# Separable PINN - Supplementary Materials

## A Source Code

We provided a demo code to show the implementation details of our work. You can run the Jupyter notebook file to reproduce the (2+1)-d Klein-Gordon experiment.

## B Pseudocode

Alg. 1 shows the pseudocode of forward pass and feature merging step of SPINN when solving a 3-dimensional PDE. The outer product and summation between feature vectors can be easily implemented and parallelized by the `einsum` function.

**Algorithm 1** JAX-style pseudocode for SPINN's forward pass and feature merging.

```
import jax.numpy as jnp
from flax import linen as nn

class SPINN(nn.Module):
    hidden_sizes: Sequence[int] # list of feature sizes for each layer

    # forward pass and feature merging
    @nn.compact
    def forward(self, x, y, z): # 3-dimensional PDE
        inputs = [x, y, z]   # full batch of the sampled coordinates
        feats = []
        for X in inputs:    # three individual MLPs for each input x, y, z
            for hidden_size in self.hidden_sizes[:-1]:
                X = nn.Dense(hidden_size)(X)
                X = nn.activation.tanh(X)
            """
            the output feature size (self.hidden_sizes[-1]) corresponds to the
            rank of the predicted solution tensor (in case of 1-dimensional output)
            """
            X = nn.Dense(self.hidden_sizes[-1])(X)
            feats += [X] # final feature representations
        # feature merging (outer product between columns of feats)
        xy = jnp.einsum('fx, fy->fxy', feats[:, 0], feats[:, 1])
        return jnp.einsum('fxy, fz->xyz', xy, feats[:, 2])
```

## C Proof of Theorem 1

Here we show the preliminary lemmas and proofs for Theorem 1. in the main paper. We start by defining a general tensor product between two Hilbert spaces.

**Definition 1.** Let $\{v_\beta\}$ be an orthonormal basis for $\mathcal{H}_2$. **Tensor product** between Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$, denoted by $\mathcal{H}_1 \otimes \mathcal{H}_2$, is a set of all antilinear mappings $A : \mathcal{H}_2 \to \mathcal{H}_1$ such that $\sum_\beta \|Av_\beta\|^2 < \infty$ for every orthonormal basis for $\mathcal{H}_2$.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Then by Theorem 7.12 in Folland [3], $\mathcal{H}_1 \otimes \mathcal{H}_2$ is also a Hilbert space with respect to norm $|||\cdot|||$ and associated inner product $\langle \cdot, \cdot \rangle$:

$$|||A|||^2 \equiv \sum_\beta \|Av_\beta\|^2, \tag{1}$$

$$\langle A, B \rangle \equiv \sum_\beta \langle Av_\beta, Bv_\beta \rangle, \tag{2}$$

where $A, B \in \mathcal{H}_1 \otimes \mathcal{H}_2$, and $\{v_\beta\}$ is any orthonormal basis of $\mathcal{H}_2$.

**Lemma 1.** *Let $x \in \mathcal{H}_1$ and $y, y' \in \mathcal{H}_2$. Then, $(x \otimes y)y' = \langle y, y' \rangle x$.*

*Proof.* Let $A$ be a mapping $A : y' \to \langle y, y' \rangle x$, where $\|y\| = 1$. We expand $y'$ with orthonormal basis $\{y, z_\beta\}$. i.e., $\{y, z_\beta\}$ is a basis for $\mathcal{H}_2$. Then,

$$\|Ay\|^2 + \sum_\beta \|Az_\beta\|^2 = \|Ay\|^2 + \sum_\beta \|\langle y, z_\beta \rangle x\|^2 \tag{3}$$

$$= \|Ay\|^2 < \infty \tag{4}$$

It is obvious that $A$ is antilinear. Then by Definition 1, $A \in \mathcal{H}_1 \otimes \mathcal{H}_2$ which is, $Ay' = (x \otimes y)y'$. Therefore, $(x \otimes y)y' = \langle y, y' \rangle x$.

□

**Lemma 2.** *$\{u_\alpha \otimes v_\beta\}$ is an orthonormal basis for $\mathcal{H}_1 \otimes \mathcal{H}_2$.*

*Proof.* Let $A, B \in \mathcal{H}_1 \otimes \mathcal{H}_2$, where $B = u_\alpha \otimes v_\beta$. Then by the definition of inner product in Eq. 2,

$$\langle A, B \rangle = \sum_i \langle Av_i, Bv_i \rangle \tag{5}$$

$$= \sum_i \langle Av_i, (u_\alpha \otimes v_\beta)v_i \rangle \tag{6}$$

$$= \sum_i \langle Av_i, \langle v_\beta, v_i \rangle u_\alpha \rangle \qquad (\because \text{Lemma 1}) \tag{7}$$

$$= \cancel{\langle Av_1, \langle v_\beta, v_1 \rangle u_\alpha \rangle} + \cancel{\langle Av_2, \langle v_\beta, v_2 \rangle u_\alpha \rangle} + \ldots + \langle Av_\beta, \langle v_\beta, v_\beta \rangle u_\alpha \rangle + \ldots \tag{8}$$

$$= \langle Av_\beta, u_\alpha \rangle \tag{9}$$

Now we check the Parseval identity:

$$\sum_{\alpha,\beta} |\langle A, u_\alpha \otimes v_\beta \rangle|^2 = \sum_{\alpha,\beta} |\langle Av_\beta, u_\alpha \rangle|^2 \qquad (\because u_\alpha \otimes v_\beta = B) \tag{10}$$

$$= \sum_\beta \|Av_\beta\|^2 \tag{11}$$

$$= |||A|||^2. \tag{12}$$

$\therefore \{u_\alpha \otimes v_\beta\}$ is a basis.

□

Now we begin the proof of Theorem 1. in the main paper.

**Theorem 1.** *Let $X, Y$ be compact subsets of $\mathbb{R}^d$. Choose $u \in L^2(X \times Y)$. Then, for arbitrary $\varepsilon > 0$, we can find a sufficiently large $r > 0$ and neural networks $f_j^{(\theta_1)}$ and $f_j^{(\theta_2)}$ such that*

$$\left\| u - \sum_{j=1}^r f_j^{(\theta_1)} f_j^{(\theta_2)} \right\|_{L^2(X \times Y)} < \varepsilon. \tag{13}$$

*Proof.* Let $\{\phi_i\}$ and $\{\psi_j\}$ be orthonormal basis for $L^2(X)$ and $L^2(Y)$ respectively. Then $\{\phi_i\psi_j\}$ forms an orthonormal basis for $L^2(X\times Y)$ ($\because$ Lemma 2). Therefore, we can find a sufficiently large $r$ such that

$$\Big\|u-\sum_{i,j}^{r} a_{ij}\phi_i\psi_j\Big\|_{L^2(X\times Y)} < \frac{\varepsilon}{2}, \tag{14}$$

where $a_{ij}$ denotes

$$a_{ij} = \int_{X\times Y} u(x,y)\phi_i(x)\psi_j(y)dxdy.$$

On the other hand, by the universal approximation theorem [1], we can find neural networks $f_j^{(\theta_1)}$ and $f_j^{(\theta_2)}$ such that

$$\|\phi_i - f_j^{(\theta_1)}\|_{L^2(X)} \le \frac{\varepsilon}{3^j\|u\|_{L^2(X\times Y)}} \text{ and } \|\psi_j - f_j^{(\theta_2)}\|_{L^2(Y)} \le \frac{\varepsilon}{3^j\|u\|_{L^2(X\times Y)}}. \tag{15}$$

We first consider the difference between $u$ and $\sum_{i,j}^r a_{ij}f_i^{(\theta_1)}f_j^{(\theta_2)}$:

$$\begin{aligned}
&\|u-\sum_{i,j}^{r} a_{ij}f_i^{(\theta_1)}f_j^{(\theta_2)}\|_{L^2(X\times Y)} && (16)\\
&\quad\le \Big\|u-\sum_{i,j}^{r} a_{ij}\phi_i\psi_j\Big\|_{L^2(X\times Y)} + \Big\|\sum_{i,j}^{r} a_{ij}\phi_i\psi_j - \sum_{i,j}^{r} a_{ij}f_i^{(\theta_1)}f_j^{(\theta_2)}\Big\|_{L^2(X\times Y)} && (17)\\
&\quad\equiv I + II && (18)
\end{aligned}$$

Since $|I| < \varepsilon/2$ from (14), it is enough to estimate $II$. For this, we consider

$$\begin{aligned}
\sum_{i,j}^{r} a_{ij}\phi_i\psi_j - \sum_{i,j}^{r} a_{ij}f_i^{(\theta_1)}f_j^{(\theta_2)} &= \sum_{i,j}^{r} a_{ij}\phi_i\big(\psi_j - f_j^{(\theta_2)}\big) + \sum_{i,j}^{r} a_{ij}\big(\phi_i - f_i^{(\theta_1)}\big)f_j^{(\theta_2)} && (19)\\
&\equiv II_1 + II_2. && (20)
\end{aligned}$$

We first compute $II_1$:

$$\begin{aligned}
\|II_1\|^2_{L^2(X\times Y)} &= \int_{X\times Y}\left\{\sum_{i,j}^{r} a_{ij}\phi_i\big(\psi_j - f_j^{(\theta_2)}\big)\right\}^2 dxdy && (21)\\
&= \int_{X\times Y}\left\{\sum_{j}^{r}\left(\sum_{i}^{r} a_{ij}\phi_i\right)\big(\psi_j - f_j^{(\theta_2)}\big)\right\}^2 dxdy && (22)
\end{aligned}$$

We set

$$A_j(x) = \sum_{i}^{r} a_{ij}\phi_i(x), \qquad B_j(y) = \psi_j(y) - f_j^{(\theta_2)}(y) \tag{23}$$

to write $II_1$ as

$$\|II_1\|^2_{L^2(X\times Y)} = \int_{X\times Y}\left\{\sum_{j}^{r} A_j(x)B_j(y)\right\}^2 dxdy. \tag{24}$$

We then apply Cauchy-Scharwz inequality to get

$$\begin{aligned}
\|II_1\|^2_{L^2(X\times Y)} &\le \int_{X\times Y}\Big(\sum_{j}^{r}|A_j(x)|^2\Big)\Big(\sum_{j}^{r}|B_j(y)|^2\Big)dxdy && (25)\\
&= \left(\int_X \sum_{j}^{r}|A_j(x)|^2dx\right)\left(\int_Y \sum_{j}^{r}|B_j(y)|^2dy\right). && (26)
\end{aligned}$$

Now

$$\int_X \sum_j^r |A_j(x)|^2 dx = \int_X \sum_j^r \left( \sum_i^r a_{ij}\phi_i \right)^2 dx \tag{27}$$

$$= \sum_j^r \Big\| \sum_i^r a_{ij}\phi_i \Big\|_{L(X)}^2. \tag{28}$$

Since $\{\phi_i\}$ is an orthonormal basis, we see that

$$\Big\| \sum_i^r a_{ij}\phi_i \Big\|_{L(X)} \le \sum_i^r |a_{ij}| \big\|\phi_i\big\|_{L(X)} = \sum_i^r |a_{ij}|. \tag{29}$$

Therefore,

$$\int_X \sum_j^r |A_j(x)|^2 dx = \int_X \sum_j^r \left( \sum_i^r a_{ij}\phi_i \right)^2 dx \tag{30}$$

$$= \sum_j^r \int_X \left( \sum_i^r a_{ij}\phi_i \right)^2 dx \tag{31}$$

$$= \sum_j^r \Big\| \sum_i^r a_{ij}\phi_i \Big\|_{L(X)}^2 \tag{32}$$

$$= \sum_j^r \Big( \sum_i^r |a_{ij}| \Big)^2. \tag{33}$$

Finally, we recall

$$\Big( \sum_i^r |a_{ij}| \Big)^2 \le 2 \sum_i^r |a_{ij}|^2 \tag{34}$$

to conclude

$$\int_X \sum_j^r |A_j(x)|^2 dx \le 2 \sum_j^r \sum_i^r |a_{ij}|^2 < 2 \sum_{i,j}^{\infty} |a_{ij}|^2 = 2\|u\|_{L^2(X\times Y)}^2. \tag{35}$$

On the other hand, we have from (15)

$$\int_Y \sum_j^r |B_j(y)|^2 dy = \sum_j^r \int_Y |\psi_j(y) - f_j^{(\theta_2)}(y)|^2 dy \tag{36}$$

$$= \sum_j^r \big\|\psi_j(y) - f_j^{(\theta_2)}(y)\big\|_{L^2(Y)}^2 \tag{37}$$

$$\le \sum_j^r \frac{\varepsilon^2}{9^j \|u\|_{L^2(X\times Y)}^2} \tag{38}$$

$$< \frac{\varepsilon^2}{8\|u\|_{L^2(X\times Y)}^2}. \tag{39}$$

Hence we have

$$\|II_1\|_{L^2(X\times Y)}^2 < \frac{\varepsilon^2}{8\|u\|_{L^2(X\times Y)}^2} 2\|u\|_{L^2(X\times Y)}^2 = \frac{\varepsilon^2}{4}. \tag{40}$$

Likewise,

$$\|II_2\|_{L^2(X\times Y)}^2 < \frac{\varepsilon^2}{4}. \tag{41}$$

Therefore,

$$\|II\|^2_{L^2(X\times Y)} < \frac{\varepsilon^2}{2}. \tag{42}$$

We go back to (16) with the estimates (14) and (42) to derive

$$\Big\|u - \sum_{i,j}^{r} a_{ij} f_i^{(\theta_1)} f_j^{(\theta_2)}\Big\|_{L^2(X\times Y)} < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon. \tag{43}$$

Finally, we reorder the index to rewrite

$$\sum_{i,j}^{r} a_{ij} f_i^{(\theta_1)} f_j^{(\theta_2)} = \sum_{i}^{\tilde{r}} b_i \tilde{f}_i^{(\theta_1)} \tilde{f}_i^{(\theta_2)} \tag{44}$$

$$= \sum_{i}^{\tilde{r}} \left\{ b_i \tilde{f}_i^{(\theta_1)} \right\} \tilde{f}_i^{(\theta_2)} \tag{45}$$

$$= \sum_{i}^{\tilde{r}} g_i^{(\theta_1)} \tilde{f}_i^{(\theta_2)} \tag{46}$$

Without loss of generality, we rewrite $\tilde{r}, g_i^{(\theta_1)}, \tilde{f}_i^{(\theta_2)}$ into $r, f_i^{(\theta_1)}, f_i^{(\theta_2)}$ respectively, to complete the proof. □

# D Training with Physics-Informed Loss

After SPINN predicts an output function with the methods described above, the rest of the training procedure follows the same process used in conventional PINN training [11], except we use forward-mode AD to compute PDE residuals (standard back-propagation, a.k.a. reverse-mode AD for parameter updates). With the slight abuse of notation, our predicted solution function is denoted as $\hat{u}^{(\theta)}(x,t)$ from onwards, explicitly expressing time coordinates. Given an underlying PDE (or ODE), the initial, and the boundary conditions, SPINN is trained with a 'physics-informed' loss function:

$$\min_{\theta} \mathcal{L}(\hat{u}^{(\theta)}(x,t)) = \min_{\theta} \lambda_{\text{pde}}\mathcal{L}_{\text{pde}} + \lambda_{\text{ic}}\mathcal{L}_{\text{ic}} + \lambda_{\text{bc}}\mathcal{L}_{\text{bc}}, \tag{47}$$

$$\mathcal{L}_{\text{pde}} = \int_{\Gamma}\int_{\Omega} \|\mathcal{N}[\hat{u}^{(\theta)}](x,t)\|^2 dxdt, \tag{48}$$

$$\mathcal{L}_{\text{ic}} = \int_{\Omega} \|\hat{u}^{(\theta)}(x,0) - u_{\text{ic}}(x)\|^2 dx, \tag{49}$$

$$\mathcal{L}_{\text{bc}} = \int_{\Gamma}\int_{\partial\Omega} \|\mathcal{B}[\hat{u}^{(\theta)}](x,t) - u_{\text{bc}}(x,t)\|^2 dxdt, \tag{50}$$

where $\Omega$ is an input domain, $\mathcal{N}, \mathcal{B}$ are generic differential operators and $u_{\text{ic}}, u_{\text{bc}}$ are initial, boundary conditions, respectively. $\lambda$ are weighting factors for each loss term. When calculating the PDE loss ($\mathcal{L}_{\text{pde}}$) with Monte-Carlo integral approximation, we sampled collocation points from factorized coordinates and used forward-mode AD. The remaining $\mathcal{L}_{\text{ic}}$ and $\mathcal{L}_{\text{bc}}$ are then computed with initial and boundary coordinates to regress the given conditions. By minimizing the objective loss in Eq. 47, the model output is enforced to satisfy the given equation, the initial, and the boundary conditions.

# E FLOPs Estimation

The FLOPs for evaluating the derivatives can be systematically calculated by disassembling the computational graph into elementary operations such as additions and multiplications. Given a computational graph of forward pass for computing the primals, AD augments each elementary operation into other elementary operations. The FLOPs in the forward pass can be precisely calculated since it consists of a series of matrix multiplications and additions. We used the method described in [4] to estimate FLOPs for evaluating the derivatives. Table 1 shows the number of additions

(ADDS) and multiplications (MULTS) in each evaluation process. Note that FLOPs is a summation of ADDS and MULTS by definition.

One thing to note here is that this is a theoretical estimation. Theoretically, the number of JVP evaluations for computing the gradient with respect to the input coordinates is $Nd$, when $N$ is the number of coordinates for each axis and $d$ is the input dimension (see section 4.3 in the main paper). However, our actual implementation of gradient calculation involves re-computing the feature representations $f^{(\theta_i)}$, which makes the complexity of network propagations from $\mathcal{O}(Nd)$ to $\mathcal{O}(Nd^2)$. Ideally, these feature vectors can be computed only once and stored to be used later for gradient computations. Although it is still significantly more efficient than the conventional PINN's complexity ($Nd^2 \ll N^d$), there is a room to bridge the gap between theoretical FLOPs and actual training runtime by further software optimization.

Table 1: The number of elementary operations for evaluating forward pass, first and second-order derivatives. The calculation is based on $64^3$ collocation points in a 3-d system and the vanilla MLP settings used for diffusion, Helmholtz, and Klein-Gordon equations. We assumed that each derivative is evaluated on every coordinate axis.

| | SPINN (ours) | | PINN (baseline) | |
|---|---|---|---|---|
| | ADDS ($\times 10^6$) | MULTS ($\times 10^6$) | ADDS ($\times 10^6$) | MULTS ($\times 10^6$) |
| forward pass | 20 | 20 | 21,609 | 21,609 |
| 1st-order derivative | 40 | 40 | 86,638 | 43,419 |
| 2nd-order derivative | 80 | 80 | 130,057 | 87,040 |
| MFLOPs (total) | 280 | | 390,370 | |

# F Experimental Details and Results

In this section, we provide experimental details, numerical results, and visualizations for each experiment in the main paper.

## F.1 Diffusion Equation

The diffusion equation is one of the most representative parabolic PDEs, often used for modeling the heat diffusion process. We especially choose a nonlinear diffusion equation where it can be written as:

$$\partial_t u = \alpha \left( \|\nabla u\|^2 + u \Delta u \right), \qquad x \in \Omega, t \in \Gamma, \tag{51}$$

$$u(x, 0) = u_{\text{ic}}(x), \qquad x \in \Omega, \tag{52}$$

$$u(x, t) = 0, \qquad x \in \partial\Omega, t \in \Gamma. \tag{53}$$

We used diffusivity $\alpha = 0.05$, spatial domain $\Omega = [-1, 1]^2$, temporal domain $\Gamma = [0, 1]$ and used superposition of three Gaussian functions for the initial condition $u_{\text{ic}}$. We obtained the reference solution ($101 \times 101 \times 101$ resolution) through a widely-used PDE solver platform FEniCS [9]. Note that FEniCS is a FEM-based solver. We particularly set the initial condition to be a superposition of three gaussian functions:

$$u_{\text{ic}}(x, y) = 0.25 \exp\left[-10\{(x - 0.2)^2 + (y - 0.3)^2\}\right] + 0.4 \exp\left[-15\{(x + 0.1)^2 + (y + 0.5)^2\}\right] + 0.3 \exp\left[-20\{(x + 0.5)^2 + y^2\}\right]. \tag{54}$$

For our model, we used three body networks of 4 hidden layers with 64/32 hidden feature/output size each. For the baseline model, we used a single MLP of 5 hidden layers with 128 hidden feature sizes. We used Adam [6] optimizer with a learning rate of 0.001 and trained for 50,000 iterations for every experiment. All weight factors $\lambda$ in the loss function in Eq. 47 are set to 1. The final reported errors are extracted where the total loss was minimum across the entire training iteration. We also resampled the input points every 100 epochs. Tab. 3 shows the numerical results, and the visualized solutions are provided in Fig. 1.

### F.2 Helmholtz Equation

The Helmholtz equation is a time-independent wave equation that takes the form:

$$\Delta u + k^2 u = q, \qquad x \in \Omega, \tag{55}$$
$$u(x) = 0, \qquad x \in \partial\Omega, \tag{56}$$

where the spatial domain is $\Omega = [-1, 1]^3$. For a given source term $q = -(a_1\pi)^2 u - (a_2\pi)^2 u - (a_3\pi)^2 u + k^2 u$, we devised a manufactured solution $u = \sin(a_1\pi x_1)\sin(a_2\pi x_2)\sin(a_3\pi x_3)$, where we take $k = 1, a_1 = 4, a_2 = 4, a_3 = 3$.

For our model, we used three body networks of 4 hidden layers with 64/32 hidden feature/output size each. For the baseline model, we used a single MLP of 5 hidden layers with 128 hidden feature sizes. We used Adam [6] optimizer with a learning rate of 0.001 and trained for 50,000 iterations for every experiment. All weight factors $\lambda$ in the loss function in Eq. 47 are set to 1. The final reported errors are extracted where the total loss was minimum across the entire training iteration. We also resampled the input points every 100 epochs. Tab. 4 shows the numerical results and the visualized solutions are provided in Fig. 2.

### F.3 Klein-Gordon Equation

The Klein-Gordon equation is a nonlinear hyperbolic PDE, which arises in diverse applied physics for modeling relativistic wave propagation. The inhomogeneous Klein-Gordon equation is given by

$$\partial_{tt} u - \Delta u + u^2 = f, \qquad x \in \Omega, t \in \Gamma, \tag{57}$$
$$u(x, 0) = x_1 + x_2, \qquad x \in \Omega, \tag{58}$$
$$u(x, t) = u_{\text{bc}}(x), \qquad x \in \partial\Omega, t \in \Gamma, \tag{59}$$

where we chose the spatial/temporal domain to be $\Omega = [-1, 1]^2$ and $\Gamma = [0, 10]$, respectively. For error measurement, we used a manufactured solution $u = (x_1 + x_2)\cos(2t) + x_1 x_2 \sin(2t)$ and $f$, $u_{bc}$ are extracted from this exact solution.

For our model, we used three body networks of 4 hidden layers with 64/32 hidden feature/output size each. For the baseline model, we used a single MLP of 5 hidden layers with 128 hidden feature sizes. We used Adam [6] optimizer with a learning rate of 0.001 and trained for 50,000 iterations for every experiment. All weight factors $\lambda$ in the loss function in Eq. 47 are set to 1. The final reported errors are extracted where the total loss was minimum across the entire training iteration. We also resampled the input points every 100 epochs. Tab. 5 shows the numerical results.

We used the same settings used in (2+1)-d Klein-Gordon experiment for the (3+1)-d experiment except the manufactured solution was chosen as:

$$u = (x_1 + x_2 + x_3)\cos(2t) + x_1 x_2 x_3 \sin(2t), \tag{60}$$

where $f$, $u_{bc}$ are extracted from this exact solution. Tab. 6 shows the numerical results. The collocation sizes of $23^4, 18^4$ were the maximum value for PINN and PINN with modified MLP, respectively.

### F.4 (2+1)-d Navier-Stokes Equation

Navier-Stokes equation is a nonlinear time-dependent PDE that describes the motion of a viscous fluid. Various engineering fields rely on this equation, such as modeling the weather, airflow, or ocean currents. The vorticity form for incompressible fluid can be written as below:

$$\partial_t \omega + u \cdot \nabla \omega = \nu \Delta \omega, \qquad x \in \Omega, t \in \Gamma, \tag{61}$$
$$\nabla \cdot u = 0, \qquad x \in \Omega, t \in \Gamma, \tag{62}$$
$$\omega(x, 0) = \omega_0(x), \qquad x \in \Omega, \tag{63}$$

where $u \in \mathbb{R}^2$ is the velocity field, $\omega = \nabla \times u$ is the vorticity, $\omega_0$ is the initial vorticity, and $\nu$ is the viscosity. We used the viscosity 0.01 and made the spatial/temporal domain $\Omega = [0, 2\pi]^2$ and $\Gamma = [0, 1]$, respectively. Note that Eq. 61 models decaying turbulence since there is no forcing term and Eq. 62 is the incompressible fluid condition. The reference solution is generated by JAX-CFD

solver [7] which specifically used the pseudo-spectral method. The initial condition was generated using the gaussian random field with a maximum velocity of 5. The resolution of the obtained solution is $100 \times 128 \times 128$ ($N_t \times N_x \times N_y$), and we tested our model on this data.

For our model, we used three body networks (modified MLP) of 3 hidden layers with 128/256 hidden feature/output sizes each. We divided the temporal domain into ten time windows to adopt the time marching method [8, 13]. We used Adam [6] optimizer with a learning rate of 0.002 and each time window is trained for 100,000 iterations. Followed by causal PINN, the PDE (residual) loss and initial condition loss function are written as follows.

$$\mathcal{L}_{\text{pde}} = \frac{\lambda_w}{N_c} \sum^{N_c} |\partial_t w + u_x \partial_x w + u_y \partial_y w - \nu(\partial_{xx} w + \partial_{yy} w)|^2 + \frac{\lambda_c}{N_c} \sum^{N_c} |\partial_x u_x + \partial_y u_y|^2, \quad (64)$$

$$\mathcal{L}_{\text{ic}} = \frac{\lambda_{ic}}{N_{ic}} \sum^{N_{ic}} \left( |u_x - u_{x0}|^2 + |u_y - u_{y0}|^2 + |w - w_0|^2 \right), \quad (65)$$

where $u_x, u_y$ are $x, y$ components of predicted velocity, $w = \partial_x u_y - \partial_y u_x$, $N_c$ is the collocation size, $N_{ic}$ is the number of coordinates for initial condition and $u_{x0}, u_{y0}, u_{w0}$ are the initial conditions. We chose the weighting factors $\lambda_w = 1$, $\lambda_c = 5,000$, and $\lambda_{ic} = 10,000$. We also resampled the input points every 100 epochs. The periodic boundary condition can be explicitly enforced by positional encoding [2], and we specifically used the following encoding function only for the spatial input coordinates.

$$\gamma(x) = [1, \sin(x), \sin(2x), \sin(3x), \sin(4x), \sin(5x), \cos(x), \cos(2x), \cos(3x), \cos(4x), \cos(5x)]^\top. \quad (66)$$

Unlike other experiments, the solution of the Navier-Stokes equation is a 2-dimensional vector-valued function $u : \mathbb{R}^3 \to \mathbb{R}^2$. We can rewrite the feature merging equation Eq. 5 in the main paper to construct SPINN into a 2-dimensional vector function:

$$u_1 = \sum_{j=1}^{r} \prod_{i=1}^{d} f_j^{(\theta_i)}(x_i), \quad (67)$$

$$u_2 = \sum_{j=r+1}^{2r} \prod_{i=1}^{d} f_j^{(\theta_i)}(x_i). \quad (68)$$

For example in our (2+1)-d Navier-Stokes equation setting, $r = 128$ since the network output feature size is 256. This can be applied to any $m$-dimensional vector function if we use a larger output feature size. More formally, if we want to construct an $m$-dimensional vector function with SPINN of rank $r$, the $k$-th element of the function output can be written as

$$u_k = \sum_{j=(k-1)r+1}^{kr} \prod_{i=1}^{d} f_j^{(\theta_i)}(x_i), \quad (69)$$

where the output feature size of each body network is $mr$.

### F.5 (3+1)-d Navier-Stokes Equation

The vorticity form of (3+1)-d Navier-Stokes equation is given as:

$$\partial_t \omega + (u \cdot \nabla)\omega = (\omega \cdot \nabla)u + \nu \Delta \omega + F, \qquad x \in \Omega, t \in \Gamma, \quad (70)$$

$$\nabla \cdot u = 0, \qquad x \in \Omega, t \in \Gamma, \quad (71)$$

$$\omega(x, 0) = \omega_0(x), \qquad x \in \Omega. \quad (72)$$

We constructed the spatial/temporal domain to be $\Omega = [0, 2\pi]^3$ and $\Gamma = [0, 5]$, respectively. We constructed the analytic solution for (3+1)-d Navier-Stokes equation introduced in Taylor et al. [12].

The manufactured velocity and vorticity are

$$u_x = 2e^{-9\nu t}\cos(2x)\sin(2y)\sin(z), \tag{73}$$
$$u_y = -e^{-9\nu t}\sin(2x)\cos(2y)\sin(z), \tag{74}$$
$$u_z = -2e^{-9\nu t}\sin(2x)\sin(2y)\cos(z), \tag{75}$$
$$\omega_x = -3e^{-9\nu t}\sin(2x)\cos(2y)\cos(z), \tag{76}$$
$$\omega_y = 6e^{-9\nu t}\cos(2x)\sin(2y)\cos(z), \tag{77}$$
$$\omega_z = -6e^{-9\nu t}\cos(2x)\cos(2y)\sin(z), \tag{78}$$

where we chose the viscosity to be $\nu = 0.05$. Each forcing term $F$ in the Eq. 70 is then given as

$$F_x = -6e^{-18\nu t}\sin(4y)\sin(2z), \tag{79}$$
$$F_y = -6e^{-18\nu t}\sin(4x)\sin(2z), \tag{80}$$
$$F_z = 6e^{-18\nu t}\sin(4x)\sin(4y). \tag{81}$$

We constructed SPINN to be three body networks (modified MLP) of 5 hidden layers with 64/384 hidden feature/output sizes each. We used Adam [6] optimizer with a learning rate of 0.001 and trained for 50,000 iterations. The weight factors in the loss function in Eq. 48 are chosen as $\lambda_{\text{pde}} = 1$, $\lambda_{\text{ic}} = 10$, and $\lambda_{\text{bc}} = 1$. We also weighted the incompressibility loss (Eq. 71) with 100. The visualized solution vector field is shown in Fig. 4.

# G Additional Experiments

## G.1 (5+1)-d Heat Equation

We tested our model on (5+1)-d heat equation to verify the effectiveness of our model on higher dimensional PDE:

$$\frac{\partial u(t,x)}{\partial t} = \Delta u(t,x), \qquad x \in [-1,1]^5, t \in [0,1], \tag{82}$$

where the manufactured solution is chosen to be $\|x\|^2 + 10t$. When trained with $8^6$ collocation points, SPINN achieved a relative $L_2$ error of 0.0074 within 2 minutes.

## G.2 Fine Tuning with L-BFGS

We also conducted some experiments to explore the use of L-BFGS when training SPINN. We found that training with Adam first and then fine-tuning with L-BFGS showed a slight increase in accuracy. Note that this training strategy is used by other works [5, 10] and is known to be effective in some cases. Tab. 2 shows the numerical results on three 3-d PDEs. Understanding the effect of the optimization algorithm is still an open question in PINNs, we believe that investigating this issue in the context of SPINN would be a valuable direction for future study.

Table 2: Numerical result of 3-d PDEs with L-BFGS fine tuning. The number of training collocation points is $64^3$.

| | Diffusion | Helmholtz | Klein-Gordon |
|---|---|---|---|
| Adam | 0.0041 | 0.0360 | 0.0013 |
| Adam + L-BFGS | 0.0041 | 0.0308 | 0.0010 |

Table 3: Full results of **diffusion equation**. $N_c$ is the number of collocation points.

| model | $N_c$ | relative $L_2$ error | runtime (ms/iter.) | memory (MB) |
|---|---|---|---|---|
| PINN | $16^3$ | 0.0095 | 3.98 | 1,022 |
| | $32^3$ | 0.0082 | 12.82 | 2,942 |
| | $64^3$ | 0.0081 | 95.22 | 18,122 |
| PINN + modified MLP | $16^3$ | 0.0048 | 14.94 | 1,918 |
| | $32^3$ | 0.0043 | 29.91 | 4,990 |
| | $54^3$ | 0.0041 | 134.64 | 22,248 |
| SPINN | $16^3$ | 0.0447 | 1.45 | 766 |
| | $32^3$ | 0.0115 | 1.76 | 766 |
| | $64^3$ | 0.0075 | 1.90 | 766 |
| | $128^3$ | 0.0061 | 2.09 | 894 |
| | $256^3$ | 0.0061 | 10.54 | 2,174 |
| SPINN + modified MLP | $16^3$ | 0.0390 | 2.17 | 766 |
| | $32^3$ | 0.0067 | 2.44 | 768 |
| | $64^3$ | 0.0041 | 2.59 | 768 |
| | $128^3$ | **0.0036** | 3.06 | 896 |
| | $256^3$ | **0.0036** | 12.13 | 2,176 |

Table 4: Full results of the **Helmholtz equation**. $N_c$ is the number of collocation points.

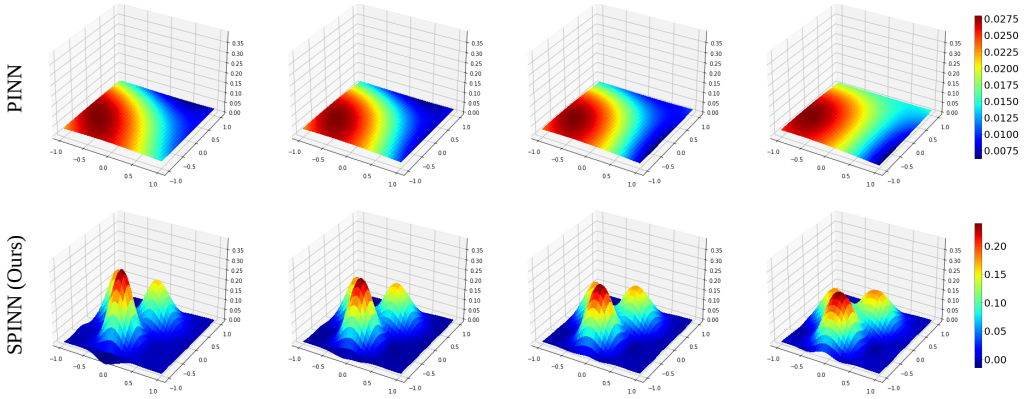
| model | $N_c$ | relative $L_2$ error | runtime (ms/iter.) | memory (MB) |
|---|---|---|---|---|
| PINN | $16^3$ | 0.9819 | 4.84 | 2,810 |
| | $32^3$ | 0.9757 | 14.84 | 2,938 |
| | $64^3$ | 0.9723 | 110.23 | 18,118 |
| PINN + modified MLP | $16^3$ | 0.4770 | 18.32 | 7,034 |
| | $32^3$ | 0.5176 | 35.02 | 9,082 |
| | $54^3$ | 0.4770 | 159.90 | 22,244 |
| SPINN | $16^3$ | 0.1177 | 1.54 | 762 |
| | $32^3$ | 0.0809 | 1.71 | 762 |
| | $64^3$ | 0.0592 | 1.85 | 762 |
| | $128^3$ | 0.0449 | 1.89 | 762 |
| | $256^3$ | 0.0435 | 3.84 | 1,146 |
| SPINN + modified MLP | $16^3$ | 0.1161 | 2.24 | 764 |
| | $32^3$ | 0.0595 | 2.50 | 764 |
| | $64^3$ | 0.0360 | 2.57 | 764 |
| | $128^3$ | **0.0300** | 2.76 | 764 |
| | $256^3$ | 0.0311 | 5.50 | 1,148 |

Table 5: Full results of the **(2+1)-d Klein-Gordon equation**. $N_c$ is the number of collocation points.

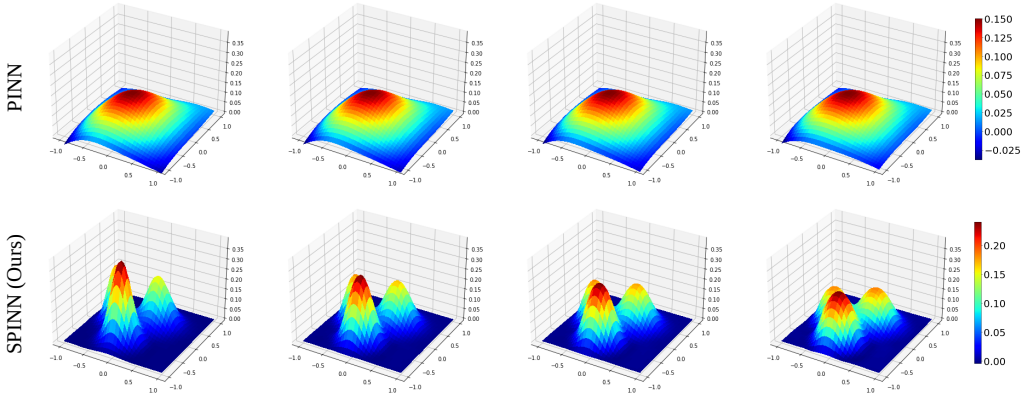
| model | $N_c$ | relative $L_2$ error | runtime (ms/iter.) | memory (MB) |
|---|---|---|---|---|
| PINN | $16^3$ | 0.0343 | 4.70 | 2,810 |
| | $32^3$ | 0.0281 | 14.95 | 2,938 |
| | $64^3$ | 0.0299 | 112.00 | 18,118 |
| PINN + modified MLP | $16^3$ | 0.0158 | 17.87 | 7,036 |
| | $32^3$ | 0.0185 | 34.61 | 9,082 |
| | $54^3$ | 0.0163 | 159.20 | 22,246 |
| SPINN | $16^3$ | 0.0193 | 1.55 | 762 |
| | $32^3$ | 0.0060 | 1.71 | 762 |
| | $64^3$ | 0.0045 | 1.82 | 762 |
| | $128^3$ | 0.0040 | 1.85 | 890 |
| | $256^3$ | 0.0039 | 3.98 | 1,658 |
| SPINN + modified MLP | $16^3$ | 0.0062 | 2.20 | 764 |
| | $32^3$ | 0.0020 | 2.41 | 764 |
| | $64^3$ | 0.0013 | 2.57 | 764 |
| | $128^3$ | **0.0008** | 2.79 | 892 |
| | $256^3$ | 0.0009 | 5.61 | 1,660 |

Table 6: Full results of the **(3+1)-d Klein-Gordon equation**. $N_c$ is the number of collocation points.

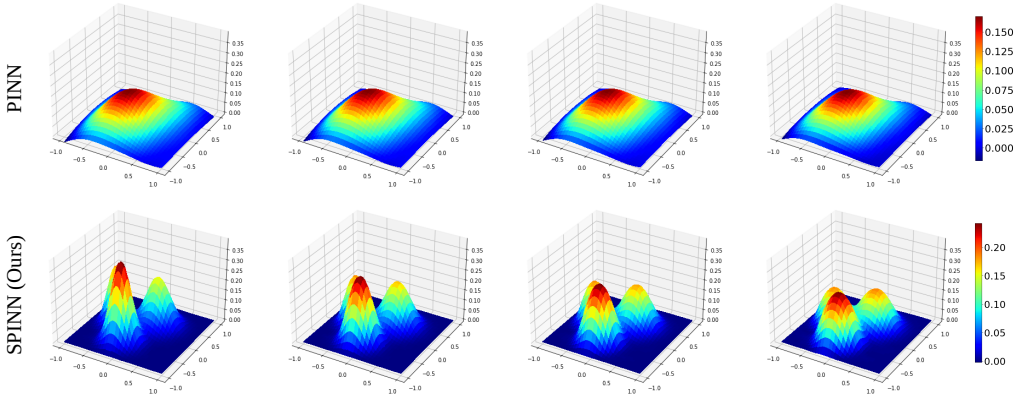
| model | $N_c$ | relative $L_2$ error | runtime (ms/iter.) | memory (MB) |
|---|---|---|---|---|
| PINN | $16^4$ | 0.0129 | 43.51 | 5,246 |
| | $23^4$ | 0.0121 | 154.24 | 22,244 |
| PINN + modified MLP | $16^4$ | 0.0061 | 100.06 | 17,534 |
| | $18^4$ | 0.0059 | 174.00 | 22,246 |
| SPINN | $16^4$ | 0.0122 | 2.45 | 890 |
| | $32^4$ | 0.0095 | 2.98 | 892 |
| | $64^4$ | 0.0093 | 9.22 | 2,172 |
| SPINN + modified MLP | $16^4$ | 0.0064 | 3.48 | 892 |
| | $32^4$ | 0.0022 | 3.66 | 892 |
| | $64^4$ | **0.0012** | 10.96 | 2,172 |



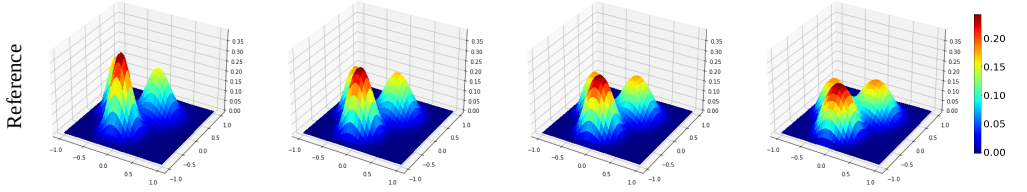

(a) Training time: 5 sec




(b) Training time: 30 sec

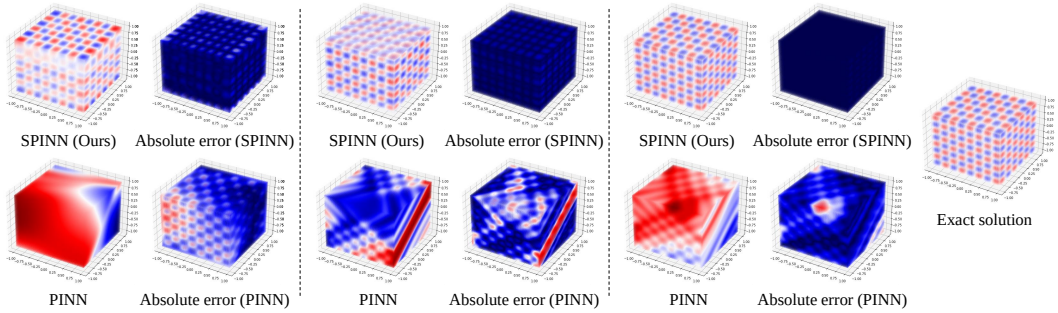



(c) Training time: 1 min




$t = 0.0$ $t = 0.25$ $t = 0.5$ $t = 1.0$

Figure 1: Visualized solution of **nonlinear diffusion equation** obtained by the baseline PINN and SPINN, both trained on $64^3$ collocation points.




(a) Training time: 5 sec (b) Training time: 30 sec (c) Training time: 1 min

Figure 2: Visualized solution of **Helmholtz equation** obtained by the baseline PINN and SPINN, both trained on $64^3$ collocation points.

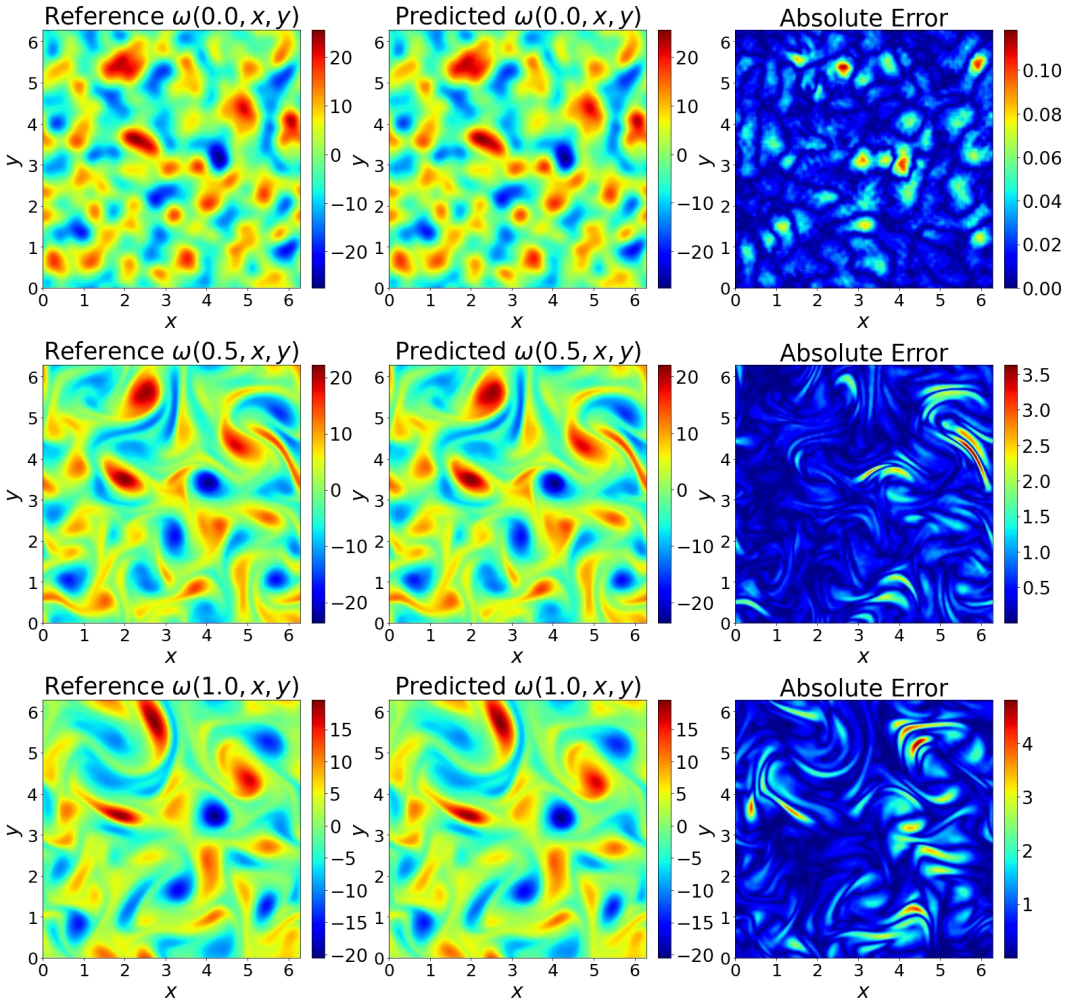
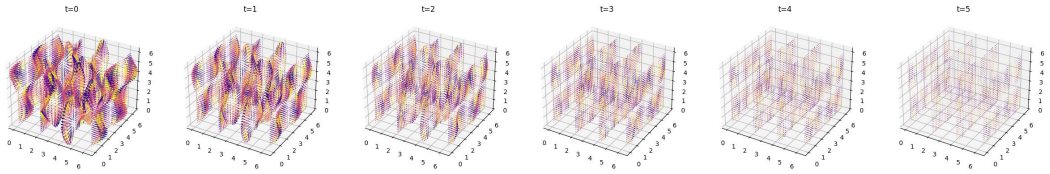
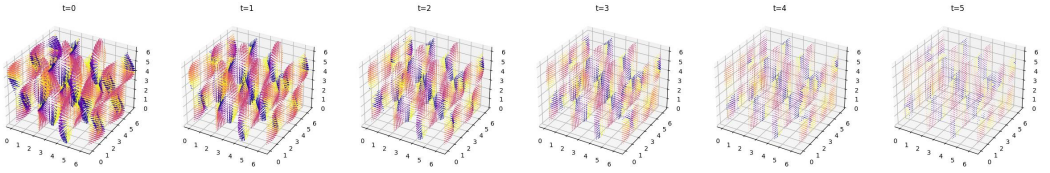


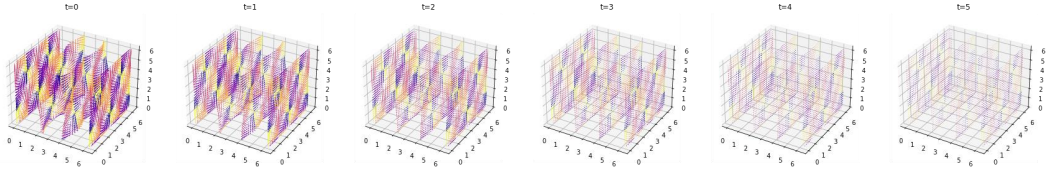

Figure 3: Visualized vorticity maps of **(2+1)-d Navier-Stokes equation** experiment predicted by SPINN. Three snapshots at timestamps $t = 0.0, 0.5, 1.0$ are presented.



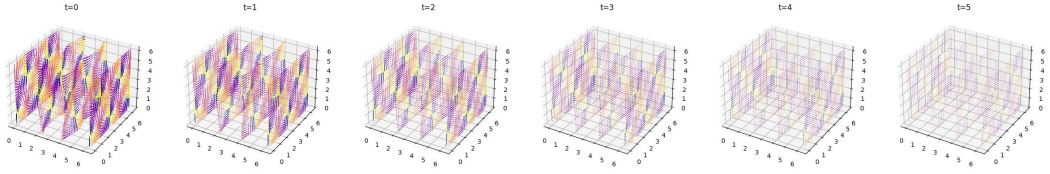

(a) Exact solution (velocity)




(b) Prediction (velocity)




(c) Exact solution (vorticity)




(d) Prediction (vorticity)

Figure 4: Visualized solution of **(3+1)-d Navier-Stokes equation** obtained by SPINN, trained on $32^4$ collocation points. Each arrow is colored by its zenith angle.

# References


[1] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[2] Suchuan Dong and Naxian Ni. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *Journal of Computational Physics*, 435:110242, 2021.

[3] Gerald B Folland. *A course in abstract harmonic analysis*, volume 29. CRC press, 2016.

[4] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[5] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[7] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

[8] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

[9] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.

[10] Xuhui Meng, Zhen Li, Dongkun Zhang, and George Em Karniadakis. Ppinn: Parareal physics-informed neural network for time-dependent pdes. *Computer Methods in Applied Mechanics and Engineering*, 370:113250, 2020.

[11] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[12] Geoffrey Ingram Taylor and Albert Edward Green. Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 158(895):499–521, 1937.

[13] Colby L Wight and Jia Zhao. Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks. *arXiv preprint arXiv:2007.04542*, 2020.