which are investigated in Section 8.1.1, overall performance is still high, demonstrating that our model can adapt to a wide range of aspect ratios similar to those commonly encountered in many applications like computational fluid dynamics.

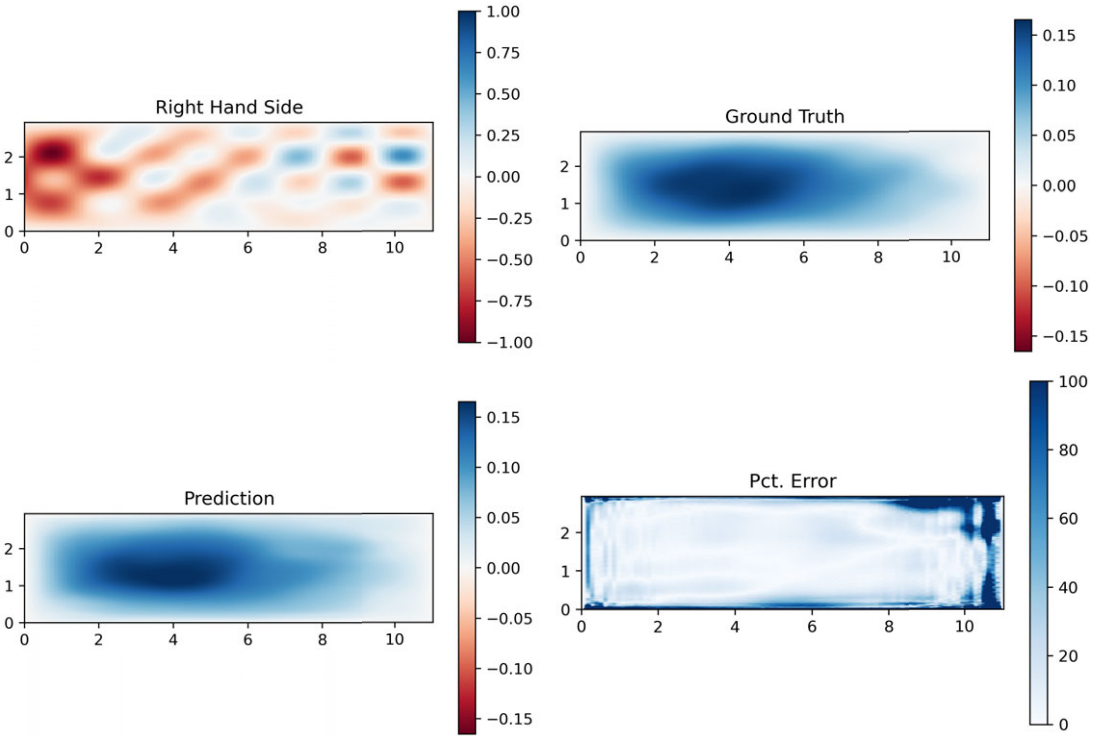

Figure 17. *Prediction of a homogeneous Poisson neural network (NN) model on an example with grid size* 384×96 *and* $\Delta = 2.64 \times 10^{-2}$. *As shown, when trained on an appropriate dataset, the model architecture is capable of handling aspect ratios well above the results in Section 8.*

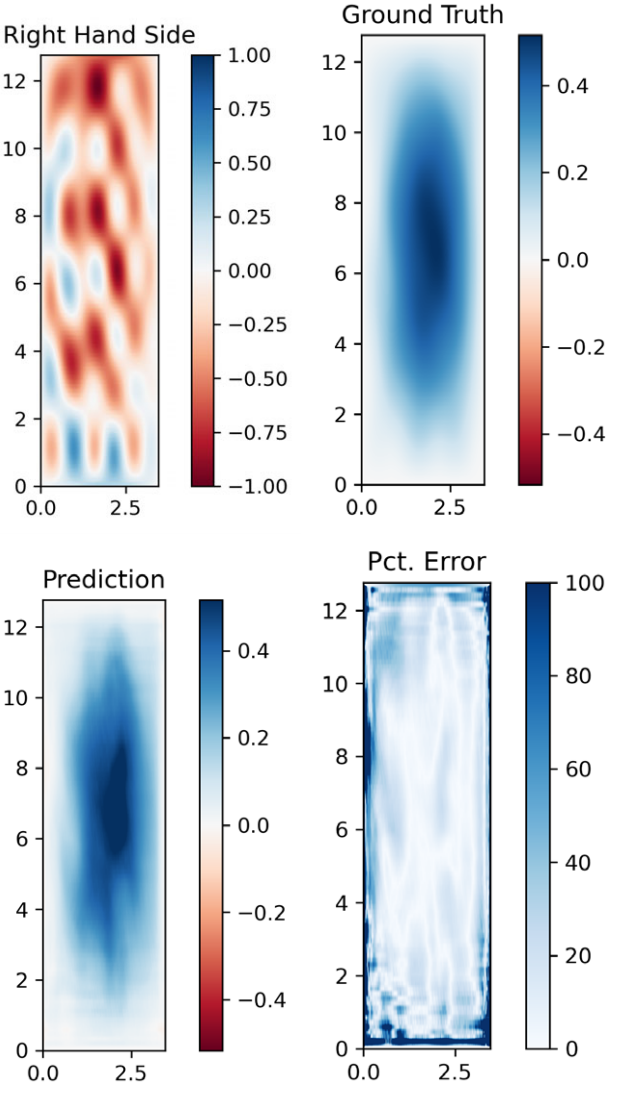

Figure 18. *Prediction of a homogeneous Poisson neural network (NN) model on an example with grid size* 96×384 *and* $\Delta = 3.26 \times 10^{-2}$. *The same model can handle domains that have low aspect ratios as well as those with large ones.*

Cite this article: Özbay AG, Hamzehloo A, Laizet S, Tzirakis P, Rizos G and Schuller B (2021). Poisson CNN: Convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. *Data-Centric Engineering*, 2: e6. doi:10.1017/dce.2021.7