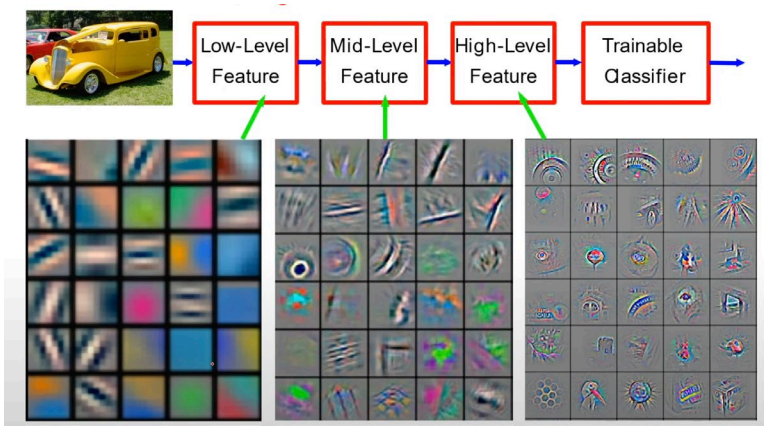
Architectural Nuances and Benchmark Gaps in Scientific ML:

Two vignettes

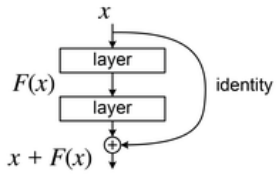
Andrej Risteski
(Carnegie Mellon University)

Two major ingredients that fueled modern ML

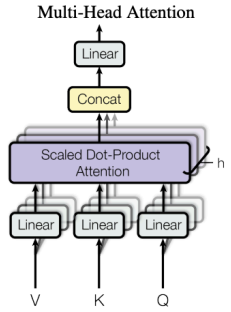
Architectures



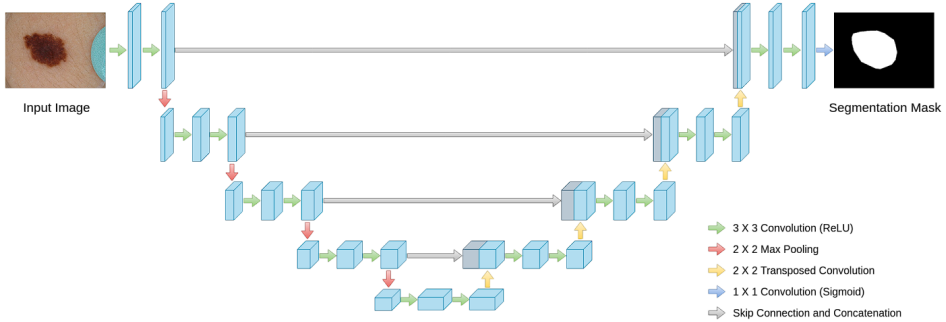
Convolution



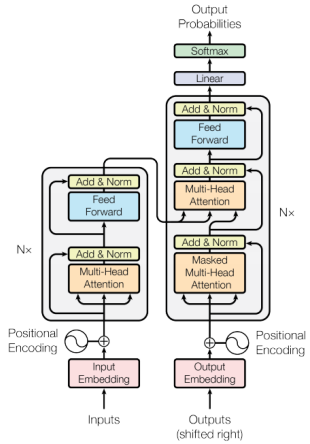
Residual connections



Attention

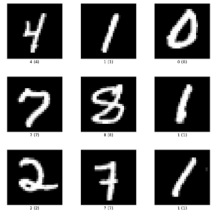


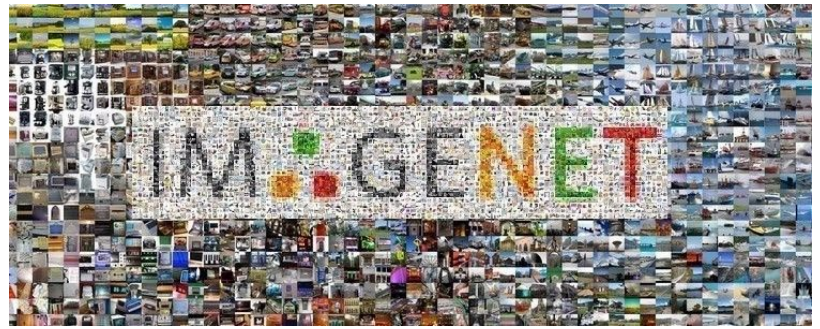
U-net



Transformer

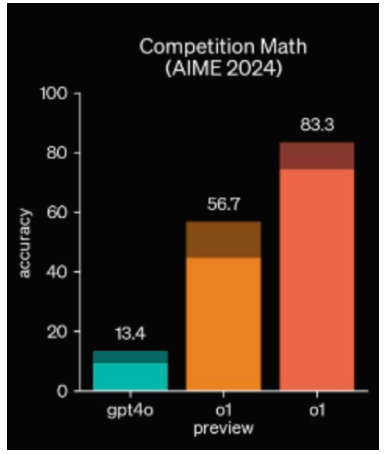
Benchmarks

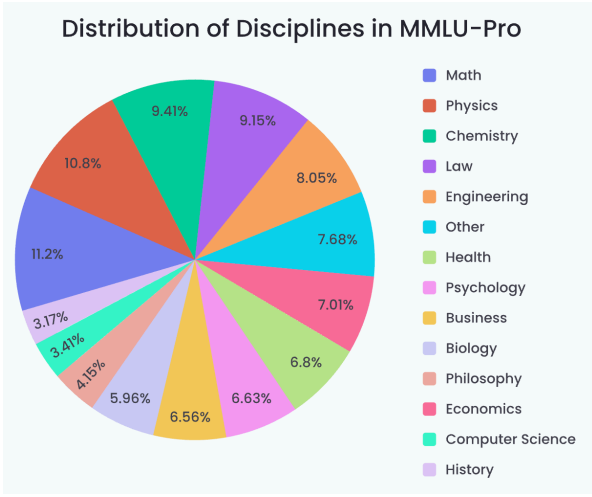




GLUE

SuperGLUE





Scientific ML: less settled ground

Flurry of architectural work, but less settled understanding

(A variety of graph neural networks (GNNs), Fourier & scale-aware architectures, symmetry-aware architectures…)

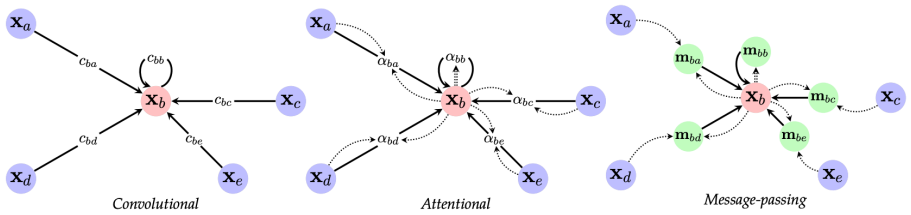


Figure from (Veličković et al '21)

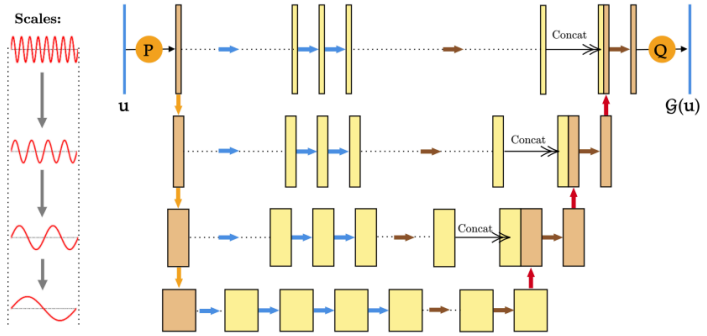


Figure from (Raonić et al '23)

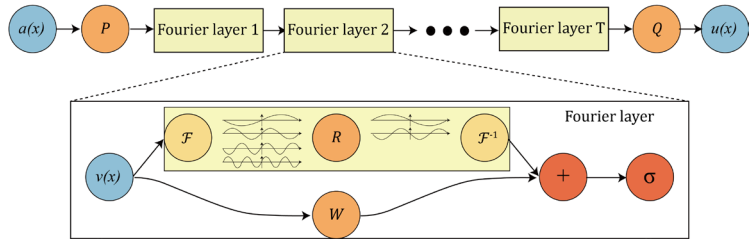


Figure from (Li et al '21)

Benchmarks often too easy, missing crucial desiderata, misaligned w/ application domain,…

Position: Graph Learning Will Lose Relevance Due To Poor Benchmarks

Maya Bechler-Speicher [*1 2] Ben Finkelshtein [*3] Fabrizio Frasca [*4] Luis Müller [*5] Jan Tönshoff [*5]
Antoine Siraudin [5] Viktor Zaverkin [6] Michael M. Bronstein [3] Mathias Niepert [7] Bryan Perozzi [8]
Mikhail Galkin [8] Christopher Morris [5]

Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations

Nick McGreivy ✉ & Ammar Hakim

Nature Machine Intelligence **6**, 1256–1269 (2024) | Cite this article

A CRITICAL LOOK AT THE EVALUATION OF GNNS UNDER HETEROPHILY: ARE WE REALLY MAKING PROGRESS?

Oleg Platonov
HSE University, Yandex Research
olegplatonov@yandex-team.ru

Denis Kuznedelev
Skoltech, Yandex Research
dkuznedelev@yandex-team.ru

Michael Diskin
HSE University, Deepcake.io
michael.s.diskin@gmail.com

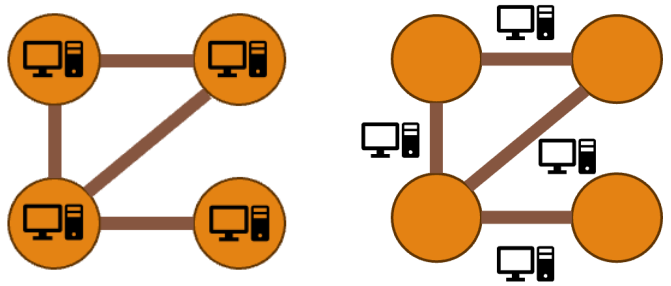
Artem Babenko
Yandex Research
artem.babenko@phystech.edu

Liudmila Prokhorenkova
Yandex Research
ostroumova-la@yandex-team.ru

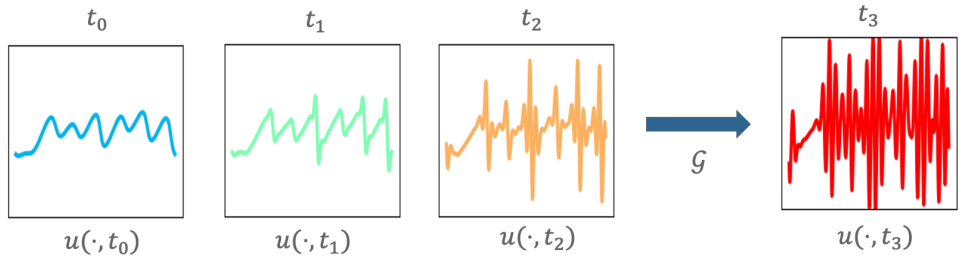
This talk

Two vignettes of **modest architectural changes** mattering in the **right regimes**:

Part I: Message-passing GNNs which maintain **edge embeddings** when the **graph has bottlenecks** & **memory is bounded**.



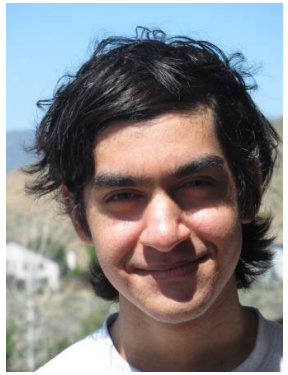
Part II: Time-dependent PDE solvers which use **memory** explicitly and the system is **partially observed**.



Common benchmarks would not have revealed these effects!

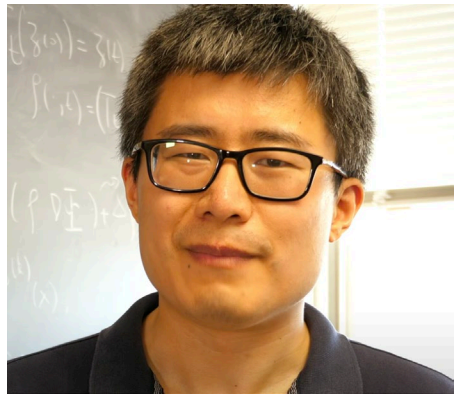
Part I: The value of edge embeddings for memory-bounded GNNs

Tanya Marwah
(CMU → Flatiron Institute)



Dhruv Rohatgi
(MIT)

Zack C. Lipton
(CMU & Abridge)



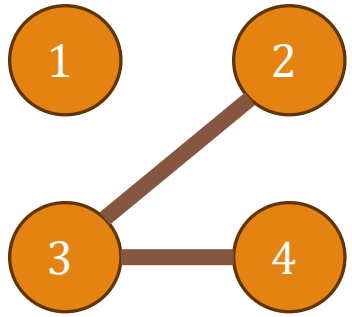
Jianfeng Lu
(Duke)

Ankur Moitra
(MIT)

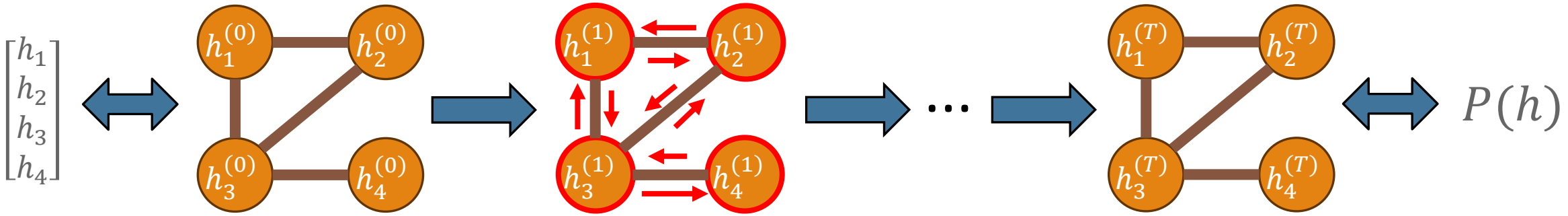
A primer on (message-passing) graph neural networks

GNNs are a generic approach to ML tasks involving graphs:

Node-level or edge-level prediction (e.g. predicting type of node or edge), graph-level prediction (e.g. predicting properties of molecule), …



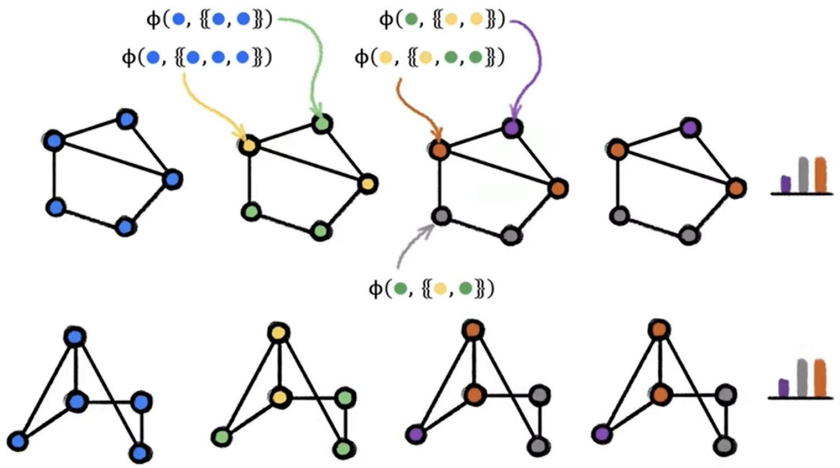
GNNs learn a message-passing protocol $\mathrm{P}: (\mathbb{R}^{\mathrm{d}})^{\mathrm{V}} \rightarrow (\mathbb{R}^{\mathrm{k}})^{\mathrm{V}}$ on a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$.



States $\{h_i^{(T)}\}$ aggregated to produce predictions (typically just linear "read-off" layers)

Motivation: Protocol parametrized to make $\{h_i^{(t)}\}$ equivariant to node permutations.

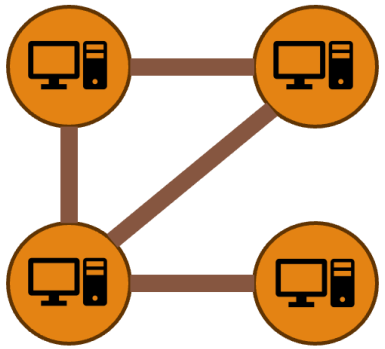
A primer on (message-passing) graph neural networks



Symmetry lens:

Much work on expressive power from the point-of-view of invariance, i.e. relationships to Weisfeihler-Lehman isomorphism tests. (Xu et al '19, Maron et al '19, Huang & Villar '21)

Does a GNN architecture necessarily output the same values even if two graphs are non-isomorphic?



Computational machine lens:

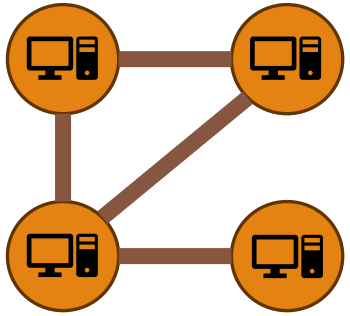
We view GNN's as a "**computational machine**" which receives inputs (=initial node features) and tries to compute some (symmetric) function (in a local, distributed fashion).

If the nodes have natural "resource bounds" (e.g. memory), what kind of impact do they have representationally?

A modest architectural change: edge-based GNNs

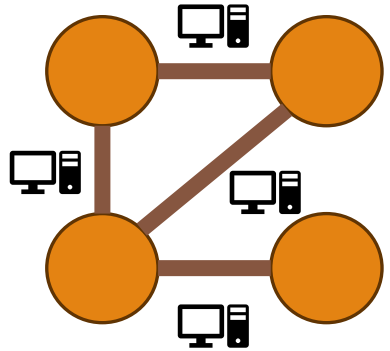
The “processors” in the message-passing protocol can also lie on the edges of the graph:

Node-based GNN



VS

Edge-based GNN



Introduced for “edge-centric” tasks (Cai et al. ‘21; Liang & Pu ‘23) for which we often have rich input edge features (Gilmer et al. ‘17; Choudhary & DeCost ‘21).

Question: Are the benefits mostly/solely due to richer input data, or do edge-embeddings confer representational benefits?

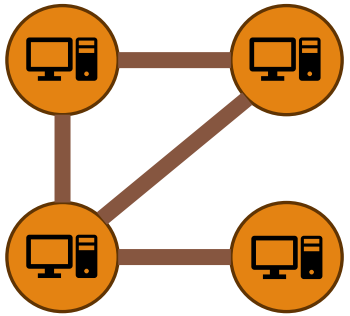
Theoretical abstractions (node variant)

Definition (informal). A node message-passing protocol P on graph G, given inputs $\{I_v\}_{v \in V}$ iteratively computes:

$$h_v^{(t)} := f_v^{(t)}\left(h_w^{(t-1)} : w \in N_G(v)\right), \qquad h_v^{(0)} := I_v$$

Moreover, nodes are memory-constrained: $B := \max_{t,v} \textsc{BitComplexity}\left(h_v^{(t)}\right)$

& the update functions are invariant to node indices.



Memory constraints model bounded dimensionality of latent representations (w/ finite precision).

No computational constraints on $f_v^{(t)}$: makes lower bounds stronger!

Alphabet for inputs

$h_v^{(t)}$, when initialized with $h^{(0)} = I$

A protocol P with T rounds implements a function $g: \Phi^V \to \{0,1\}^V$ if:

$$\forall v \in V, \forall I: P_{T,v}(I) = g(I)_v$$

***Read**: there exists a GNN with depth T that implements the function g.*

Theoretical abstractions (edge variant)

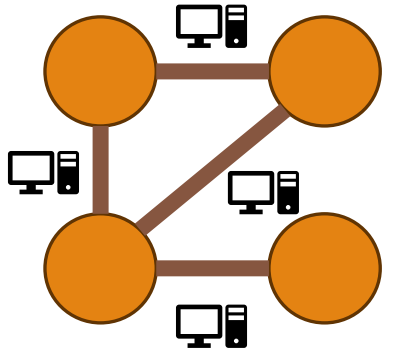
{{ }} = multiset

Definition (informal). An **edge** message-passing protocol P on graph G, given inputs $\{I_v\}_{v \in V}$ iteratively computes:

$$h_e^{(t)} := f_e^{(t)}\left(h_{e'}^{(t-1)} : e' \in M_G(e)\right), \qquad h_e^{(0)} := \{\{I_u, I_v\}\}$$

Moreover, nodes are memory-constrained: $B := \max_{t,v} \textsc{BitComplexity}\left(h_e^{(t)}\right)$

& the update functions are invariant to node indices.



Memory constraints model bounded dimensionality of latent representations (w/ finite precision).
In all our constructions, the functions $f_e^{(t)}$ are exceedingly simple.

Symmetric "read-off" functions

A protocol P with T rounds implements a function $g: \Phi^V \to \{0,1\}^V$ if:

$$\forall v \in V, \forall I: \tilde{f}_v\left(P_{T,e}(I) : v \in e\right) = g(I)_v$$

***Read**: there exists a GNN with depth T that implements the function g.*

Main results

Thm 1 (informal). For any n, there is a graph with $O(n)$ vertices, and a function $g: \{0,1\}^V \to \{0,1\}^V$ which can be implemented by an edge protocol with $B = O(1)$ and $T = O(1)$.

On the other hand, any node protocol requires $TB = \Omega(\sqrt{n})$

***Read**: node-based protocols have to pay either with depth or memory.*

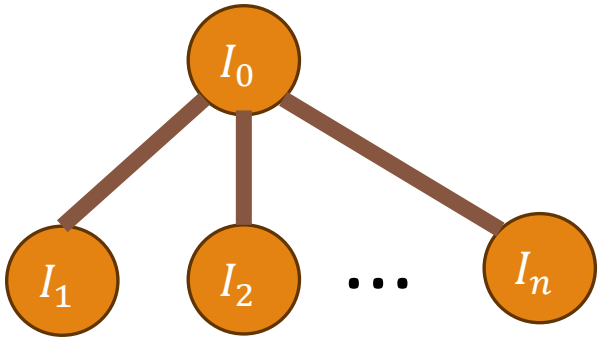
Thm 2 (informal). We can even construct function $g: \{0,1\}^V \to \{0,1\}^V$ representing natural task: calculating the MAP (= Maximum A-Posteriori) value in a pairwise graphical model.

***Read**: belief propagation in graphical models cannot be simulated with a shallow node-based GNN with small memory.*

Thm 3 (informal). Without memory constraints, any T-round edge protocol can be simulated by a $(T+1)$-round node protocol.

***Read**: the symmetry lens alone is insufficient to capture separation.*

Main intuition: bottleneck nodes



Warm-up task: Each node $i > 0$ wants to calculate if another node $j > 0$ is is equal to it: $\mathrm{g}(I)_i = 1(\exists j, I_i = I_j)$.

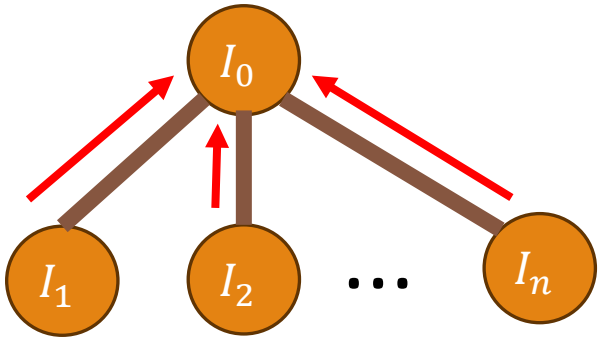
Furthermore, let's allow larger alphabet: $\forall i \in [n],\ \ I_i \in [n]$

Intuitions:

- In edge-based protocol, every edge sees every other edge (they are all adjacent).

To check if node i and node j are the same, just check if multisets $I_{\{0,i\}}$ and $I_{\{0,j\}}$ are equal.

Main intuition: bottleneck nodes



Warm-up task: Each node $i > 0$ wants to calculate if another node $j > 0$ is is equal to it: $\mathrm{g}(I)_i = 1(\exists j, I_i = I_j)$.

Furthermore, let's allow larger alphabet: $\forall i \in [n], \ \ I_i \in [n]$

Intuitions:

- In edge-based protocol, every edge sees every other edge (they are all adjacent).
- In node-based protocol, node i > 0 can only communicate with node j > 0 via node 0.

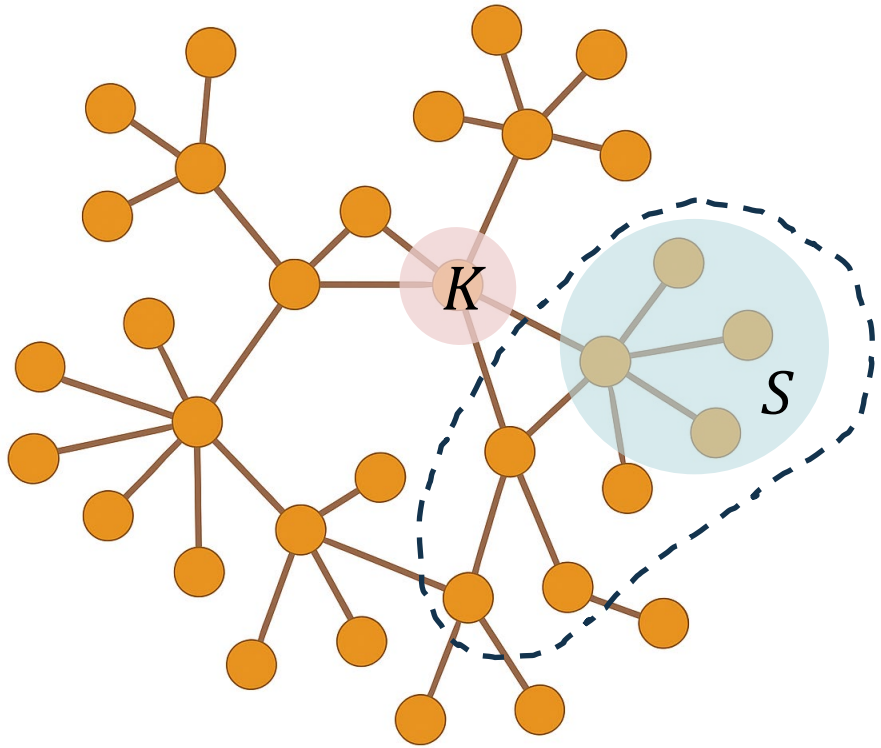
One option is for node 0 to collect & broadcast all counts, but requires memory $\sim n \log n$

If memory is bounded by B *bits, do we have to pay with rounds?*

The main lemma

Lemma: Consider a "destination" set S and a "bottleneck" set K. Let F be the distance T-neighborhood of S in G[$\overline{K}$]. Then, for any node protocol implementing g:

$$TB|K| \geq \log \max_{I_F} |\{g_S(I_F, I_{\overline{F}})\}_{I_{\overline{F}}}|$$



Proof sketch: Values of nodes in S after T rounds can only depend on:

(1) Inputs on nodes within dist. T neighborhood of S in G[$\overline{K}$], i.e. nodes in F.

(2) The protocol values up to round T of the nodes in K.

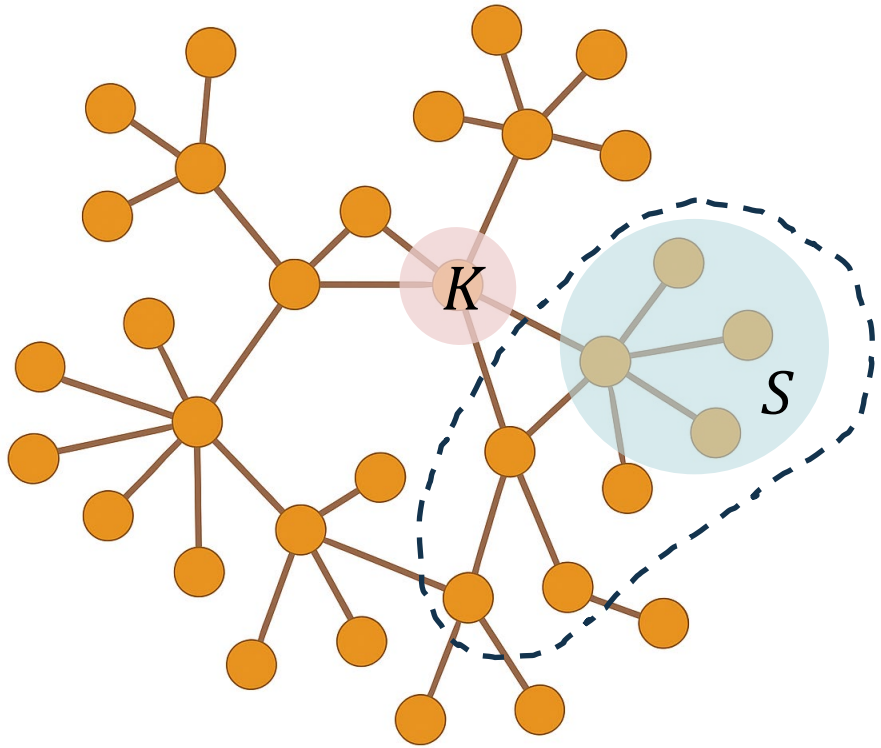
Thus, for any value of the inputs I_F:

$$\left|\{g_S(I_F, I_{\overline{F}})\}_{I_{\overline{F}}}\right| \leq 2^{B\,T|K|}$$

The main lemma

Lemma: Consider a "destination" set S and a "bottleneck" set K. Let F be the distance T-neighborhood of S in G[$\overline{K}$]. Then, for any node protocol implementing g:

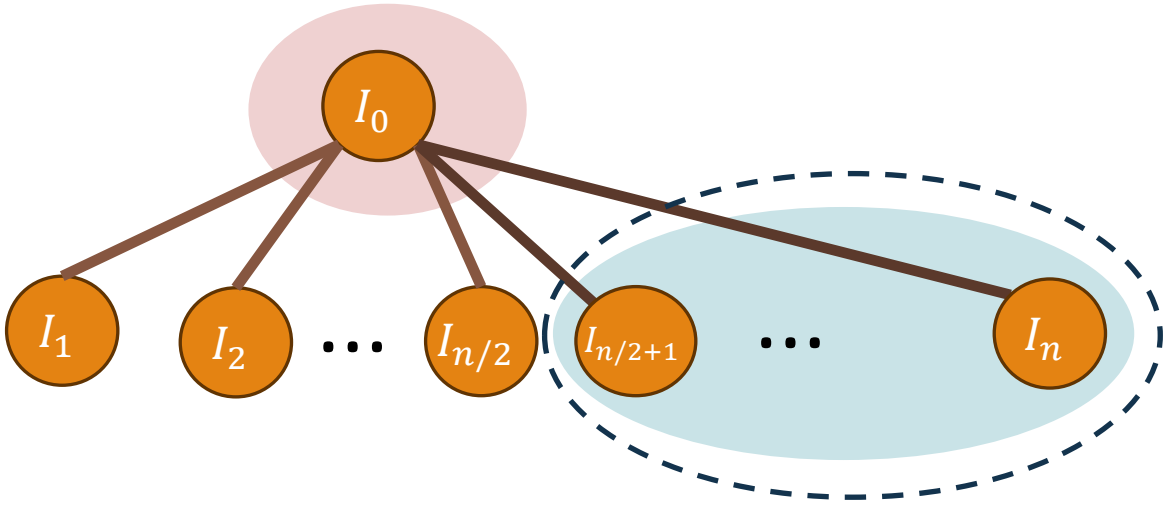
$$TB|K| \geq \log \max_{I_F} |\{g_S(I_F, I_{\overline{F}})\}_{I_{\overline{F}}}|$$



Related ideas in two areas of TCS:

- "Flow" between variables in functions in *time-space tradeoffs* (Grigoriev '76, Savage '98)
- "Light-cone" techniques in lower bounds for *distributed computation* (e.g. LOCAL in Linial '92, CONGEST in Peleg '92)

Using the lemma for warm-up task



Take $\mathrm{K} = \{0\}, \mathrm{S} = \{\mathrm{n}/2 + 1, \dots, n\}$.

Then, $\mathrm{G}[\bar{K}]$= isolated nodes $\{1, \dots, n\}$.

Thus, $F = [n/2 + 1, n]$ and $\bar{F} = [1, n/2]$

Take $\mathrm{I_F} = (1,2, \dots, n/2)$.

Then, for any string $x \in \{0,1\}^{n/2}$ we can "engineer" an input in $\mathrm{I}_{\bar{F}}$ s.t. $g_S(I_F, I_{\bar{F}}) = x$:

- *If we want to to make* $g_i(I_F, I_{\bar{F}}) = 1$, *choose* $I_{i-n/2} = I_i$
- *If we want to to make* $g_i(I_F, I_{\bar{F}}) = 0$, *choose* $I_{i-n/2} \in [n/2 + 1, n]$

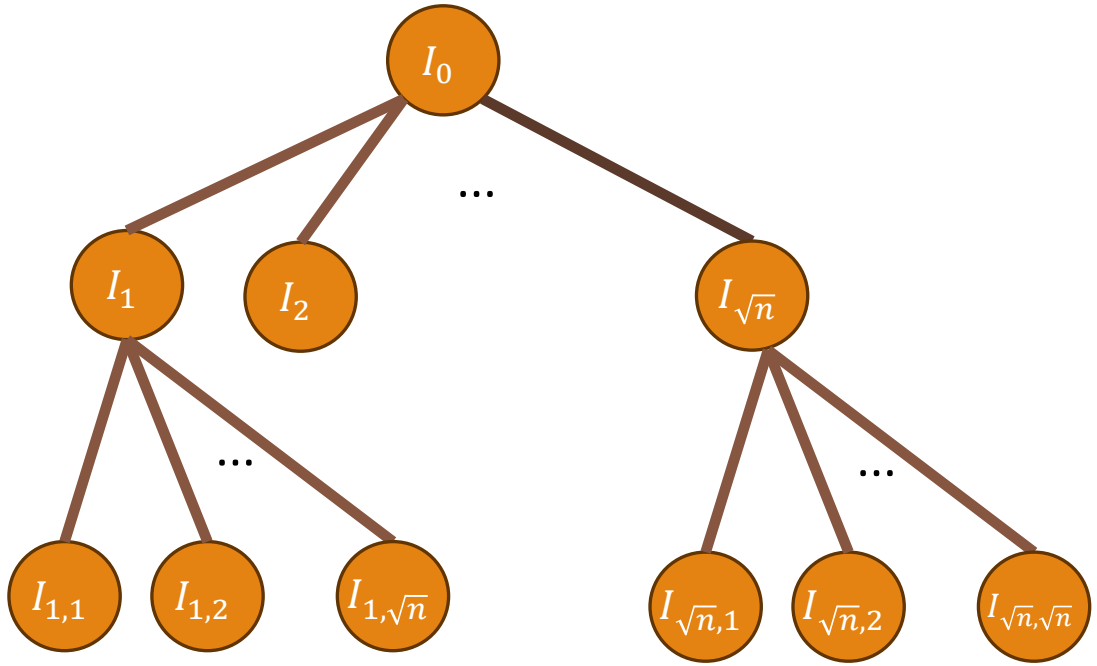
Thus, $TB \geq \log |\{g_S(I_F, I_{\bar{F}})\}_{I_{\bar{F}}}| = n/2$

Seems we got a stronger result than claimed?

Beyond the warm-up: smaller alphabet

The function we constructed has domain $[n]^n$. We can also construct function w/ domain $\{0,1\}^n$.

⚠ Cannot just change domain in construction: node 0 can remember count of 0's and 1's (only $\log n$ bits needed). In other words, a "compact summary" of inputs suffices.



Solution:

Can simulate large-domain construction by attaching "subtree" to each node i & "encode inputs in unary":

$$g(I)_i = 1(\exists j \text{ s.t., subtree i \& j have same \# of 1's})$$

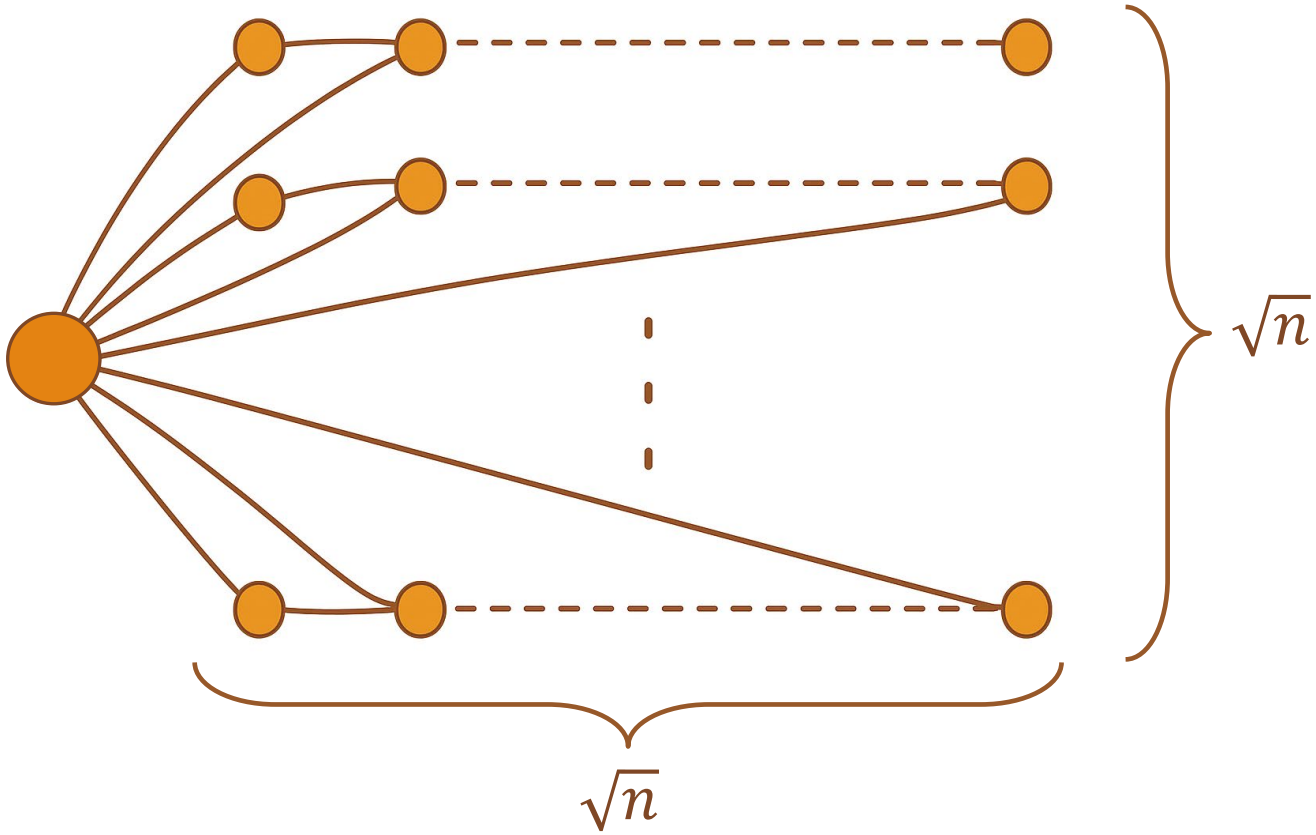
$$= 1(\exists j \text{ s.t., i \& j have same \# neighbors = 1})$$

This is where we get $BT = \Omega(\sqrt{n})$

Beyond the warm-up: “natural” graph task

Thm 2 (informal). We can even construct function $g: \{0,1\}^V \to \{0,1\}^V$ representing natural task:

calculating the MAP (= Maximum A-Posteriori) value in a pairwise graphical model.



We can also make the task be a canonical pairwise graphical model task:

$$\text{argmax}_x \sum_{\{i,j\} \in E(G)} \phi_{\{i,j\}}(x_i, x_j) + \sum_{i \in V(G)} \phi_i(x_i)$$

Previous function we constructed involves “higher order” interactions, so graph topology needs to be modified.

Idea: ”Copying” rightmost node on each path to the leftmost end can be written in the above form.

Remarks

Morally, similar to fine-grained architectural separations in standard deep learning:

- Depth separation for feedforward nets (Telgarsky '16, Schmidt-Hieber '19,'20)
- Depth separation for Transformers (Sanford et al '24)
- Parallelizability of "sequential tasks" using Transformers? (Liu et al '23)

Morally, seems related to "over-squashing" ? (Alon & Yadav '21)

- Task specification + topological bottlenecks lead to memory/computation bottlenecks:

  TB being large means either depth or width has to grow
  (unclear how this interacts w/ training dynamics)

Would common benchmarks catch this?

On standard datasets, advantage of edge-based GNNs isn't substantial:

Model	ZINC	MNIST	CIFAR-10	Peptides-Func	Peptides-Struct
	MAE (↓)	ACCURACY (↑)	ACCURACY (↑)	AP (↑)	MAE (↓)
GCN	0.3430 ± 0.034	$\mathbf{95.29 \pm 0.163}$	55.71 ± 0.381	0.6816 ± 0.007	0.2453 ± 0.0001
Edge-GCN (Ours)	$\mathbf{0.3297 \pm 0.011}$	94.37 ± 0.065	$\mathbf{57.44 \pm 0.387}$	$\mathbf{0.6867 \pm 0.004}$	$\mathbf{0.2437 \pm 0.0005}$

Table 1: Comparison of node-based (3) and edge-based (4) GCN architectures across various graph benchmarks. The performance of the edge-based architecture robustly matches or improves the node-based architecture.

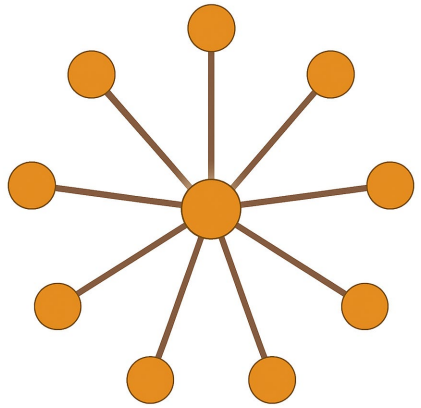
But, we create simple (synthetic, diagnostic) tasks on which node-based architectures are much worse:

Idea: "Plant" a dataset on a star graph:

(1) Take edge-based GNN with random weights.

(2) Choose random initial node features.

(3) Set targets to outputs of planted edge GNN (1) w/ initial features (2).



Would common benchmarks catch this?

On standard datasets, advantage of edge-based GNNs isn't substantial:

Model	ZINC	MNIST	CIFAR-10	Peptides-Func	Peptides-Struct
	MAE (↓)	ACCURACY (↑)	ACCURACY (↑)	AP (↑)	MAE (↓)
GCN	0.3430 ± 0.034	$\mathbf{95.29 \pm 0.163}$	55.71 ± 0.381	0.6816 ± 0.007	0.2453 ± 0.0001
Edge-GCN (Ours)	$\mathbf{0.3297 \pm 0.011}$	94.37 ± 0.065	$\mathbf{57.44 \pm 0.387}$	$\mathbf{0.6867 \pm 0.004}$	$\mathbf{0.2437 \pm 0.0005}$

Table 1: Comparison of node-based (3) and edge-based (4) GCN architectures across various graph benchmarks. The performance of the edge-based architecture robustly matches or improves the node-based architecture.

But, we create simple (synthetic, diagnostic) tasks on which node-based architectures are much worse:

Number of Leaves	Depth of Planted Model (RMSE)					
	5		3		1	
	Edge	Node	Edge	Node	Edge	Node
64	0.004	0.3790	0.011	0.3596	0.008	0.3752
32	0.003	0.3664	0.005	0.3626	0.003	0.3614
16	0.007	0.3336	0.002	0.2100	0.002	0.2847

Table 2: Performance (in RMSE ↓) of edge-based and node-based architectures on a star-graph topology. The first number is the performance of the best edge-based model, and the second is the best node-based model, across a range of depths up to 10 (2× the planted model), widths $\in \{16, 32, 64\}$, and a range of learning rates.

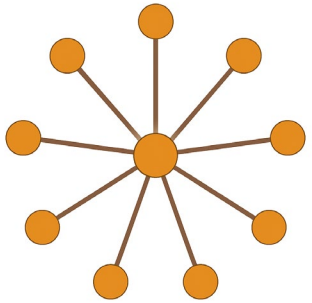
Outlook

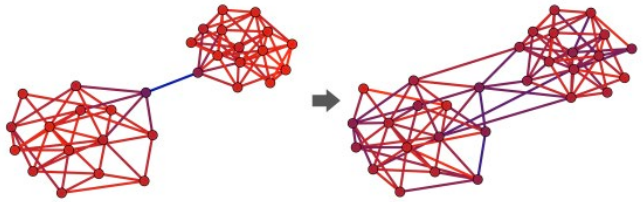
Computational challenges:

"Natural" way to implement an edge-based architecture is computationally expensive for hubs: on a degree n star graph, each edge has n neighbors.

Thus, "total" computation by all edges is $\sim n^2$ for one standard GCN layer, whereas it's $\sim n$ for a node-based architecture.

Note, a variety of works on "rewiring" the graph, but unclear what kind of notion of "performance quality" they preserve.
(e.g. Topping et al '22, Linkerhägner et al '25,…)





Better benchmarks:

Many current datasets are solved by very simple approaches, so unclear we are extracting signal (cf. MNIST and CIFAR era in images.)

"Graph tasks" is a very heterogeneous concept: we probably need more tailored, higher quality benchmarks for each domain.

Position: Graph Learning Will Lose Relevance Due To Poor Benchmarks

Maya Bechler-Speicher [*1 2] Ben Finkelshtein [*3] Fabrizio Frasca [*4] Luis Müller [*5] Jan Tönshoff [*5]
Antoine Siraudin [5] Viktor Zaverkin [6] Michael M. Bronstein [3] Mathias Niepert [7] Bryan Perozzi [8]
Mikhail Galkin [8] Christopher Morris [5]

Oversmoothing, "Oversquashing", Heterophily, Long-Range, and more: Demystifying Common Beliefs in Graph Machine Learning

Adrian Arnaiz-Rodriguez*
ELLIS Alicante
adrian@ellisalicante.org

Federico Errica*
NEC Laboratories Europe
federico.errica@neclab.eu

Part II: The value of memory in (time-dependent) PDE solvers

Tanya Marwah
(CMU ->
Flatiron Institute)

Ricardo Buitrago
(CMU -> Cartesia)

Albert Gu
(CMU &
Cartesia)

The ML approach to PDE solvers

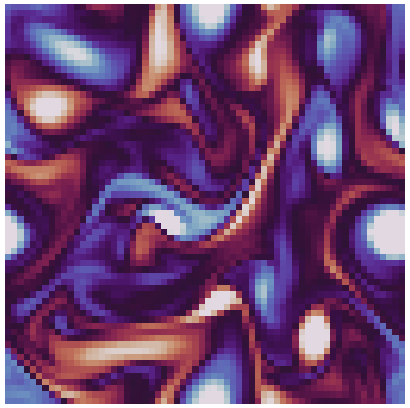
A common scientific computing primitive: solving (time-dependent) PDEs:

$$\begin{cases} \partial_t u(x,t) = \mathcal{L}_t\big(u(x,t)\big), & x \in \Omega, t \geq 0 \\ u(x,0) = f(x), & x \in \Omega \\ u(x,t) = g(x), & x \in \partial\Omega, \mathrm{t} \geq 0 \end{cases}$$

Governing equations

Initial conditions

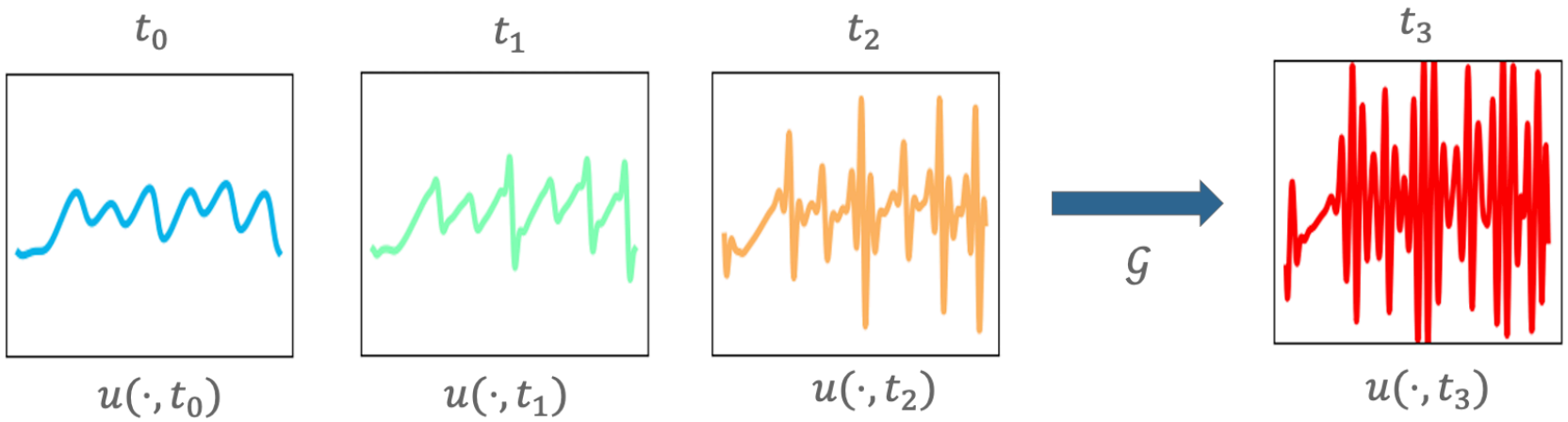
Boundary conditions



The machine learning approach:
Instead of running fixed numerical solver,
learn (hopefully faster) solver from data!
(Lu et al. ‘19, Li et al. ’20, Kovachki et al. ‘21,…)

The ML approach to PDE solvers

Basic idea: discretize time and treat it as a sequence prediction problem (& unroll to "run" model):



How do we parametrize $\mathcal{G}$? (Lots of work!)

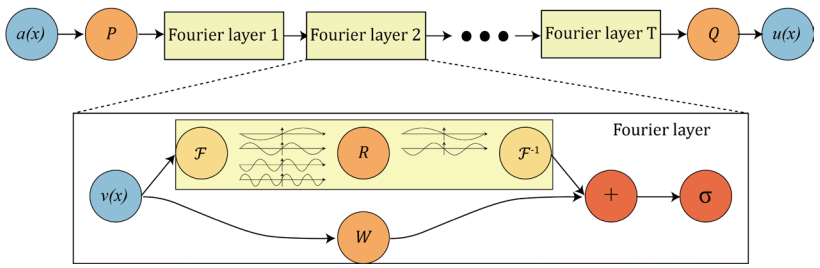


Figure from (Li et al '21)

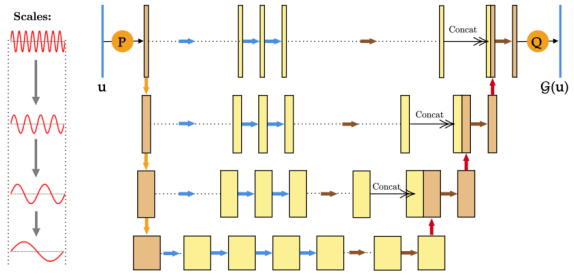


Figure from (Raonić et al '23)

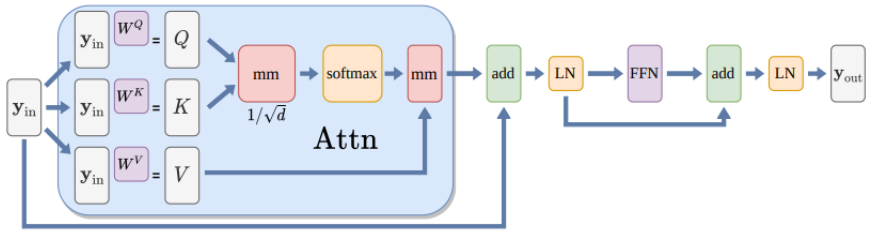
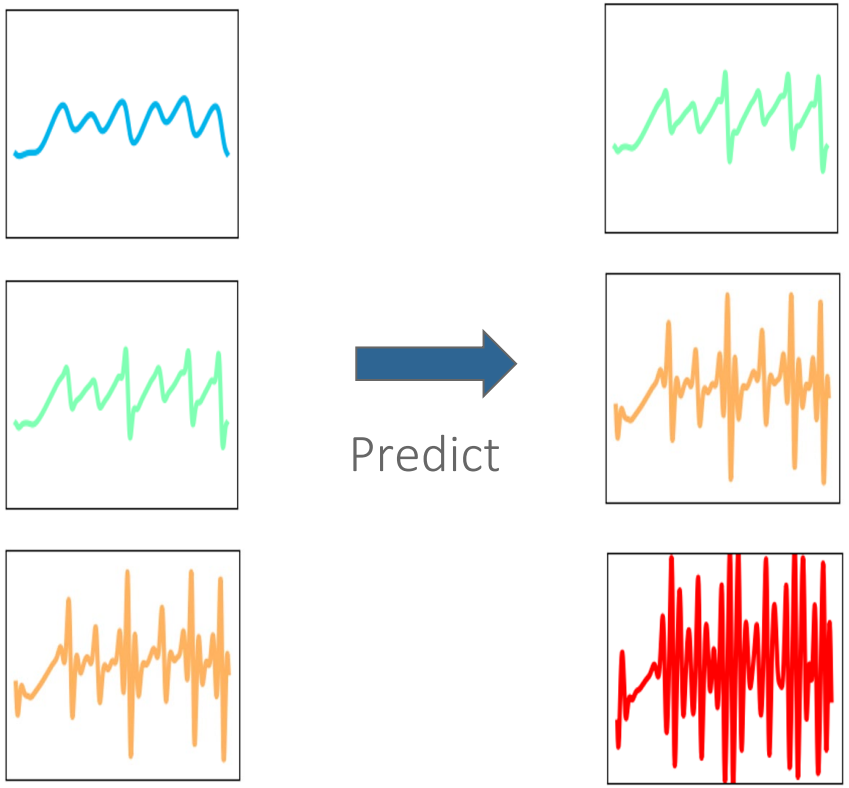


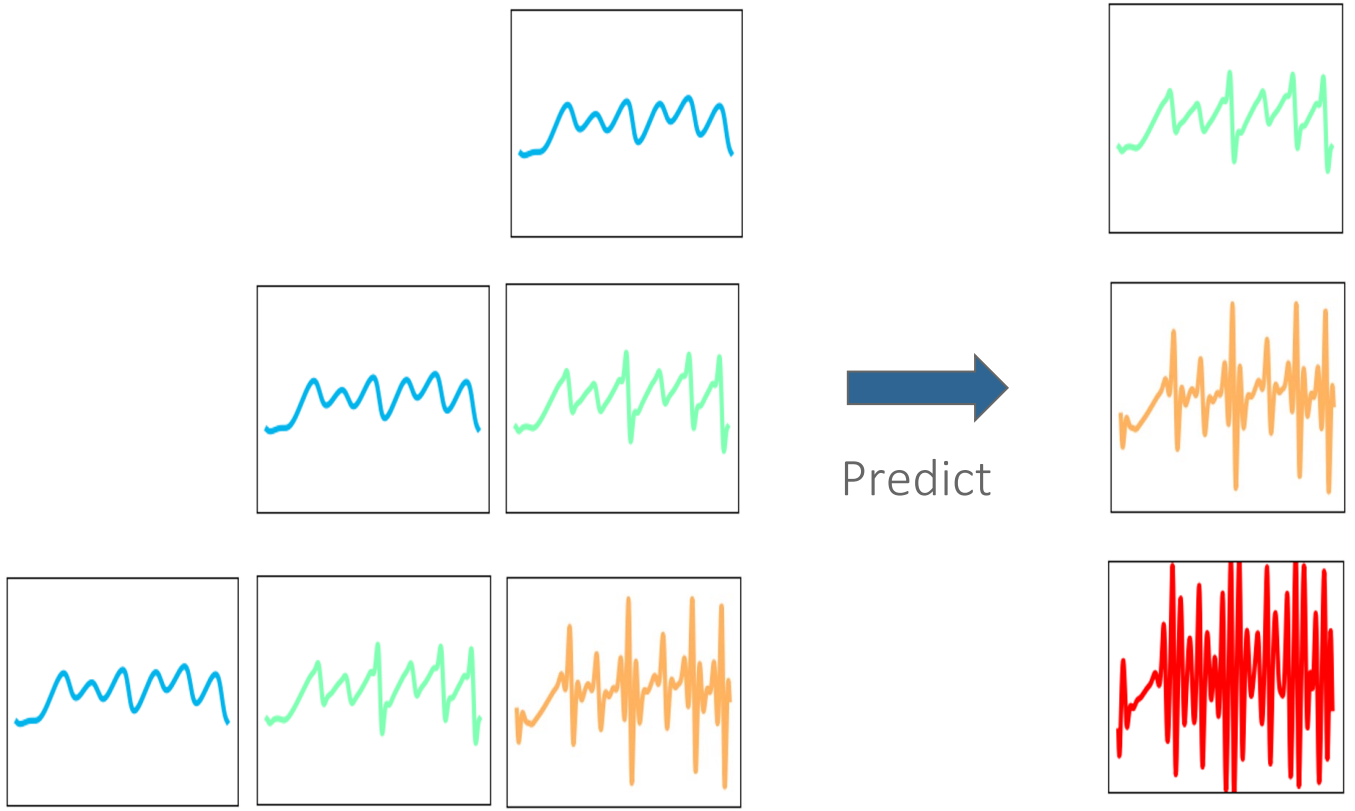
Figure from (Cao et al '21)

Modest q: to be Markovian or not to be Markovian ?



"Markovian" Operator

$$u(t+\Delta t) \approx \mathcal{G}_t(u(t))$$



Operator with memory

$$u(t+\Delta t) \approx \mathcal{G}_t(u(0), u(\Delta t), \ldots u(t))$$

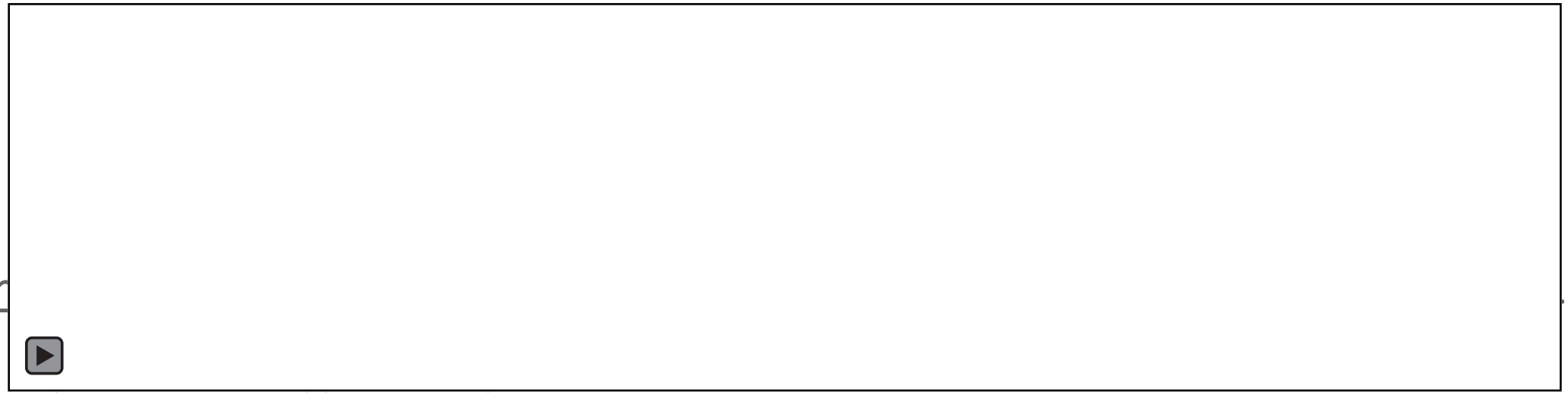
Modest q: to be Markovian or not to be Markovian ?

Potential pros: representationally, including memory is strictly more general.

(So maybe, if training works, it should be only better?)

Potential cons: many natural ways to add memory are computationally expensive.

Er



Maybe sensible, since the PDE we are solving is "Markovian"

$$\partial_t u(x,t) = \mathcal{L}_t\big(u(x,t)\big) \ ?$$

⚠

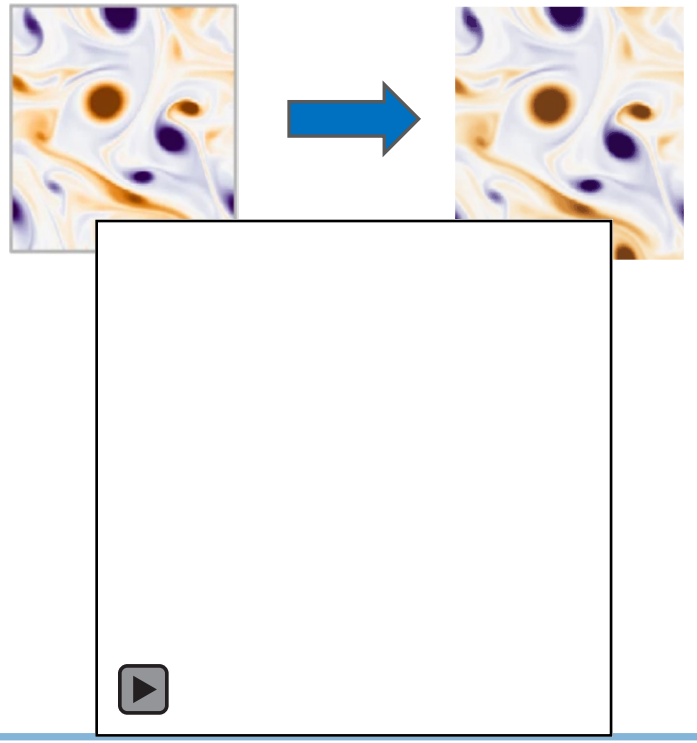
Only if the initial state is "fully observed"!

Mori-Zwanzig-Nakajima formalism

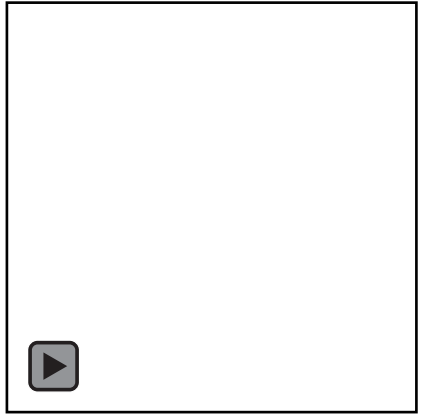
Even for linear operators $\mathcal{L}$:

If we're observing the initial conditions partially,
(e.g. due to noise, aliasing, equipment imperfections, ..)

the best approximation on the
"observable subspace" is non-Markovian.



Mori-Zwanzig-Nakajima formalism



More formally (Nakajima '58, Zwanzig '60):

Suppose $\partial_t u = \mathcal{L}u$ and we're approximating the system in the image of projection $\mathcal{P}$.

Since $u = \mathcal{P}u + (I - \mathcal{P})u$, denoting $\mathcal{Q} \coloneqq (I - \mathcal{P})u$, a simple calculation yields:

$$\partial_t \mathcal{P}u(t) = \underbrace{\mathcal{P}\mathcal{L}\mathcal{P}u(t)}_{\text{"Markovian" approximation}} + \underbrace{\mathcal{P}\mathcal{L}\int_0^t \exp\{\mathcal{Q}\mathcal{L}(s-t)\}\mathcal{Q}\mathcal{L}\mathcal{P}u(s)ds}_{\text{Convolution w/ "memory kernel"}} + \underbrace{\mathcal{P}\mathcal{L}\exp(\mathcal{Q}\mathcal{L}t)\mathcal{Q}u_0}_{\text{Unobservable}}$$

The magnitude of the "memory correction" can be arbitrarily large!

The value of memory

"Memory-free" soln

"Memory-corrected" soln

Proposition (informal). For any $B > 0$, there exist $\mathcal{L}, \mathcal{P}$ and $u(0)$, such that:

If $u_1(t)$ solves $\partial_t \mathcal{P} u_1(t) = \mathcal{P}\mathcal{L}\mathcal{P} u_1(t)$,

And $u_2(t)$ solves $\partial_t \mathcal{P} u_2(t) = \mathcal{P}\mathcal{L}\mathcal{P} u_2(t) + \mathcal{P}\mathcal{L}\int_0^t \exp\{\mathcal{Q}\mathcal{L}(s-t)\}\mathcal{Q}\mathcal{L}\mathcal{P} u(s) ds$, we have:

$$\left|\left|u_1(t) - u_2(t)\right|\right| \gtrsim B \, ||u_1(t)|| \qquad \left|\left|u_1(t) - u_2(t)\right|\right| \gtrsim Bt \exp(\sqrt{2}B \, t)$$

Idea: In Fourier basis $\{e_i\}_{i \in \mathbb{N}_0}$, take $\mathcal{L}$ such that: $\mathcal{L} e_n = n^2 e_n + B(e_{n-1} + e_{n+1})$

Take $\mathcal{P}$ to project to $\mathrm{span}\{e_0, e_1\}$.

$\mathcal{L}$ "leaks" information outside of $\mathcal{P}$ which is dropped by $\mathcal{P}\mathcal{L}\mathcal{P}$, but recovered by $\mathcal{Q}\mathcal{L}\mathcal{P}$ term.

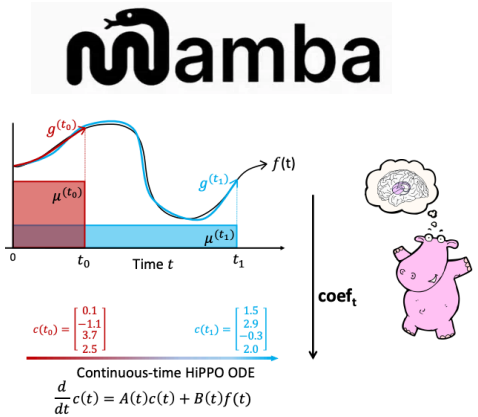
(Neurally) operationalizing memory with SSMs

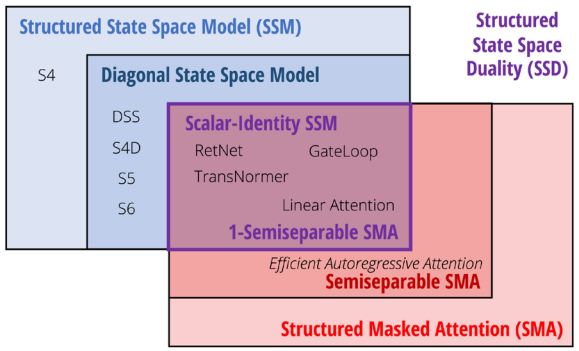
Structured state space models (S4, Gu et al '21) parametrize (learned) linear dynamical systems & are conducive to a highly parallel-efficient "convolutional" evaluation:

Sequential description:

State $\leftarrow$ $h_{t+1} = Ah_t + B\,x_t$ $\rightarrow$ Input

Predicted output $\leftarrow$ $y_{t+1} = C\,h_t$





Convolutional description:

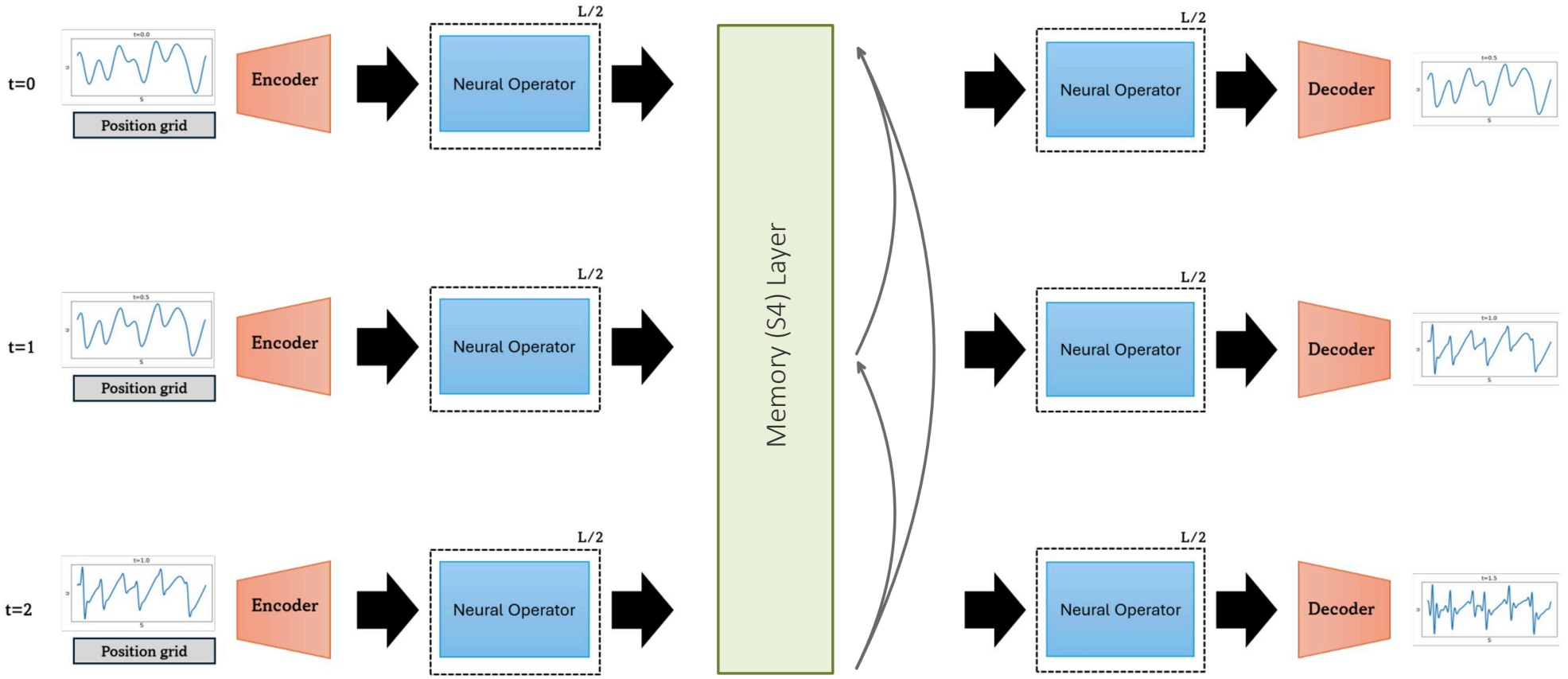
$$y_t = \sum_{s=0}^{k} K_{t-s}\,u_s = (K \star u)_t$$

$$K_t = C\,A^t\,B$$

"Structured" A for fast parallel eval of K_t

Convolutions are sped up w/ FFT.

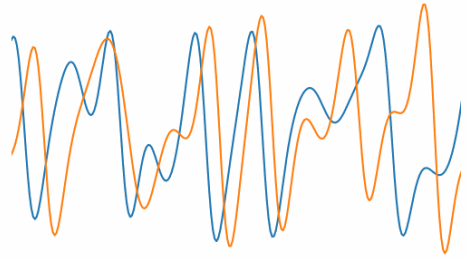
(Neurally) operationalizing memory with SSMs



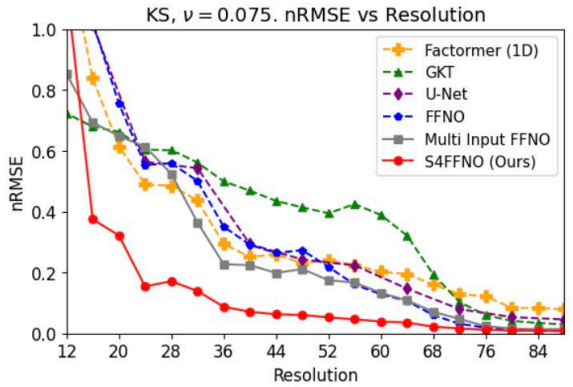
Memory layer has access to all the past states,
but can be parallelized via convolutional interpretation.

Neural operator is Factorized Fourier Neural Operator (Tran et al '21), though in principle any combination of neural operator & sequence mixer can be used.

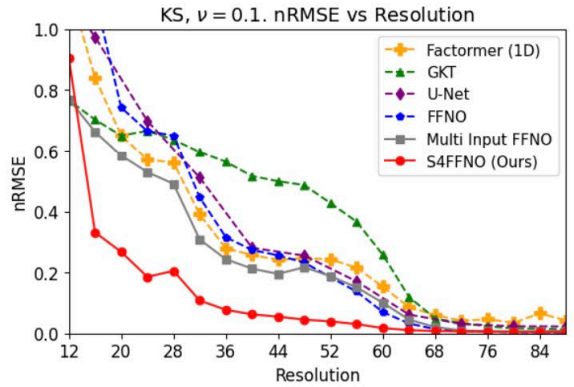
Results I: A study in low resolution



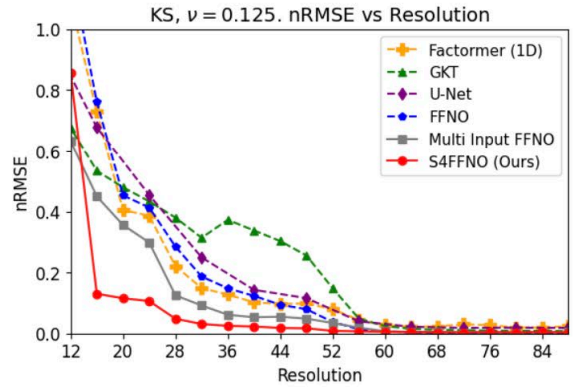
Kuramoto-Sivashinsky (1D) with different viscosity ν: $\frac{\partial u}{\partial t} + \frac{\partial^2 u}{\partial x^2} + \nu \frac{\partial^4 u}{\partial x^4} + \frac{u \partial u}{\partial x} = 0$



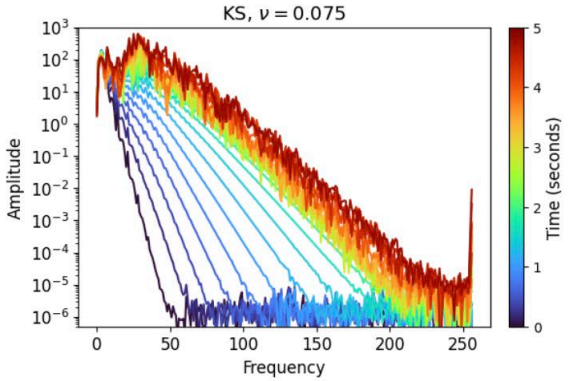
(a) $\nu = 0.075$

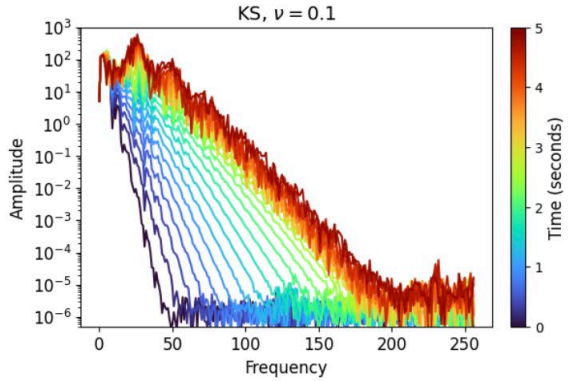


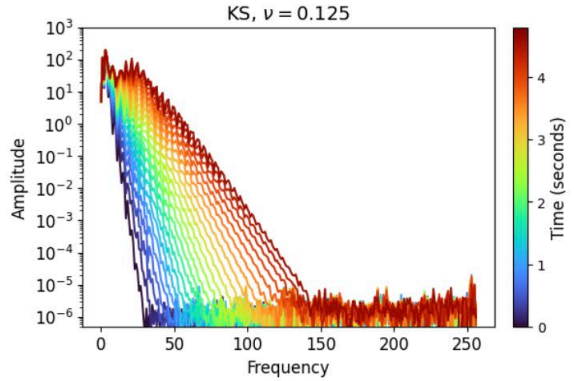
(b) $\nu = 0.1$



(c) $\nu = 0.125$



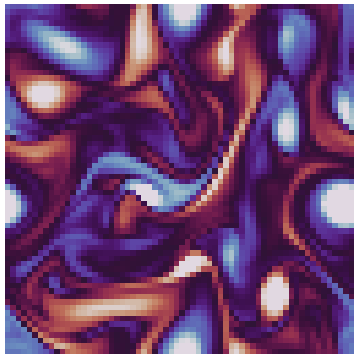




Advantage is larger for smaller viscosity.

(Smaller viscosity tends to introduce more of higher Fourier freqs later in time)

Results II: A study in observation noise

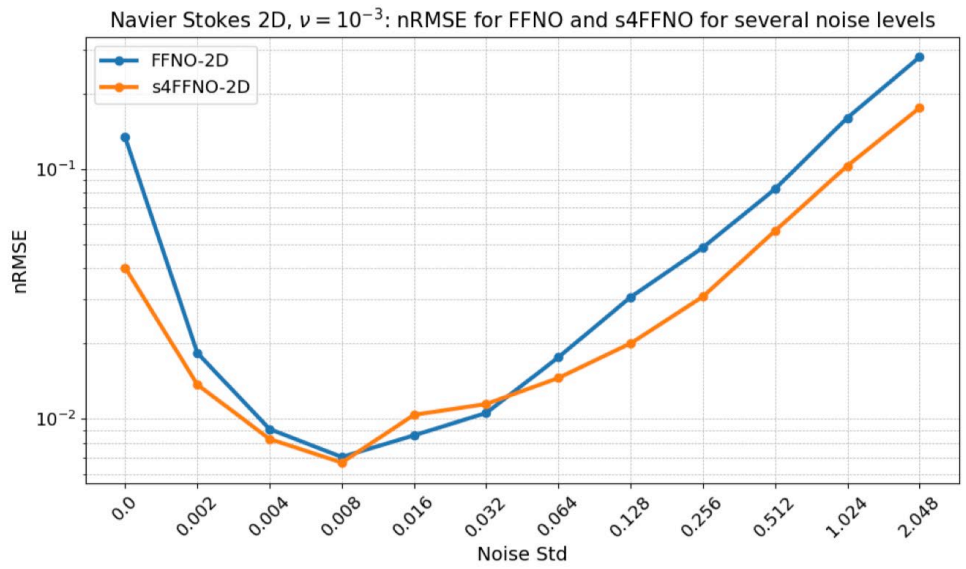


Navier-Stokes (2D) with viscosity ν:

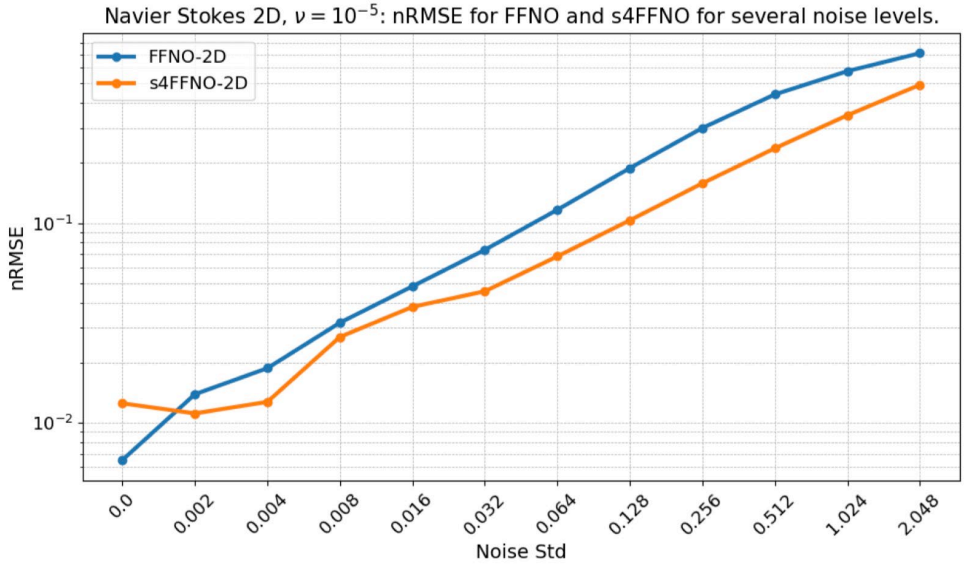
$$\frac{\partial \omega(x,t)}{\partial t} + u(x,t) \cdot \nabla \omega(x,t) = \nu \Delta \omega(x,t) + f(x)$$

$$\nabla \cdot u(x,t) = 0$$

$$\omega(x,0) = \omega_{0(x)}$$



(a) $\nu = 10^{-3}, T = 16s, N_t = 32$



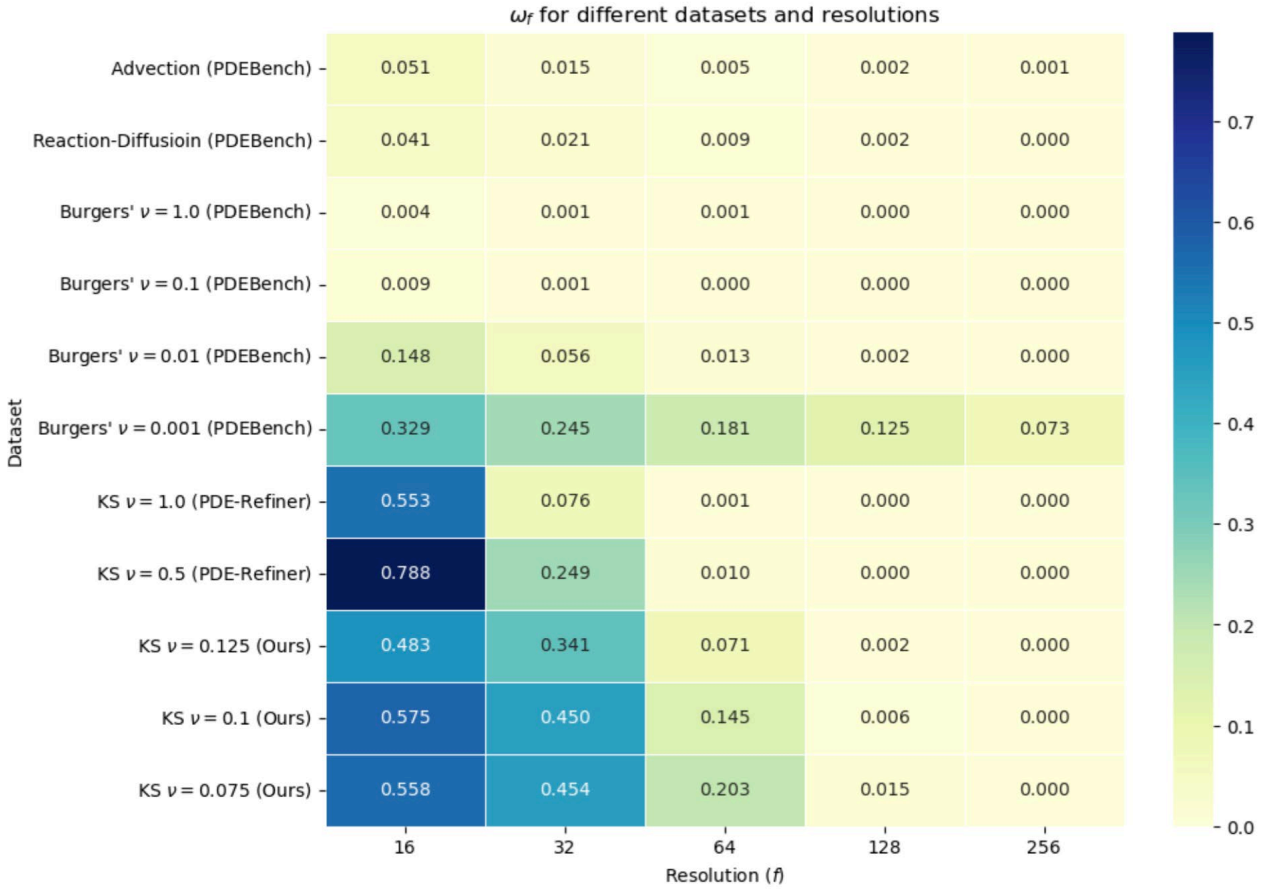
(b) $\nu = 10^{-5}, T = 3.2s, N_t = 32$

ΔRMSE at the final time T on Navier Stokes dataset under different noise standard deviations

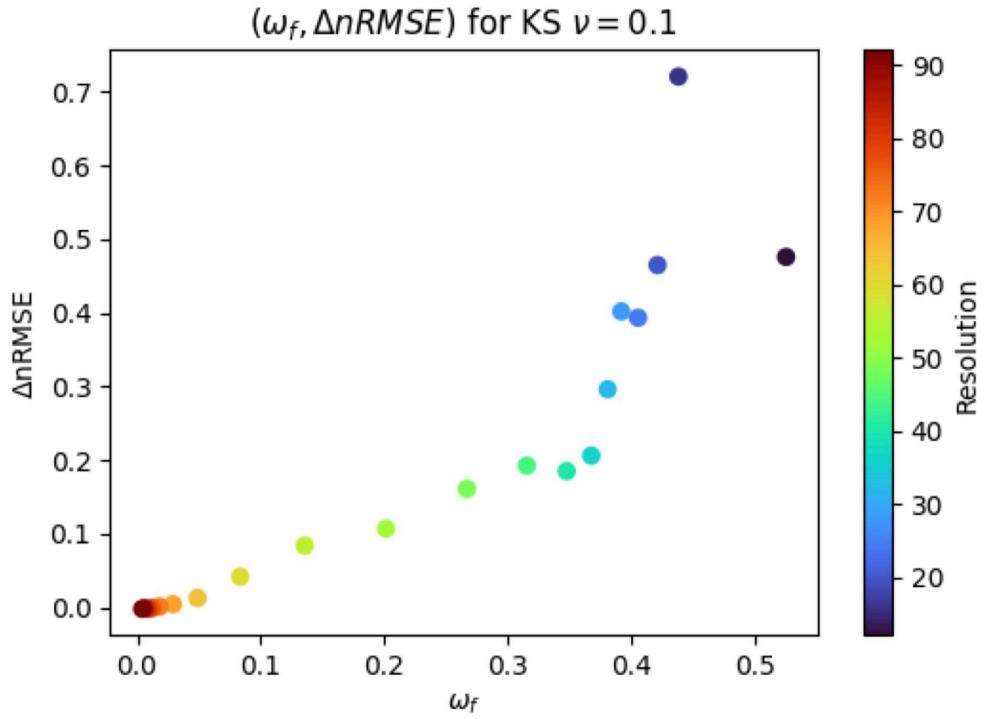
Would common benchmarks catch this?

Consider the ”relative energy” of the unobserved modes at resolution f: (averaged over time & trajectories)

$$\omega_f = \frac{\sum_{|n|>\frac{f}{2}} |a_n|^2}{\sum_n |a_n|^2}$$



The state of common benchmarks



ΔnRMSE = |S4FFNO nRMSE – FFNO nRMSE|

Outlook

Computational challenges:

SSMs chosen for computational reasons: are there (provable) tradeoffs w/ Transformers?

Formalisms to capture “parallelism to accuracy” tradeoffs?

When does stochasticity (i.e. generative modeling) help?

”Inference-time compute” strategies:

Ways to smoothly trade-off accuracy for runtime at inference time?

Better benchmarks:

Many current datasets are solved by very simple approaches, so unclear we are extracting signal (cf. MNIST and CIFAR era in images.)

“PDE solving” is a very heterogeneous concept: we probably need more tailored, higher quality benchmarks for each domain.

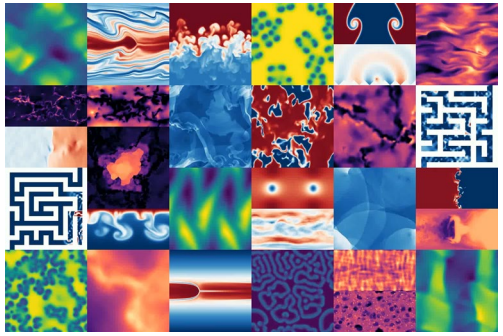
Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations

Nick McGreivy & Ammar Hakim

Nature Machine Intelligence **6**, 1256–1269 (2024) | Cite this article

TheWell

The Well: 15TB of Physics Simulations



References

https://arxiv.org/abs/2410.09867

Towards characterizing the value of edge embeddings in Graph Neural Networks (ICML 2025)

https://arxiv.org/abs/2409.02313

On the Benefits of Memory for Modeling Time-Dependent PDEs (ICLR 2025)