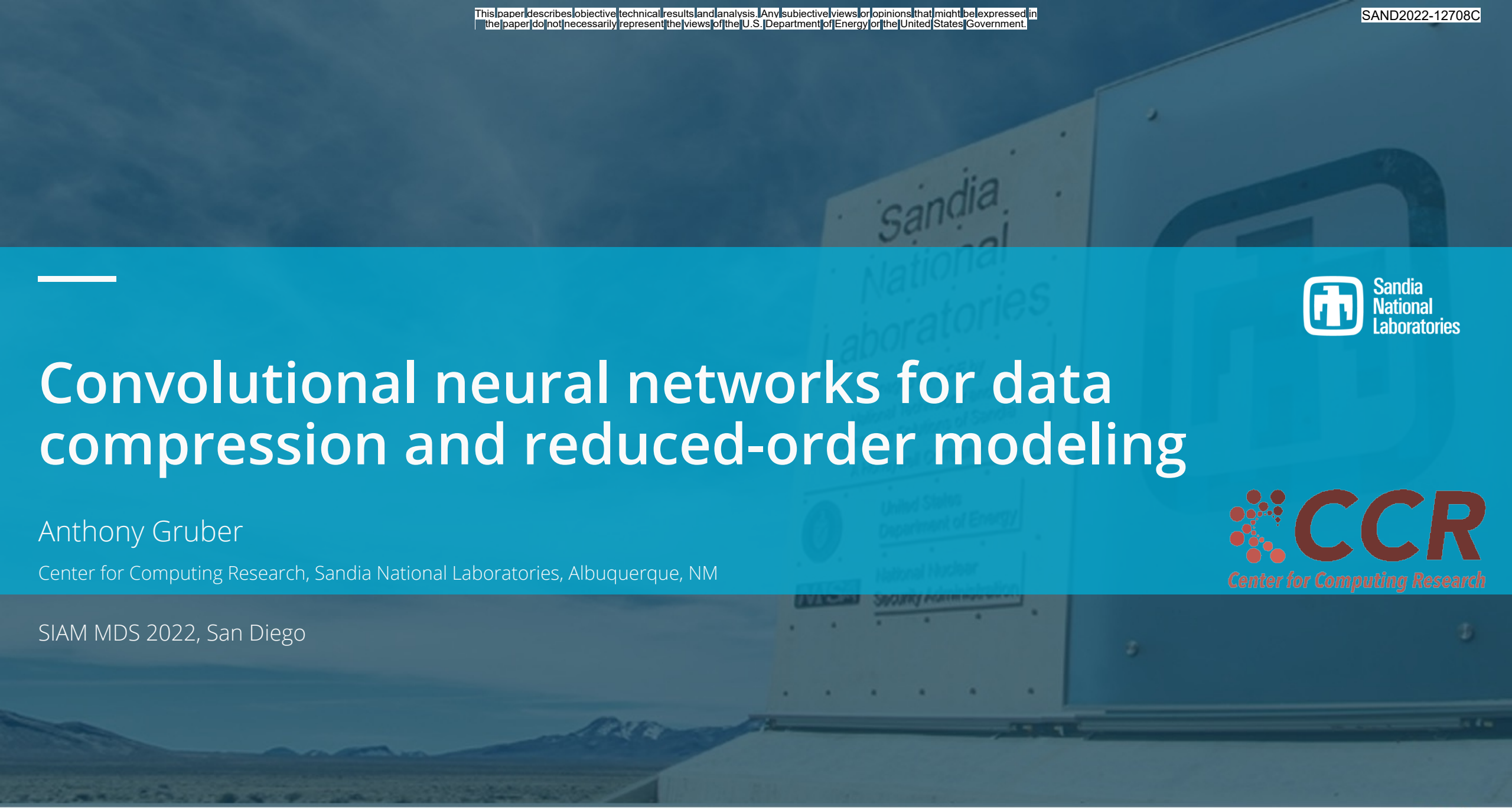
SAND2022-12708C



Convolutional neural networks for data compression and reduced-order modeling

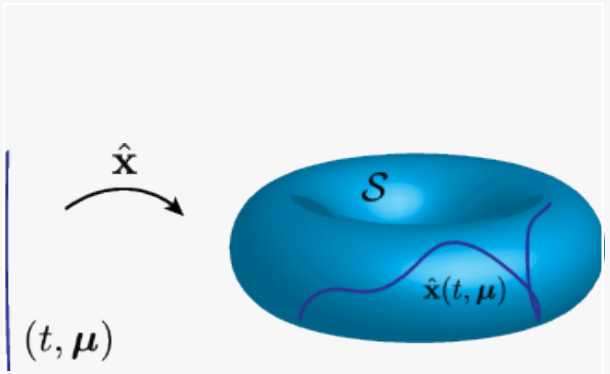
Anthony Gruber

Center for Computing Research, Sandia National Laboratories, Albuquerque, NM

SIAM MDS 2022, San Diego

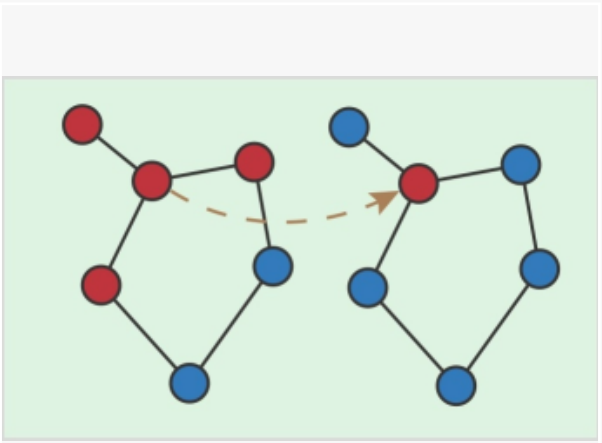
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline



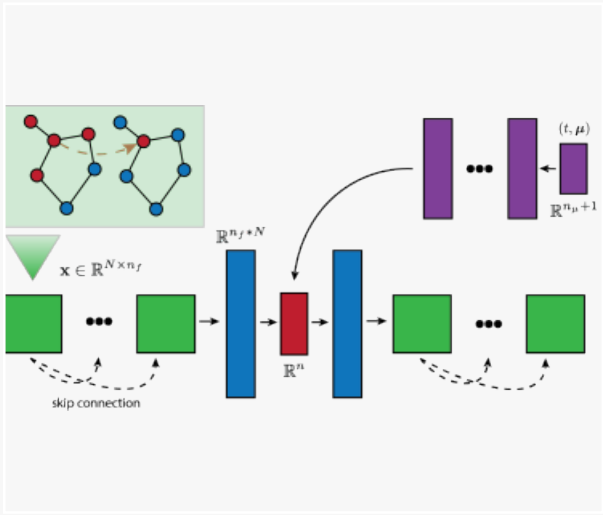
1

ROM Problem



2

Graph CNN Approach



3

Algorithm and Results

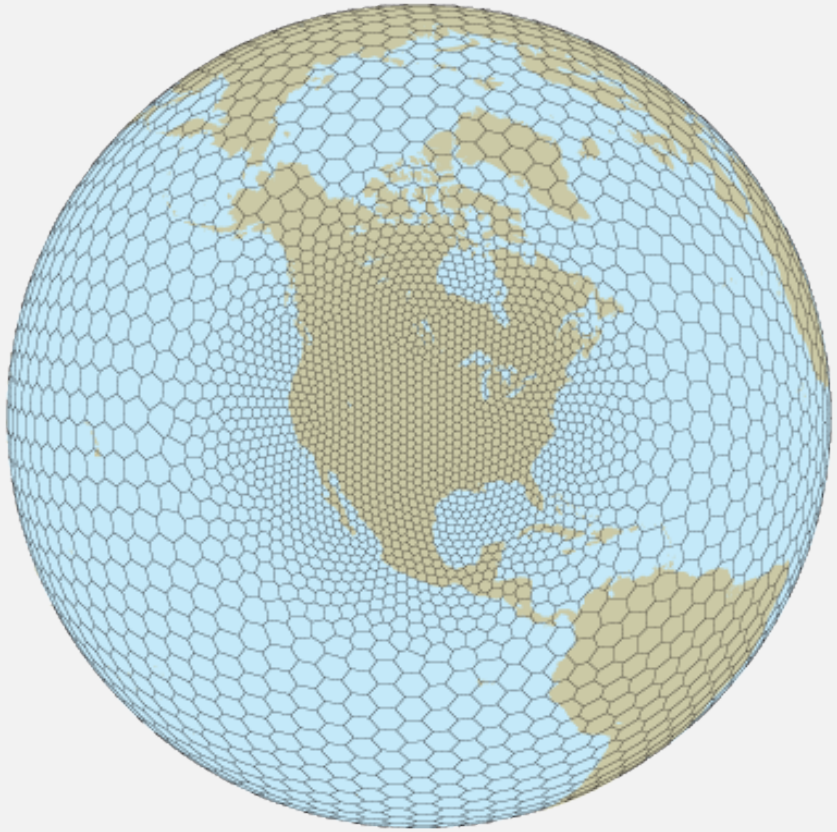
- Joint work with **Max Gunzburger (UT Austin)**, **Lili Ju (U of SC)**, and **Zhu Wang (U of SC)**
- See "A comparison of neural network architectures for data-driven reduced-order modeling", CMAME 2022.

Full-Order Model

- FOM: $\dot{\boldsymbol{x}}(t, \boldsymbol{\mu}) = \boldsymbol{f}(t, \boldsymbol{x}(t, \boldsymbol{\mu}), \boldsymbol{\mu}), \quad x(0, \boldsymbol{\mu}) = \boldsymbol{x}_0(\boldsymbol{\mu}).$
  - $\boldsymbol{\mu}$ is vector of parameters.
  - $\dim(\boldsymbol{x})$ large: 10^4 to 10^9 or more.
- Typically solved with time integrator e.g. Runge-Kutta.
  - Can be expensive. How to reduce cost?

Reduced-Order Models

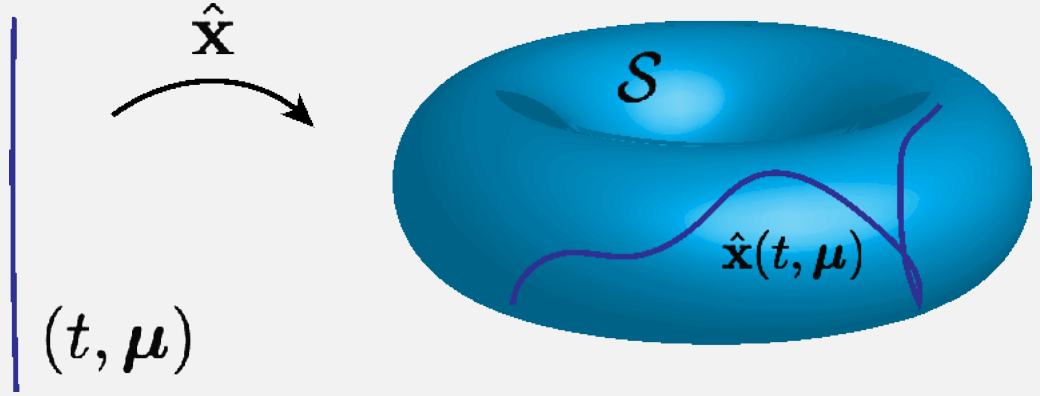
- High-fidelity PDE simulations are **expensive.**
  - semi-discretization blows up dimensionality.
- Good results possible without solving full PDE?
- Standard is to **encode -> solve -> decode.**
  - Linear: POD, RBM, etc.
  - Nonlinear: networks (FCNN and CNN)



https://mpas-dev.github.io/atmosphere/atmosphere.html

Idea Behind ROM

- Do we really need all 10^9 dimensions?
- No, if $(t, \boldsymbol{\mu}) \mapsto \boldsymbol{x}(t, \boldsymbol{\mu})$ is unique.
- $\mathcal{S} = \{\boldsymbol{x}(t, \boldsymbol{\mu}) \mid t \in [0, T],\ \boldsymbol{\mu} \in D\} \subset \mathbb{R}^N$, solution manifold.
- $(n_\mu + 1)$ dimensions enough for loss-less representation of $\mathcal{S}$.
- How can we recover $\mathcal{S}$ efficiently?



Idea Behind ROM

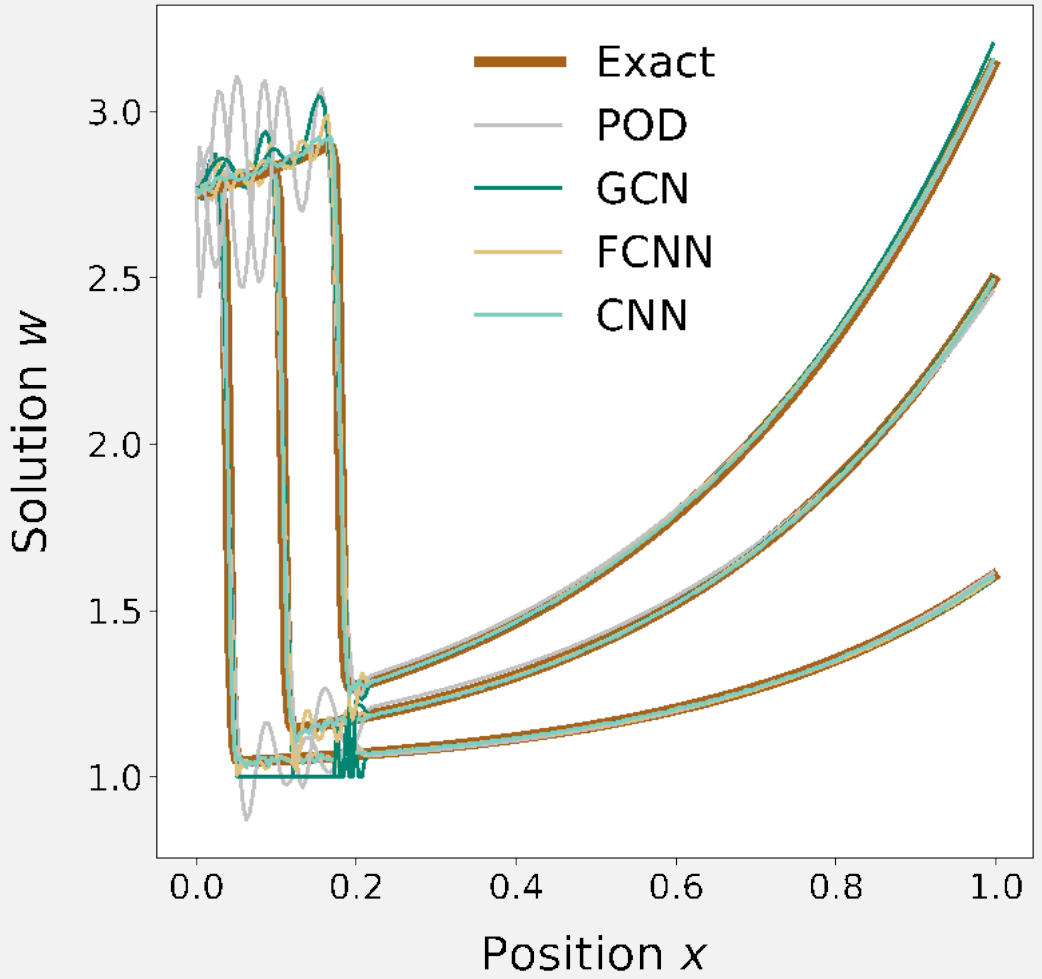
- Consider finding $\tilde{\boldsymbol{x}} = \boldsymbol{g} \circ \hat{\boldsymbol{x}}$ so that $\tilde{\boldsymbol{x}} \approx \boldsymbol{x}$.
  - $\boldsymbol{g}: \mathbb{R}^n \to \mathbb{R}^N$ a decoder function, $n \ll N$.
- Residual $\left\| \dot{\tilde{\boldsymbol{x}}} - \boldsymbol{f}(\tilde{\boldsymbol{x}}) \right\|^2$ is minimized when:
  - $\dot{\hat{\boldsymbol{x}}}(t, \boldsymbol{\mu}) = \boldsymbol{g}'(\hat{\boldsymbol{x}})^+ \boldsymbol{f}(t, \boldsymbol{g}(\hat{\boldsymbol{x}}), \boldsymbol{\mu}), \qquad \hat{\boldsymbol{x}}(0, \boldsymbol{\mu}) = \boldsymbol{h}\big(\boldsymbol{x}_0(\boldsymbol{\mu})\big).$
  - $\boldsymbol{h}: \mathbb{R}^N \to \mathbb{R}^n$ left inverse to $\boldsymbol{g}$.
- ODE of size $\boldsymbol{N}$ converted to ODE of size $\boldsymbol{n}$.
  - Hard part is determining/computing $\boldsymbol{g}$!

Example: Proper Orthogonal Decomposition (POD)

- Do PCA on solution snapshots $\boldsymbol{x}(t_j, \boldsymbol{\mu}_j),\ 1 \leq j \leq N_t$.
- Compute SVD $\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$.
  - First $\boldsymbol{n}$ columns of $\boldsymbol{U}$ (say $\boldsymbol{A}$) – reduced basis of POD modes.
  - $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ replaced with $\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}^{+}\boldsymbol{f}(\boldsymbol{A}\hat{\boldsymbol{x}})$.
- Totally linear procedure (good and bad).

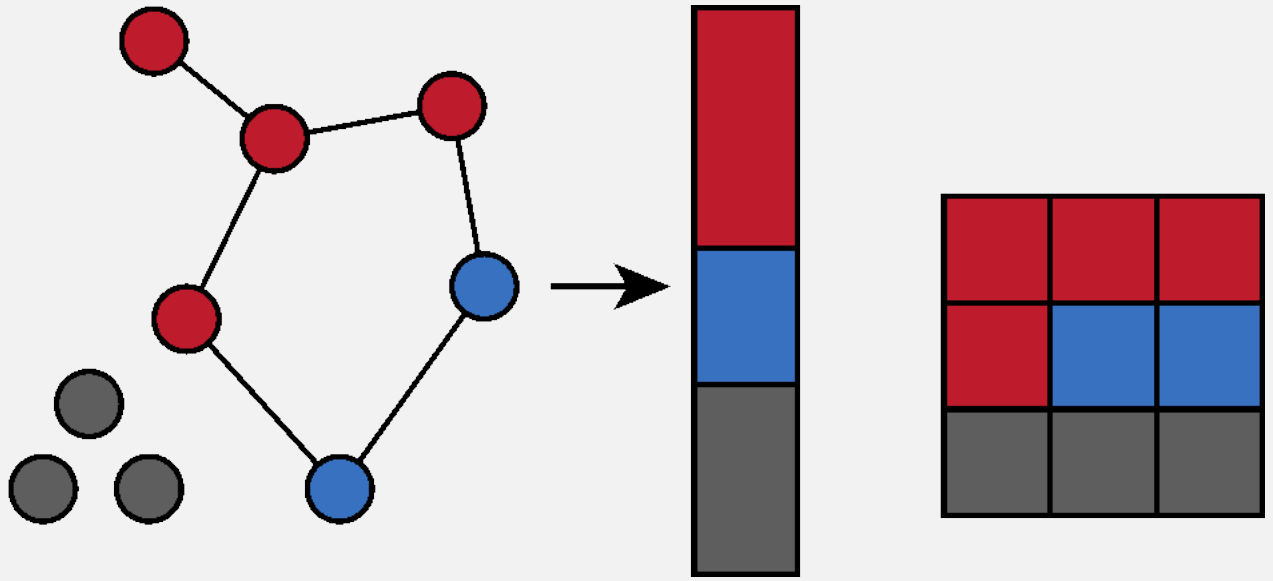
ROM Methods

- POD works until EWs of $\mathbf{\Sigma}$ decay slowly.
- "solved" by CNN autoencoder/decoder
  - Improved performance over POD**.
    - ** (In some cases)
- BUT slower and more difficult to train.
  - Also more memory consumptive!
- Now often used "by default".



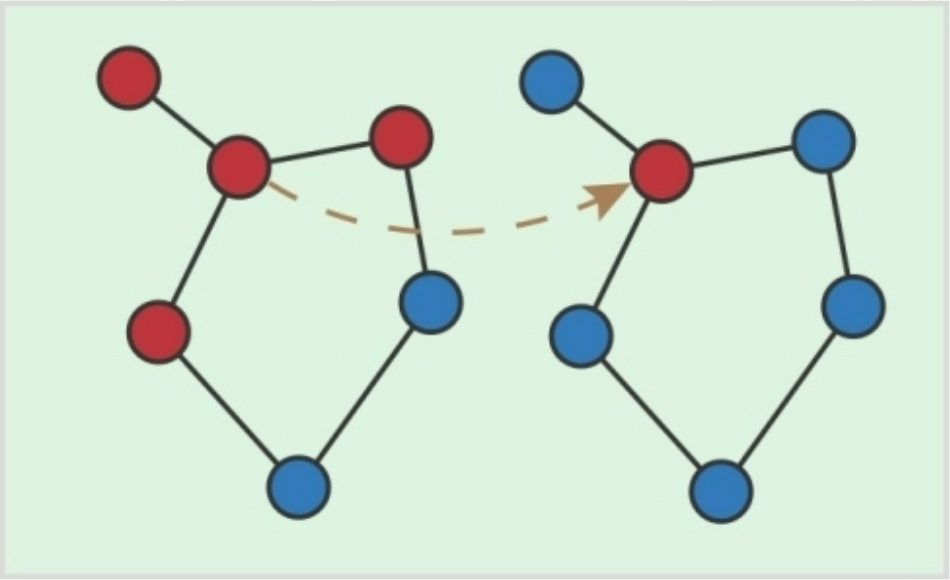
Disadvantages of CNN ROMs

- Standard CNN *not well suited* for irregular data.
- Standard practice: ignore the issue!
  - **Pad** inputs with fake nodes.
  - **Convolve** square-ified input.
  - **Reassemble** at end;
    fake nodes ignored!
- Works surprisingly well!
  - But, not very meaningful.



Graph Convolutional Networks

- Alternatively: *use a graph convolutional network (GCNN).*
- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ undirected graph; adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$.
- $\boldsymbol{D}$: degree matrix $d_{ii} = \sum_j a_{ij}$.
- Laplacian of $\mathcal{G}$: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$.
  - Columns of $\boldsymbol{U}$ are Fourier modes of $\mathcal{G}$.
  - Discrete FT/IFT: multiply by $\boldsymbol{U}$ / $\boldsymbol{U}^T$.

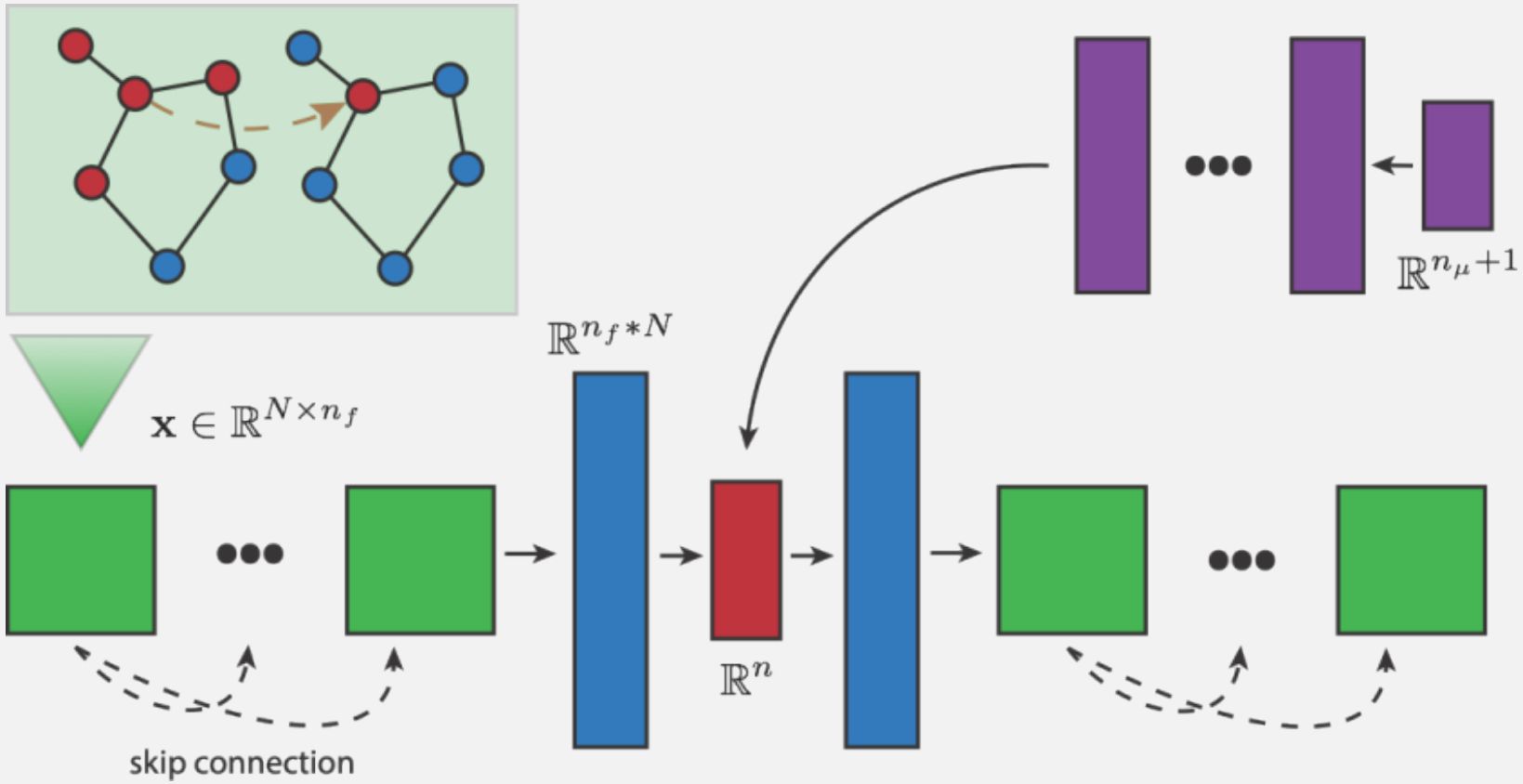


Graph Convolutional Networks

- $\widetilde{\boldsymbol{P}} = (\boldsymbol{D} + \boldsymbol{I})^{-1/2}(\boldsymbol{A} + \boldsymbol{I})(\boldsymbol{D} + \boldsymbol{I})^{-1/2}$ **(self-loops)**
- Simple 1-localized GCNN (Kipf and Welling 2016):
  - $\boldsymbol{x}_{l+1} = \boldsymbol{\sigma}(\widetilde{\boldsymbol{P}}\boldsymbol{x}_l\boldsymbol{W}_l), \quad \boldsymbol{W}_l \in \mathbb{R}^{c_{l-1} \times c_l}$ learnable weights.
  - Good for small scale classification, but known for **oversmoothing**.
- (Chen et. al 2020) proposed *GCN2*:
  - $\boldsymbol{x}_{l+1} = \boldsymbol{\sigma}\left[\left((1-\alpha_l)\widetilde{\boldsymbol{P}}\boldsymbol{x}_l + \alpha_l\boldsymbol{x}_0\right)\left((1-\beta_l)\boldsymbol{I} + \beta_l\boldsymbol{W}_l\right)\right]$.
  - Adds *skip connection* and *identity map*.
- Equivalent to $\boldsymbol{L}$-degree polynomial filter ($\boldsymbol{L}$-localized on $\mathcal{G}$).

GC Autoencoder ROM

- Split network idea (Fresca et al. 2020).
- GCN2 layers (Chen et al. 2020) encode-decode.
- Blue layers are fully connected.
- Purple network simulates low-dim dynamics.



CNN Architecture

- 5x5 kernels.
- Zero padding necessary.
- Channels increase x4 at each layer.

layer	input size	kernel size	stride	padding	output size	activation
Samples of size (2145, 1) are zero padded to size (4096,1) and reshaped to size (1, 64, 64).						
1-C	(1, 64, 64)	5x5	2	SAME	(4, 32, 32)	ELU
2-C	(4, 32, 32)	5x5	2	SAME	(16, 16, 16)	ELU
3-C	(16, 16, 16)	5x5	2	SAME	(64, 8, 8)	ELU
4-C	(64, 8, 8)	5x5	2	SAME	(256, 4, 4)	ELU
Samples of size (256, 4, 4) are flattened to size (4096).						
1-FC	4096				n	ELU
End of encoding layers. Beginning of decoding layers.						
2-FC	n				4096	ELU
Samples of size (4096) are reshaped to size (256, 4, 4).						
1-TC	(256, 4, 4)	5x5	2	SAME	(64, 8, 8)	ELU
2-TC	(64, 8, 8)	5x5	2	SAME	(16, 16, 16)	ELU
3-TC	(16, 16, 16)	5x5	2	SAME	(4, 32, 32)	ELU
4-TC	(4, 32, 32)	5x5	2	SAME	(1, 64, 64)	ELU
Samples of size (1, 64, 64) are reshaped to size (4096, 1) and truncated to size (2145, 1).						

2-D Parameterized Heat Equation

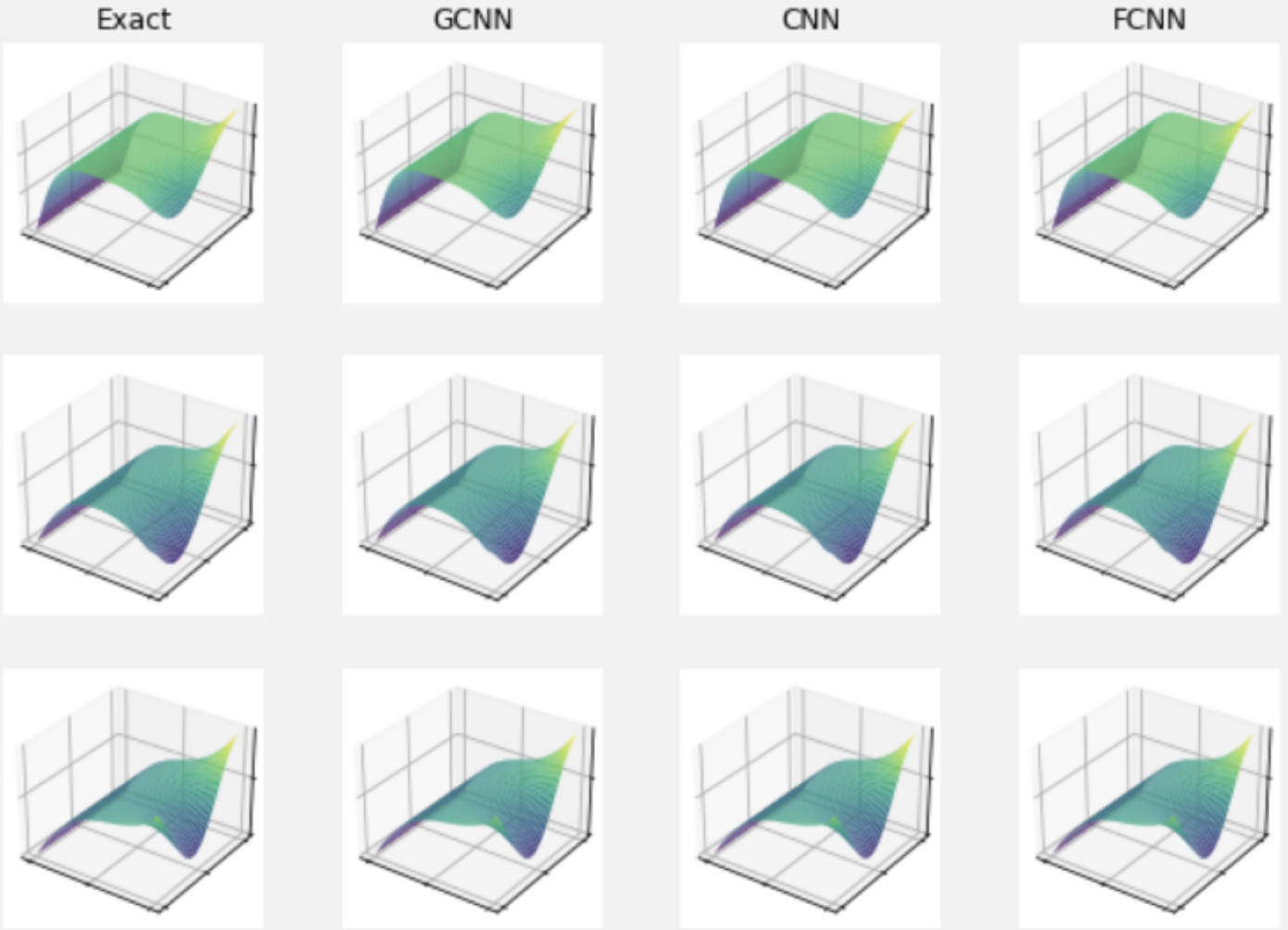
- Consider $u = u(x, y, t, \mu)$,

  $U = [0,1] \times [0,2]$,

  $\mu \in [0.05,0.5] \times \left[\frac{\pi}{2}, \pi\right]$.

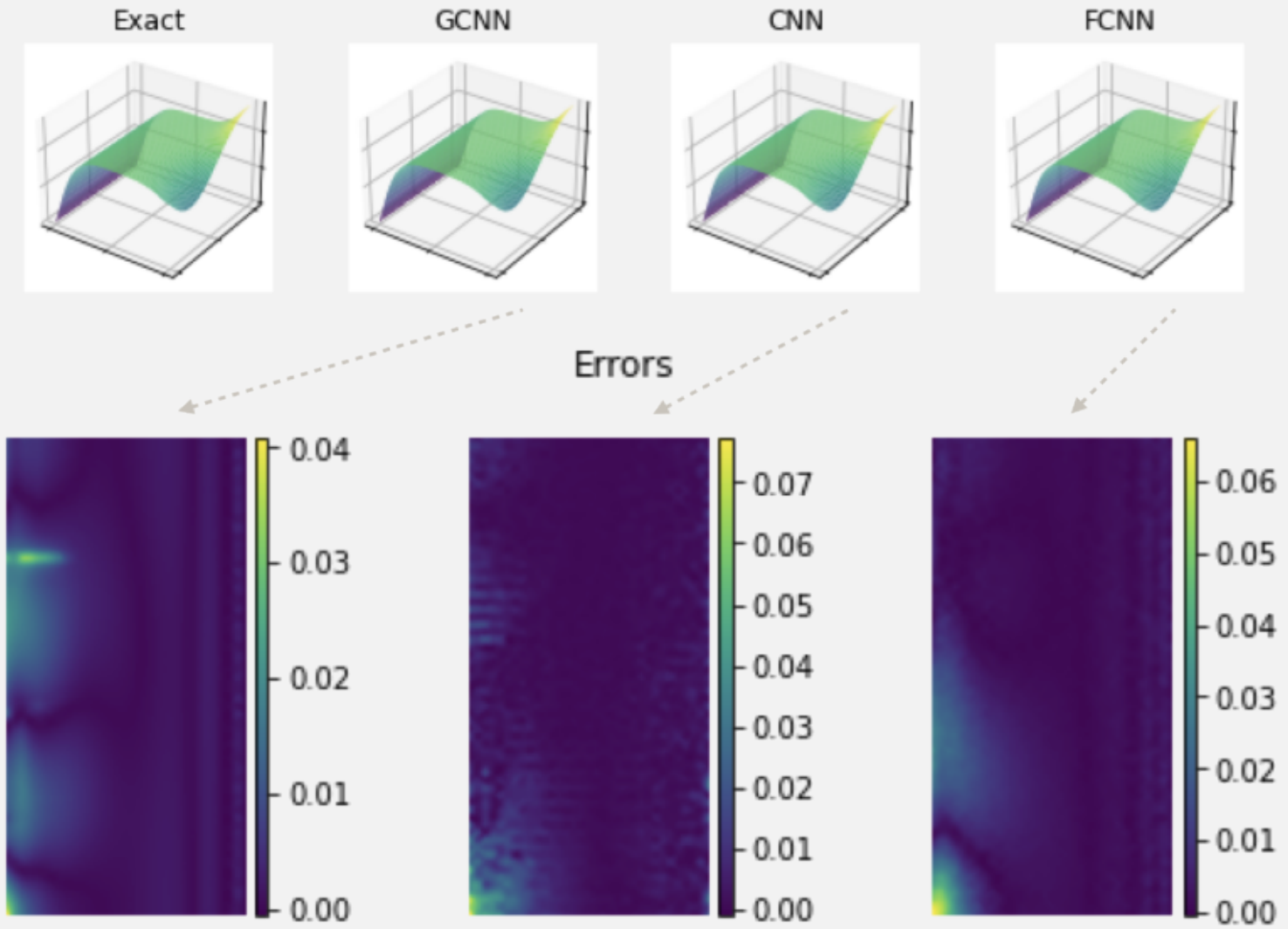
- Solve:

$$u_t - \Delta u = 0,$$
$$u(0, y, t) = -0.5,$$
$$u(1, y, t) = \mu_1 \cos(\mu_2 y),$$
$$u(x, y, 0) = 0.$$



2-D Parameterized Heat Equation: Results

- Results shown for $N = 4096,\ n = 10$.
- GCNN has lowest error and least memory requirement (by >10x) .
- CNN is worst…
  - Cheap hacks have a cost!



Unsteady Navier-Stokes Equations

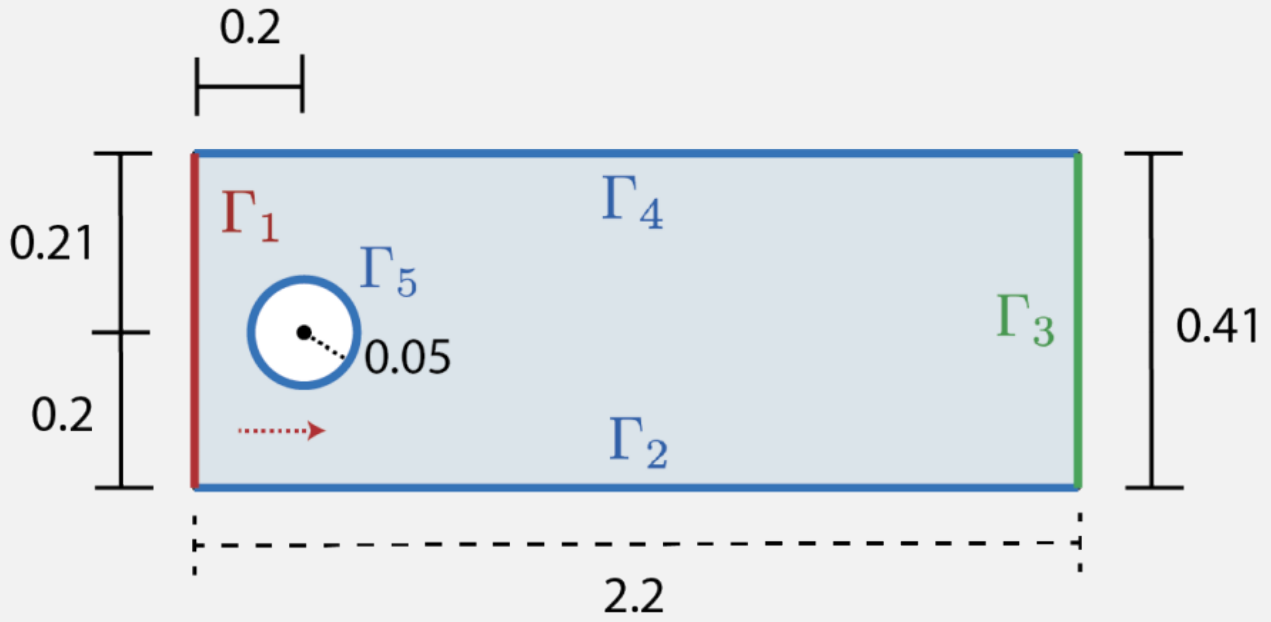
- Schafer-Turek benchmark:

$$\dot{\boldsymbol{u}} - \nu\Delta\boldsymbol{u} + \nabla_{\boldsymbol{u}}\boldsymbol{u} + \nabla p = \boldsymbol{f},$$

$$\nabla \cdot \boldsymbol{u} = 0,$$

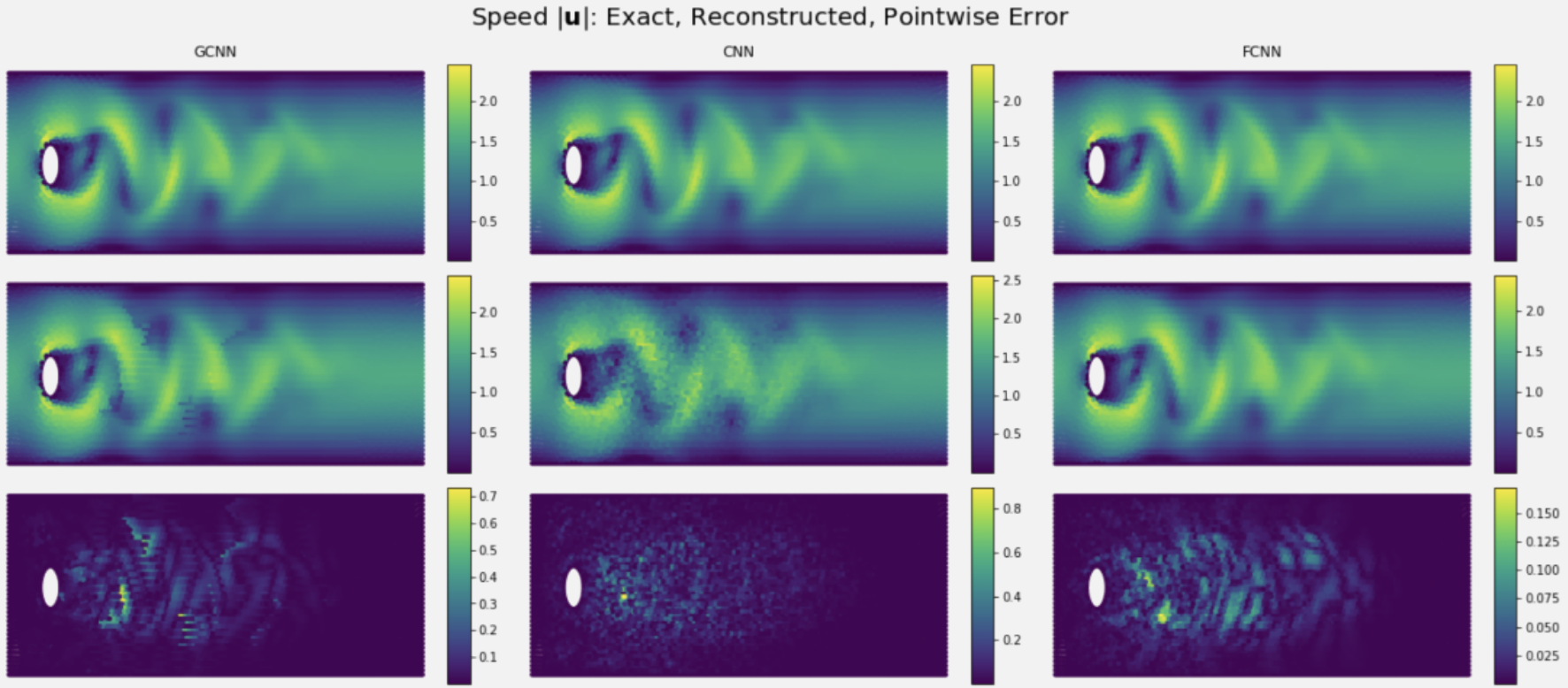
$$\boldsymbol{u}|_{t=0} = \boldsymbol{u}_0.$$

- Impose $\mathbf{0}$ boundary conditions on $\Gamma_2, \Gamma_4, \Gamma_5$. Do nothing on Γ_3. Parabolic inflow on Γ_1.



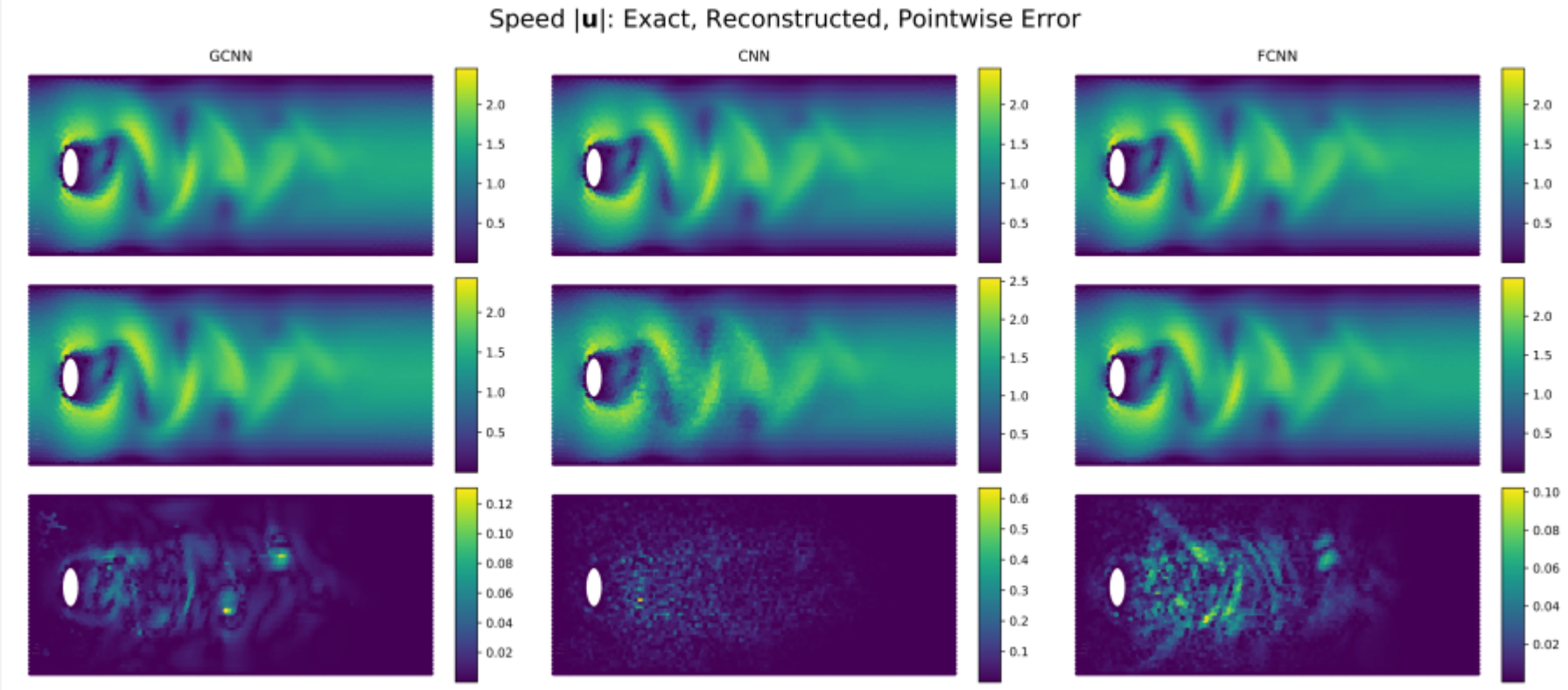
Unsteady NSE: ROM Results

- $N = 10208$, $n = 32$.
- Reynolds # 185
- FCNN best result.
- GCNN still beats CNN.



Unsteady NSE: Enc/Dec Results

- GCNN as accurate as FCNN.
- GCNN memory cost **>50x less** than FCNN.
- CNN still worst.



Unsteady NSE: Results

- ADAM optimized.
- GCNN slightly overfits on ROM.

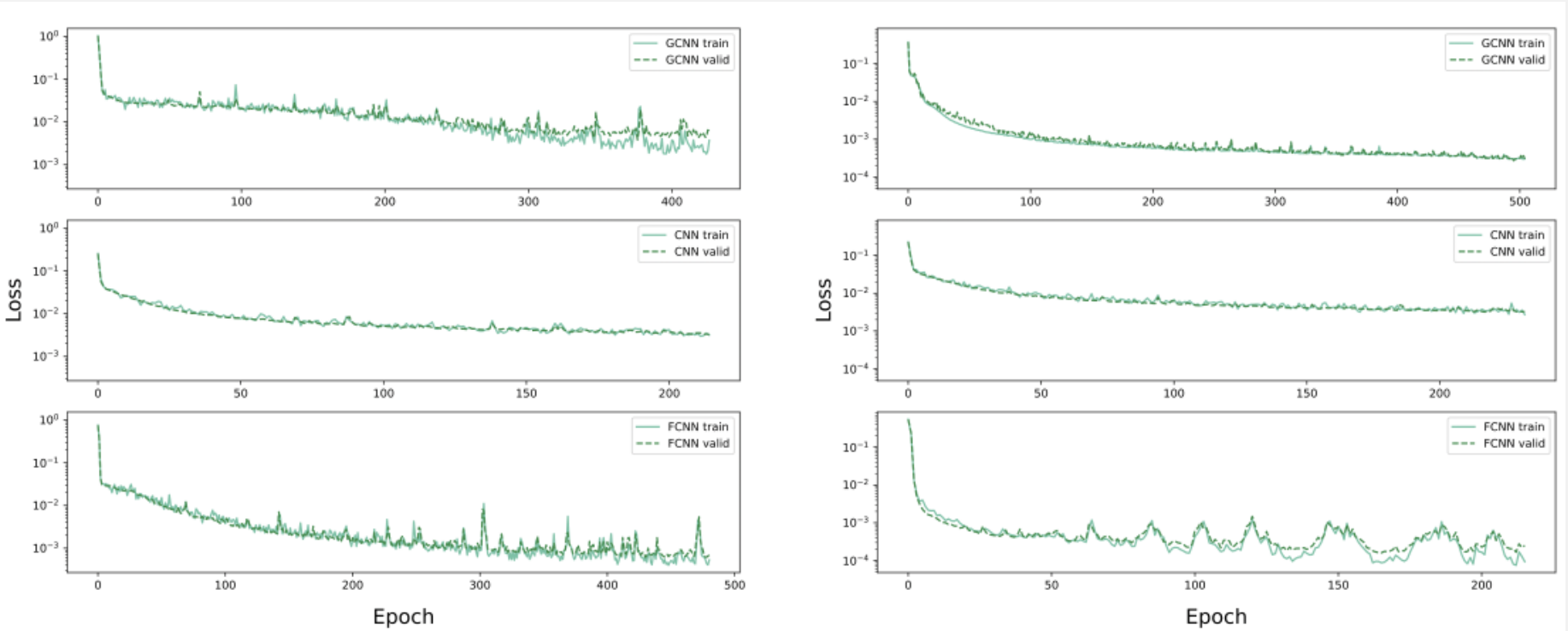


FIGURE 10. Training and validation losses incurred during neural network training for the Navier-Stokes example. Left: prediction problem with $n = 32$. Right: compression problem with $n = 32$.

Challenges with NN methods

- POD always improves with n.
- NNs break "curse of dimensionality", but at high cost.
- POD has inherent advantage: knowledge of equations.

	Encoder/Decoder + Prediction					Encoder/Decoder only				
Network	n	$R\ell_1\%$	$R\ell_2\%$	Size (MB)	Time per Epoch (s)	n	$R\ell_1\%$	$R\ell_2\%$	Size (MB)	Time per Epoch (s)
POD	2	10.0	15.85	0.323	N/A	2	6.75	10.0	0.323	N/A
GCN		9.07	14.6	0.476	33		7.67	12.2	0.410	32
CNN		7.12	11.1	224	210		11.2	17.6	224	190
FCNN		1.62	2.87	330	38		1.62	2.70	330	38
POD	32	2.70	3.97	5.17	N/A	32	0.428	0.674	5.17	N/A
GCN		2.97	5.14	5.33	32		0.825	1.49	5.26	32
CNN		4.57	7.09	232	230		4.61	7.24	232	220
FCNN		1.39	2.64	330	38		0.680	1.12	330	38
POD	64	2.80	4.36	10.3	N/A	64	0.191	0.318	10.3	N/A
GCN		2.88	4.96	10.5	33		0.450	0.791	10.4	33
CNN		3.42	5.33	241	270		2.42	3.57	241	260
FCNN		1.45	2.64	330	38		0.704	1.19	330	37

Thank you!

Contact: adgrube@sandia.gov