

Supplemental Information

A Baselines and network architectures

This section details the neural network architectures we employ in our experiments, as well as implementation details of the baselines to which we compare the reparameterized optimization.

Network architectures. We deliberately use generic off-the-shelf neural network architectures. In all performed experiments, the solution space consists of a finite number of scalars while the initial and final simulation states often involve spatial data, i.e. grids. For grid data, we use convolutional layers on multiple resolution levels while fully-connected layers process non-grid data. Applying this approach to our problems results in the following scenarios:

- **Grid to scalars (G2S).** We use a standard architecture for classification that largely follows by VGG [SZ14]. It consists of multiple convolutional blocks followed by fully-connected layers. Each convolutional block consists of one or multiple convolutional layers with kernel size 3^d where d denotes the number of grid dimensions. A batch normalization, activation, and max pooling layer follows each convolutional layer, reducing the resolution by half at the end of each block. The result is passed to a multilayer perceptron (MLP) which alternates linear, activation, and batch normalization layers before the result is outputted by a final linear layer.
- **Scalars to scalars (S2S).** When no grid data is involved, we simply use MLPs [GBCB16] to map inputs to outputs, optionally with positional encoding at the inputs. The MLP consists of multiple linear layers and activation layers but we do not use batch normalization layers since our networks are relatively shallow with no more than three hidden layers.
- **Grid to grid (G2G).** This case is only needed for the surrogate network required by the neural adjoint method. Here, we use the popular U-Net [RFB15] architecture with residual connections. Multiple convolutional blocks progressively decrease the spatial resolution, followed by upsampling convolutional blocks. The downsampling blocks match the ones described above. The upsampling blocks linearly interpolate the result to double the resolution before concatenating the corresponding processed input of the same resolution for the residual connections.

The specific hyperparameter values used for these generic architectures are given in the corresponding experimental details sections.

BFGS. We use the BFGS Implementation from SciPy [VGO⁺20] which runs the optimizer-internal computations on the CPU. All loss and gradient evaluations are bundled and dispatched to the GPU to be processed in parallel.

Neural adjoint. The neural adjoint method [RPM20] employs a neural network $\tilde{\mathcal{N}}$ to act as a surrogate for F . $\tilde{\mathcal{N}}$ is then used in place of F in an iterative optimization. Since $\tilde{\mathcal{N}}$ cannot be expected to produce accurate results outside of the region covered by its training data, a boundary loss term is added to the optimization to prevent the optimizer from leaving that region [RPM20]. We formulate this boundary loss in a differentiable manner to make it compatible with higher-order optimizers. First, we determine the minimum ξ_{\min}^j and the maximum ξ_{\max}^j value in the training set for each parameter j . Then, we formulate the boundary loss as

$$B(\xi) = \sum_j \text{SoftPlus}_\gamma \left(\frac{\max(\xi^j - \xi_{\max}^j, \xi_{\min}^j - \xi^j)}{\xi_{\max}^j - \xi_{\min}^j} \right),$$

where $\text{SoftPlus}_\gamma(x) \equiv \frac{1}{\gamma} \log(1 + e^{\gamma x})$.

56 B Experimental details

57 This section lists additional details about the experiments showcased in the main paper. An overview
58 of the symbols introduced with the experiments is given in Tab. 1.

Table 1: Physical quantities corresponding to the abstract symbols used in Eqs. 1 and 2 for each experiment.

Experiment	ξ	x	y
Wave packet fit	t_0	$\epsilon(t)$	$u(t)$
Billiards	\vec{v}_0	Initial ball positions	Final ball positions
Kuramoto–Sivashinsky	α, β	$u(x) _{t=0}$	$u(x) _{t=25}$
Incompr. Navier-Stokes	x_0, \vec{v}_0	$u(x, y) _{t=0}$	$u(x, y) _{t=56}$

59 **Software and hardware.** We used PyTorch [PGM⁺19] and Φ_{Flow} [HKT20] to run our experiments.
60 The full source code is part of the supplemental material. The first three experiments were run on a
61 GeForce RTX 3090. Due to memory requirements, the fluid experiment was run partly on a Quadro
62 RTX 8000 which allowed 128 simulations to be held in GPU memory.

63 **Hyperparameter selection.** For each network, we select one of the three generic architectures
64 listed above. Our main objective in choosing the values of the hyperparameters, such as the number
65 of layers and layer width, is keeping the total number of parameters large enough to fit the problem
66 easily but low enough to train the network quickly. The only hyperparameter which we tune is the
67 Adam [KB15] learning rate η . We start with $\eta = 0.01$ and progressively reduce it by a factor of 10
68 until the loss decreases during the optimization. The exact hyperparameter values are given in the
69 corresponding experiment section.

70 **Refinement.** We apply BFGS refinement as a second stage to reparameterized optimization, super-
71 vised training, and the neural adjoint method. In all cases, we run a standard BFGS optimization on
72 the actual \mathcal{L} , so the gradients are backpropagated through F . For reparameterized optimization and
73 the neural adjoint method, we use the parameter estimate with the lowest recorded loss value. As
74 supervised training makes use of pre-trained network, we use the final parameter estimate ξ_i as an
75 initial guess. The full evolution of the parameter estimates over the course of training is shown for all
76 experiments below.

77 **Results.** Complementary to Tab. 2 of the main paper, Tab. 2 gives an overview of how many exam-
78 ples were improved by the various neural-network-based approaches *without* refinement. Learning
79 curves, loss and improvement statistics, as well as example parameter trajectories are shown in the
80 following subsections.

Table 2: Fraction of inverse problems for which neural-network-based methods *without* refinement find better or equal solutions than BFGS. Mean over multiple seeds and all n shown in subfigures (d) of the main paper.

Experiment	Reparameterized		Supervised		Neural Adjoint	
	Better	Equal	Better	Equal	Better	Equal
Wave packet fit	80.5%	0.9%	55.9%	2.0%	28.2%	0.8%
Billiards	44.3%	9.0%	14.6%	19.9%	1.6%	29.3%
Kuramoto–Sivashinsky	42.8%	0.0%	14.4%	0.0%	6.4%	0.0%
Incompr. Navier-Stokes	62.5%	0.0%	23.5%	0.0%	1.1%	0.0%

81 B.1 Wave packet localization

82 For the wave packet experiment, we first determine the true position t_0 of the wave packet by sampling
83 random values from a uniform distribution between $t_0 \in [26, 230]$. Noise $\epsilon(t)$ is sampled from a
84 normal distribution with standard deviation $\sigma = 0.1$ for every $t = 1, \dots, 256$ and superimposed on

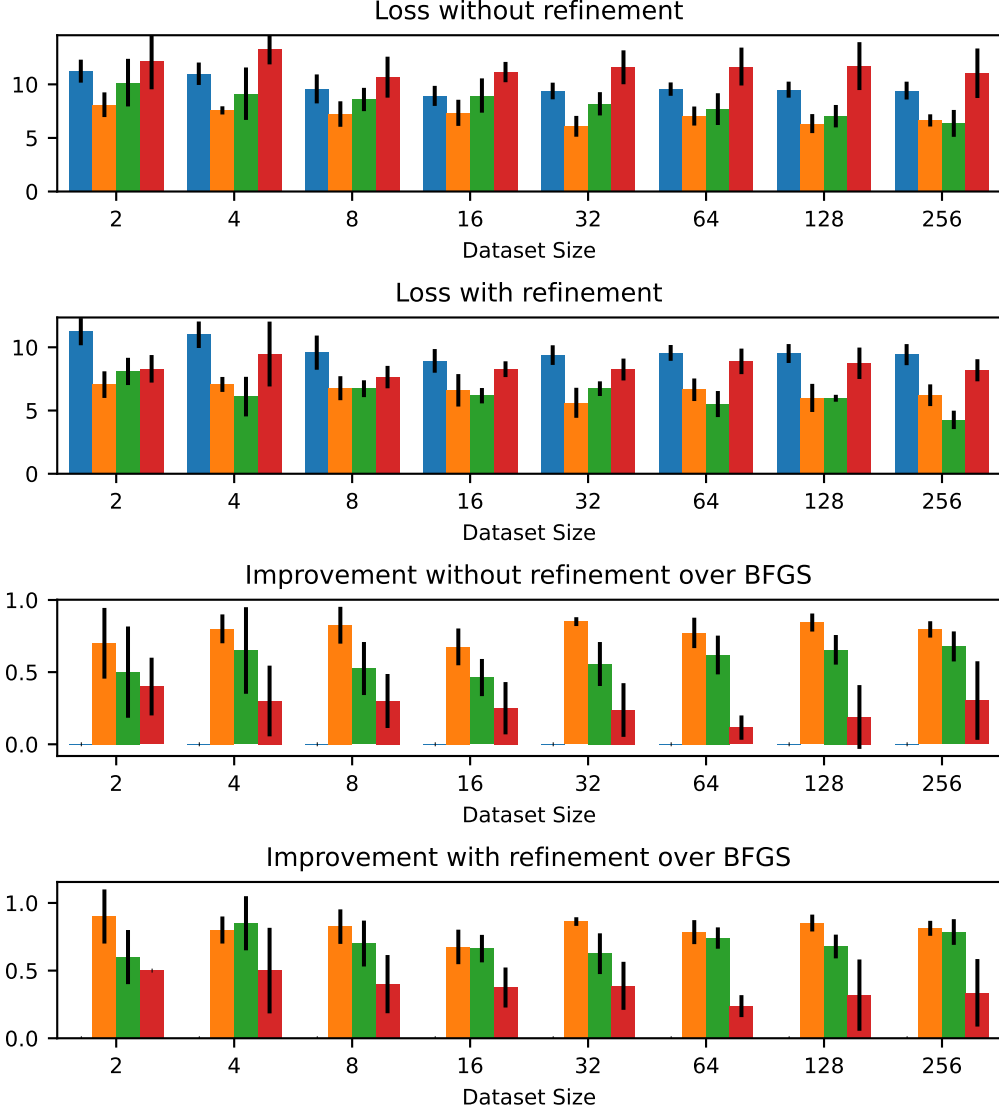


Figure 1: Loss and improvement over BFGS before and after refinement for the wave packet experiment. Colors match figures from the main paper (blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint).

85 the signal, as described in the main text. The noise pattern is only used to generate the reference data
86 and is not available to the optimizers. We run this experiment five times with varying initialization
87 seeds for both networks and data sets.

88 **Networks.** The surrogate network, required by the neural adjoint method, takes t_0 and $\epsilon(t)$ as input
89 and outputs an approximation of $u(t)$. Since $\epsilon(t)$ and $u(t)$ are one-dimensional grids, we employ
90 the G2G architecture with two input feature maps of size 256 and one output feature map, totaling
91 13,073 parameters. The reparameterization network maps the grid $u(t)$ to the estimated scalar t_0 .
92 Consequently, we use the G2S architecture described in section A. We use five blocks with one
93 convolutional layer with 16 feature maps each, reducing the resolution from 256 to 8. The MLP part
94 consists of two hidden fully-connected layers with sizes 64 and 32. In total, this network contains
95 13,925 parameters. All networks are trained using Adam with a learning rate of $\eta = 0.001$.

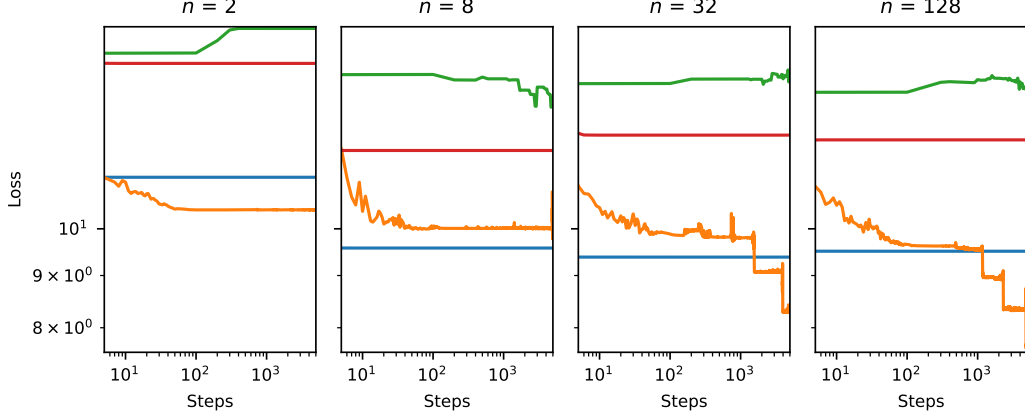


Figure 2: Optimization curves for different data set sizes of the wave packet experiment before refinement. Curves show the mean over 5 network and data set initialization seeds. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint.

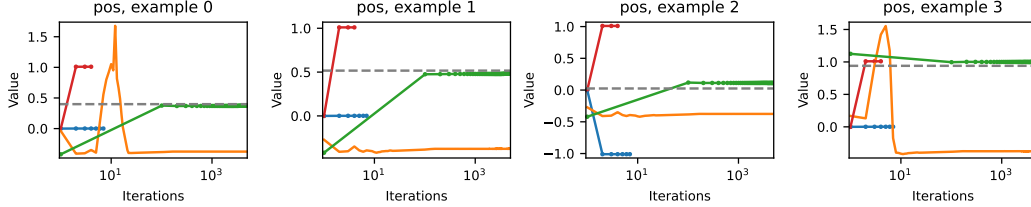


Figure 3: Example parameter evolution during optimization of the wave packet experiment with $n = 128$. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint. The dashed gray lines indicate the reference solution from which the example was generated. BFGS-based optimization curves stop when all examples have fully converged to an optimum.

96 **Additional results.** Fig. 1 shows the resulting loss and improvement over BFGS, both before and
 97 after refinement. The learning curves for four data set sizes n are shown in Fig. 2, and the parameter
 98 evolution of four examples during optimization are shown in Fig. 3.

99 B.2 Billiards

100 For the billiards experiment, we set up a rigid body simulation of spherical balls with radius $r = 0.2$
 101 moving in the x-y-plane. In each step, the simulator analytically integrates the evolution until the
 102 time of the subsequent collision, allowing us to simulate the dynamics at little computational cost.
 103 Collisions use a fixed elasticity of 0.8 and preserve momentum. Friction is assumed to be proportional
 104 to the speed of the balls. The simulation stops once no more collisions take place and integrates up
 105 to $t = \infty$ to let all balls come to rest. we sample initial states by randomly placing the second ball
 106 between $(1, 0)$ and $(1, 1)$ while keeping the target fixed at $(2, 0.5)$. The cue ball, located at $x = 0$
 107 is given a starting initial velocity of $\vec{v}_0^{\text{start}} = (1, 0)$ so it will collide with the second ball in many
 108 cases by default. Starting the optimization with $\vec{v}_0^{\text{start}} = 0$ would yield $\nabla L_i = 0 \forall i$ and prevent any
 109 optimization using F . However, in none of these examples does the ball exactly reach the target.
 110 As the distribution of actual solutions \vec{v}_0 is unknown, the generated training sets for supervised and
 111 surrogate network training must rely on this broader data set, making learning more difficult.

112 **Networks.** The surrogate neural network is given a value for the initial velocity \vec{v}_0 and the balls'
 113 positions as input, and it outputs the predicted final position of the ball. We use positional encoding
 114 for the input using sine, cosine functions with four equidistant frequencies. The surrogate network
 115 follows the S2S architecture from section A. It is an MLP with three hidden layers containing 128
 116 neurons, each, and comprises 37,506 parameters in total. The reparameterization network predicts

117 \vec{v}_0 based on the initial and final ball positions, and we use the same network architecture as for the
 118 surrogate network. All networks are trained using Adam. For the reparameterized optimization, we
 119 use a learning rate of $\eta = 10^{-4}$ while all other methods use $\eta = 0.001$.

120 **Additional results.** Fig. 4 shows the resulting loss and improvement over BFGS, both before and
 121 after refinement. The learning curves for four data set sizes n are shown in Fig. 5, and the parameter
 122 evolution of four examples during optimization are shown in Fig. 6.

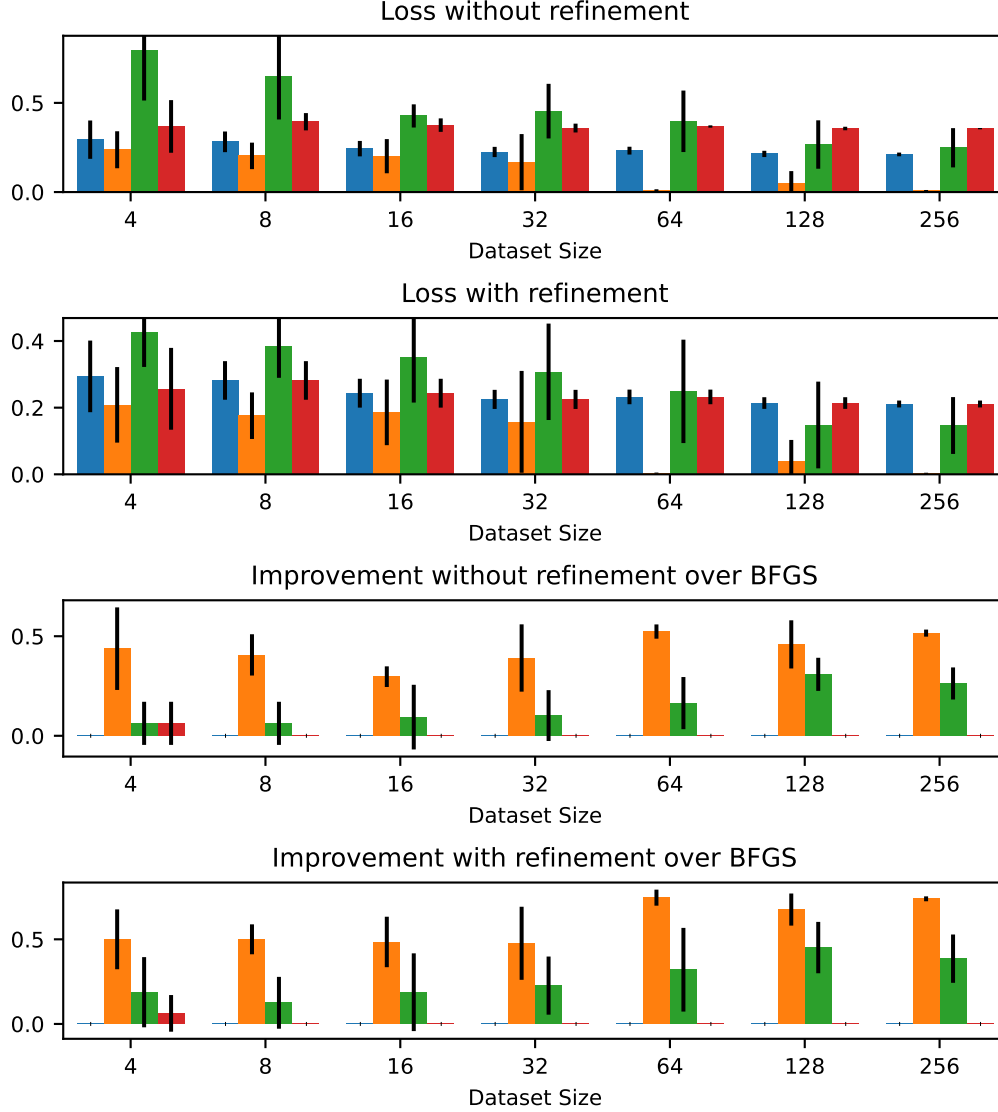


Figure 4: Loss and improvement over BFGS before and after refinement for the billiards experiment. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint.

123 B.3 Kuramoto–Sivashinsky equation

124 For this experiment, we set up a differentiable simulation of the Kuramoto–Sivashinsky (KS) equation
 125 in one dimension with a resolution of 128. We simulate the linear terms of KS equation in frequency
 126 space and use a Runge-Kutta-2 [PTVF07] scheme for the non-linear term. The initial state is sampled
 127 from random noise in frequency space with smoothing applied to suppress high frequencies. In each

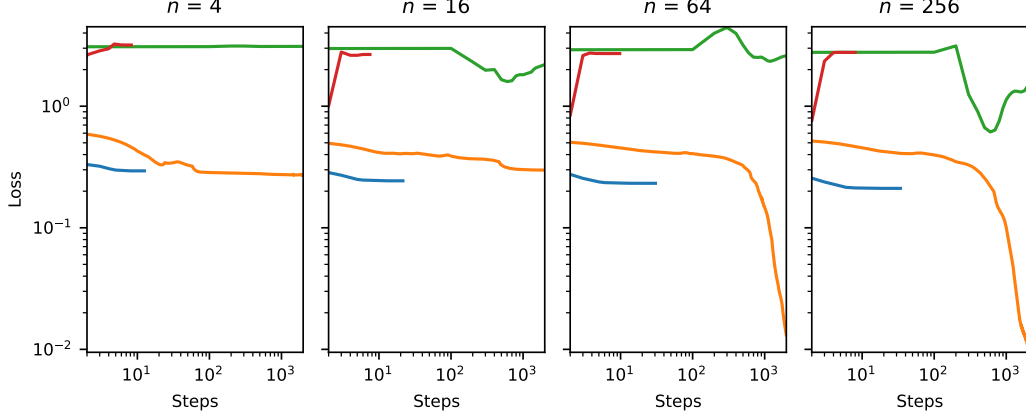


Figure 5: Optimization curves for different data set sizes of the billiards experiment before refinement. Envelopes show the standard deviation over 4 network and data set initialization seeds. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint. BFGS-based optimization curves stop when all examples have fully converged to an optimum.

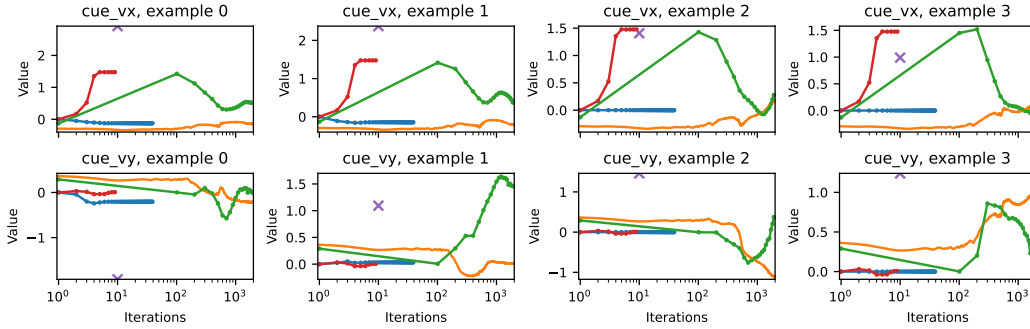


Figure 6: Example parameter evolution during optimization of the billiards experiment with $n = 128$. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint. The purple crosses indicate the reference from which the example was generated and is not a valid solution in this experiment. BFGS-based optimization curves stop when all examples have fully converged to an optimum.

simulation step, we add a forcing of the form $G(x) = 0.1 \cos(x) - 0.01 \cos(x/16) \cdot (1 - 2 \sin(x/16))$ which is controlled by the parameter α as described in the main text.

Networks. The surrogate network maps the initial state $u(x, t = 0)$ and parameters α, β to the final state $u(x, t = 25)$. Since u is sampled on a grid, we use the G2G architecture from section A with three input and one output feature map, operating on four resolution levels. The reparameterization network maps $u(x, t = 0)$ and $u(x, t = 25)$ to α, β and we employ the G2S architecture with four convolutional layers of widths 32, 32, 64, 64, followed by two hidden fully-connected layers with 64 neurons each. We train both networks using Adam with a learning rate of $\eta = 0.001$ for 1000 iterations.

Additional results. Fig. 7 shows the resulting loss and improvement over BFGS, both before and after refinement. The learning curves for four data set sizes n are shown in Fig. 8, and the parameter evolution of four examples during optimization are shown in Fig. 9.

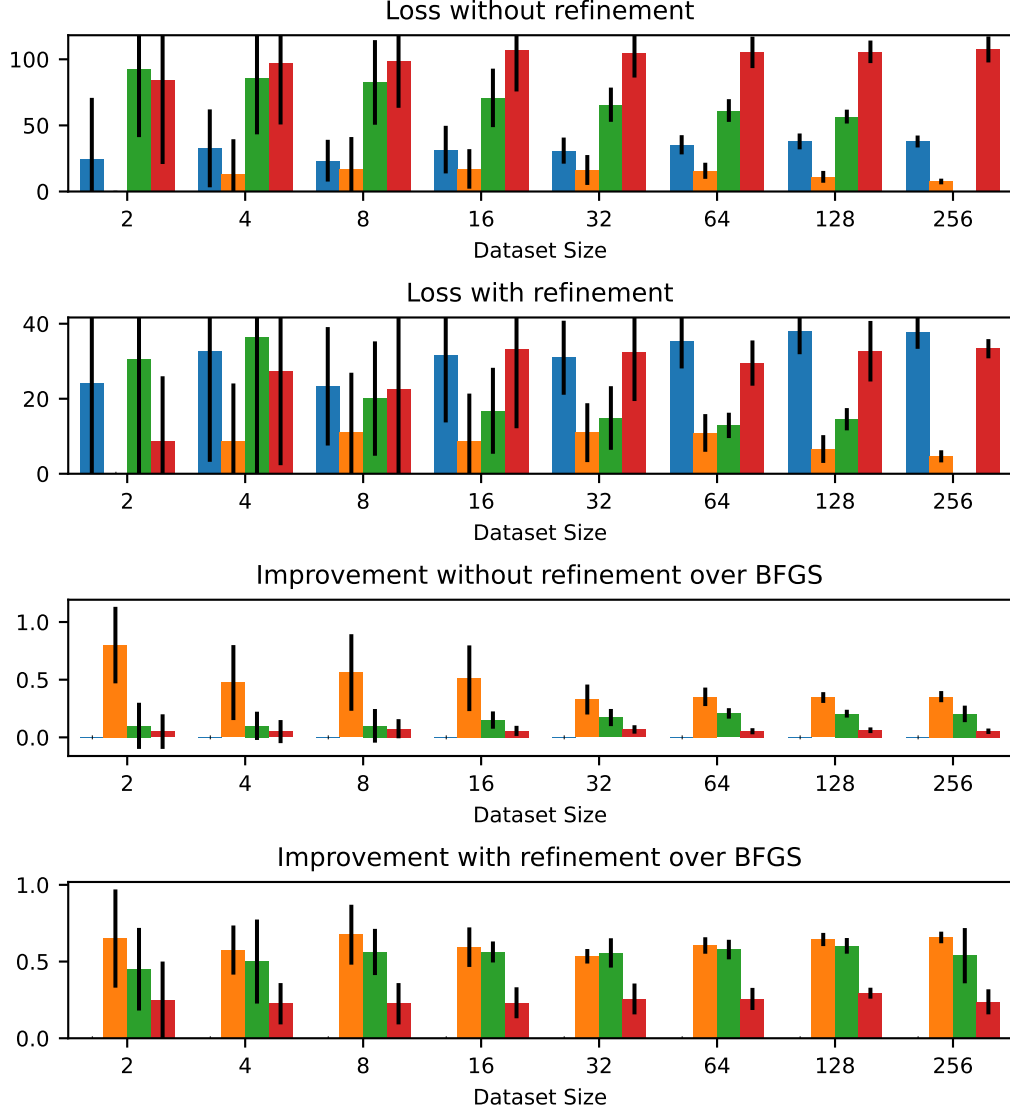


Figure 7: Loss and improvement over BFGS before and after refinement for the Kuramoto-Sivashinsky experiment. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint.

140 B.4 Incompressible Navier-Stokes

141 We simulate an incompressible two-dimensional fluid in a 100 by 100 box with a resolution of 64 by
 142 64, employing a direct numerical solver for incompressible fluids from Φ_{Flow} [HKT20]. Specifically,
 143 we use the marker-in-cell (MAC) method [HW65, Har72] which guarantees stable simulations even
 144 for large velocities or time increments. The velocity vectors are sampled in staggered form at the face
 145 centers of grid cells while the marker density is sampled at the cell centers. The initial velocity v_0 is
 146 specified at cell centers and resampled to a staggered grid for the simulation. Our simulation employs
 147 a second-order advection scheme [SFK⁺08] to transport the marker and the velocity vectors. We do
 148 not simulate explicit diffusion as the numerical diffusion introduced by the advection scheme on this
 149 resolution is sufficient for our purposes. Incompressibility is achieved via Helmholtz decomposition
 150 of the velocity field using a conjugate gradient solve.

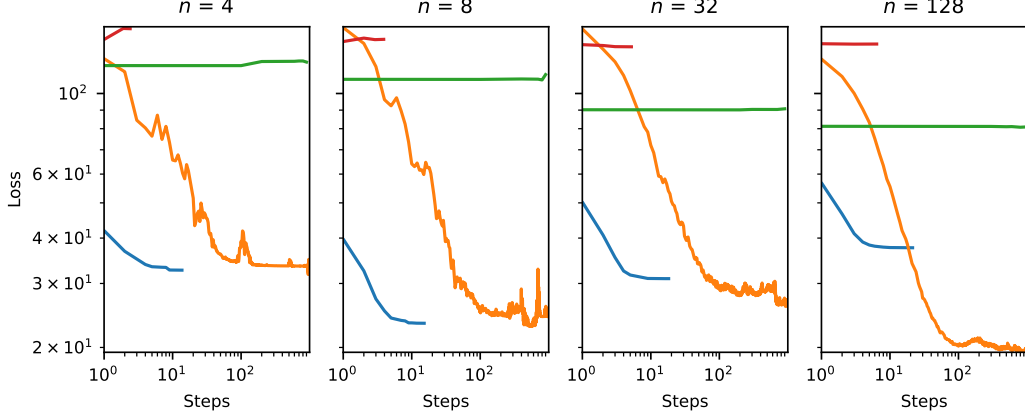


Figure 8: Optimization curves for different data set sizes of the Kuramoto–Sivashinsky experiment before refinement. Envelopes show the standard deviation over 10 network and data set initialization seeds. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint. BFGS-based optimization curves stop when all examples have fully converged to an optimum.

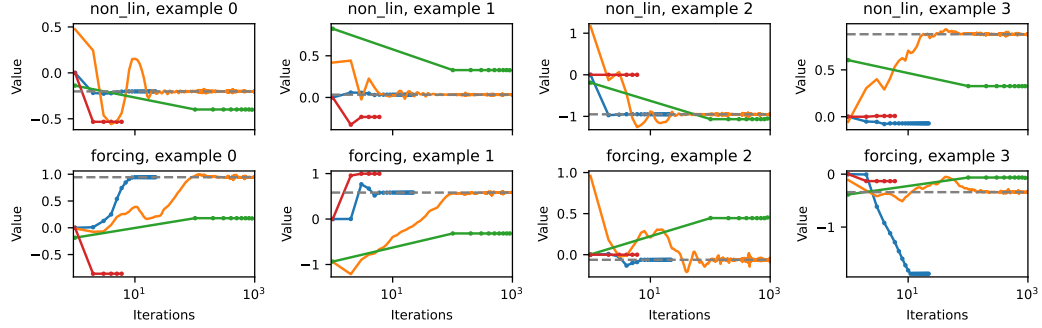


Figure 9: Example parameter evolution during optimization of the Kuramoto–Sivashinsky experiment with $n = 128$. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint. The dashed gray lines indicate the reference solution from which the example was generated. BFGS-based optimization curves stop when all examples have fully converged to an optimum.

151 We initialize the whole domain with a velocity field sampled from random noise in frequency space,
 152 resulting in eddies of various sizes. The initial velocity values have a mean of zero and a standard
 153 deviation of 0.5. Then, ground truth values for x_0 and \vec{v}_0 are sampled from uniform distributions with
 154 $\vec{v}_0^y \geq 0$ never pointing downward. These values are used to initialize a spherical force or wind blast
 155 near the bottom of the domain that moves upwards during the simulation and induces flow around all
 156 obstacles from the pressure computation. The velocity is only observable in the domain’s upper half,
 157 and all optimizers assume a zero-initialization in the unobservable bottom half.

158 **Networks.** The surrogate network approximates the final state $u(x, y \geq 50, t = 56)$ in the upper
 159 half of the domain from the initial state $u(x, y, t = 0)$ and the parameters x_0, \vec{v}_0 . As before, we
 160 implement this as G2G (section A) with five input and two output feature maps, totaling 38.290
 161 parameters. The G2S reparameterization network comprises four convolutional layers with 16, 32,
 162 32, and 32 feature maps, respectively, followed by two fully-connected layers with 64 neurons each,
 163 resulting in 44.723 total parameters. Both networks are trained using Adam with a learning rate of
 164 $\eta = 0.001$.

165 **Additional results.** As noted in the main text, the high loss value of the reparameterized opti-
 166 mization is largely due to a fraction of examples with considerably higher loss than the average. A
 167 summary of the individual loss values for $n = 4, 128$ is given in Figs. 10 and 11, respectively. While

168 the neural adjoint method produces the highest loss values before refinement, these all get mapped to
 169 relatively small values during the refinement stage. Meanwhile, the reparameterized training finds
 170 better solutions without refinement since it uses feedback from F . However, that also means that
 171 the secondary BFGS optimization cannot improve the estimates by nearly as much since many are
 172 already close to a (local) minimum. This leaves a fraction of examples stranded on sub-optimal
 173 solutions that contribute significantly to the total loss, despite most problems finding better solutions
 174 than BFGS. Fig. 12 shows the resulting loss and improvement over BFGS, both before and after
 175 refinement. The learning curves for four data set sizes n are shown in Fig. 13, and the parameter
 176 evolution of four examples during optimization are shown in Fig. 14.

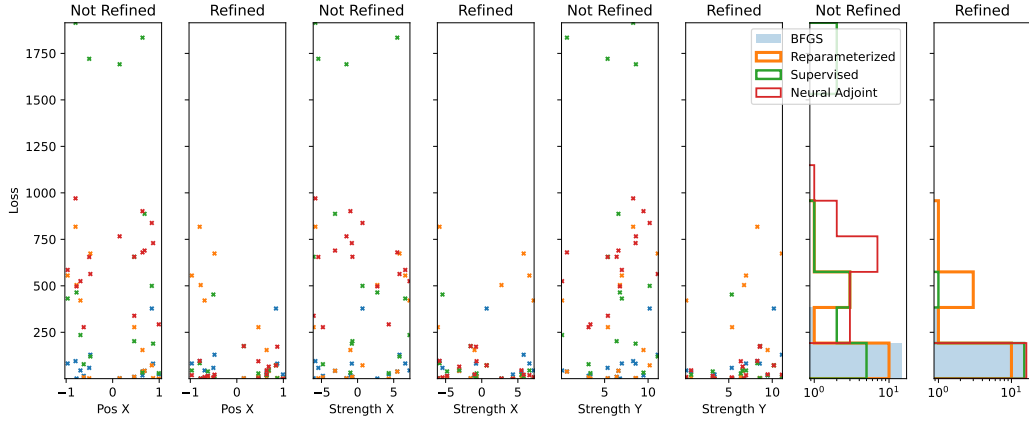


Figure 10: Distribution of loss values for $n = 4$ in the Navier-Stokes experiment, with and without refinement. The first six plots show the loss distribution along the ground truth value of one of the parameters to be optimized. The right plots show the margin distribution of loss values. The results of 4 network and data set initialization seeds are accumulated.

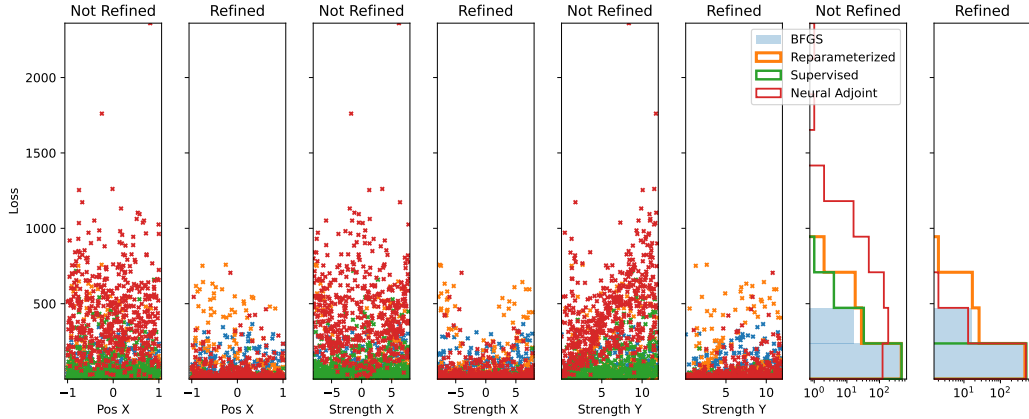


Figure 11: Distribution of loss values for $n = 128$ in the Navier-Stokes experiment, with and without refinement. The first six plots show the loss distribution along the ground truth value of one of the parameters to be optimized. The right plots show the margin distribution of loss values. The results of 4 network and data set initialization seeds are accumulated.

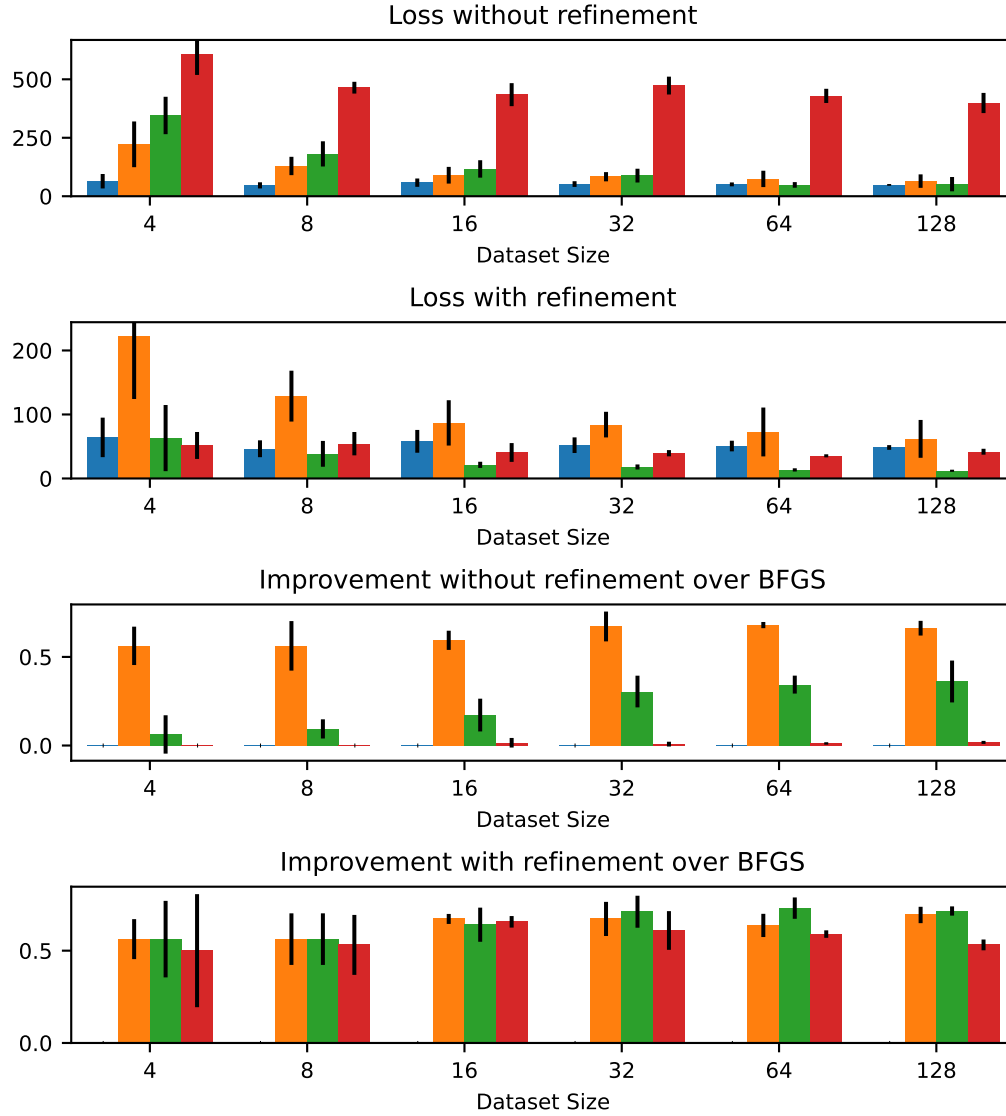


Figure 12: Loss and improvement over BFGS before and after refinement for the Navier-Stokes experiment. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint.

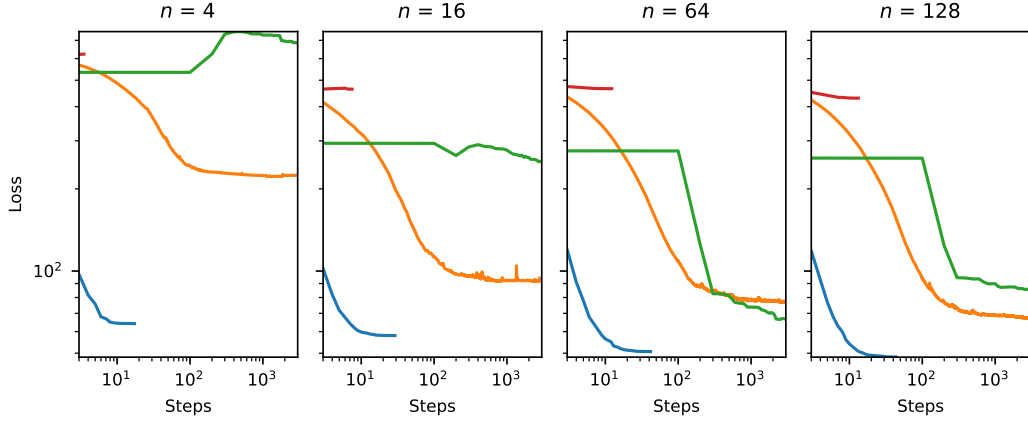


Figure 13: Optimization curves for different data set sizes of the Navier-Stokes experiment before refinement. Envelopes show the standard deviation over 4 network and data set initialization seeds. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint. BFGS-based optimization curves stop when all examples have fully converged to an optimum.

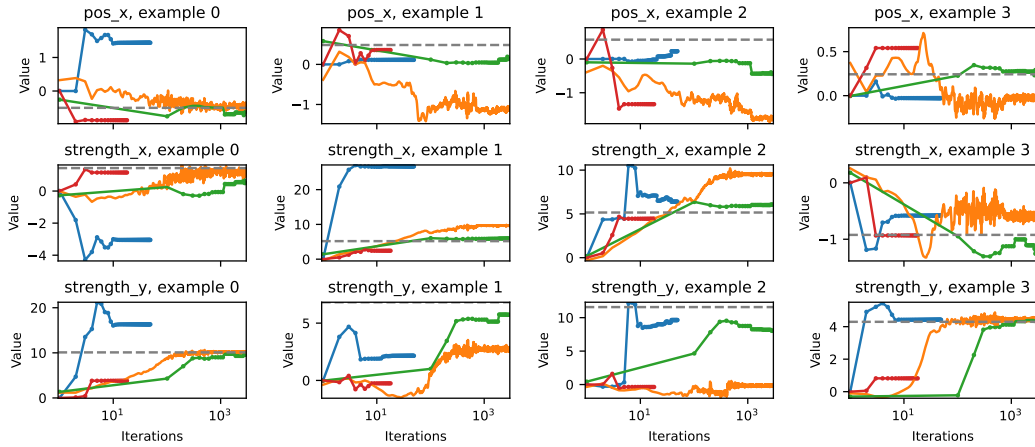


Figure 14: Example parameter evolution during optimization of the Navier-Stokes experiment with $n = 128$. Blue: BFGS, orange: reparameterized, green: supervised, red: neural adjoint. The dashed gray lines indicate the reference solution from which the example was generated. BFGS-based optimization curves stop when all examples have fully converged to an optimum.

References

- [GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. MIT press Cambridge, 2016.
- [Har72] FH Harlow. The marker-and-cell method. *Fluid Dyn. Numerical Methods*, 38, 1972.
- [HKT20] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. In *International Conference on Learning Representations (ICLR)*, 2020.
- [HW65] Francis H Harlow and J Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.
- [KB15] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
- [PGM⁺19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes*. Cambridge University Press, 3 edition, 2007.
- [RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [RPM20] Simiao Ren, Willie Padilla, and Jordan Malof. Benchmarking deep inverse models over time, and the neural-adjoint method. *Advances in Neural Information Processing Systems*, 33:38–48, 2020.
- [SFK⁺08] Andrew Selle, Ronald Fedkiw, ByungMoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2-3):350–371, June 2008.
- [SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [VGO⁺20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.