Machine Learning: Bane or Boon for Control?

2023 BODE PRIZE LECTURE

MIROSLAV KRSTIC

President Parisini, President Egerstedt, Past President Baillieul: thank you for choosing me to give the 35th Bode Lecture.

It is not without at least a little bit of irony that the first Bode Lecture on machine learning (ML) and control be given by somebody as "old school" control theoretic as they get. However, if I do not address the elephant in the room, at this time, did I deserve this opportunity, at this time?

Digital Object Identifier 10.1109/MCS.2024.3402581
Date of current version: 19 July 2024

BRIDGE BETWEEN 1991 AND 2023 BODE LECTURES

Among the extraordinary Bode Lectures from the past, the one that has a special place in my heart is, naturally, my advisor Petar Kokotovic's 1991 Bode Lecture. His legendary "The Joy of Feedback: Nonlinear and Adaptive" [1]. Pivotal in the development of our field.

Petar used his lecture to promote the work of several early and midcareer researchers in nonlinear and adaptive control. Nobody benefited, I feel, more than I did.

I had arrived in Santa Barbara in July that year. I submitted my first paper in October, the "tuning functions" design of

1066-033X/24©2024IEEE

Summary

It is no longer a question whether machine learning (ML) should be incorporated into the control toolbox but only how. Some high-profile artificial intelligence pioneers caution their colleagues about the use of ML for control in a manner where physical modeling is unnecessarily bypassed and where the distinguishing requirements of feedback systems, stability and safety, are not guaranteed. I propose a use of ML that leverages the control community's heritage of rigorous, certificate-bearing control designs. A game-changing "supporting and empowering role" is entrusted to ML, in automating and accelerating by several orders of magnitude the implementation of model-based control designs. The benefit of such a control + ML blend, where both control and ML get to perform complementary roles for which they are best suited, is nowhere as evident as for hard-to-control systems modeled by partial differential equations. I employ, in partial differential equation (PDE) control, the recent breakthroughs in deep learning approximations of not functions but function-to-function mappings (nonlinear operators), the so-called "neural operators." With neural operators, entire PDE control methodologies are encoded into what amounts to a function evaluation, leading to a thousandfold speedup in real-time implementation, while retaining the stability guarantees. Applications range from traffic and epidemiology to manufacturing, energy generation, and supply chains.

adaptive nonlinear controllers. In December, Petar was kind enough to include this design as the focal point of his Bode Lecture.

So, on with my lecture.

NO RETROSPECTIVES TODAY

Making up my mind regarding the topic for the lecture has caused me more than a little apprehension, over several months.

Do I share my excitement about the advances in partial differential equation (PDE) control for traffic, epidemiology, and social networks, under the title "PDE Backstepping in the Century of Sociotechnical Systems"?

Or do I go with a talk titled "Inverse Optimal Safety: Kalman Meets Nagumo," where I review results, mine and my students', on what now goes by the name "control barrier functions" (CBFs)? Some of these results go back to 2006 (backstepping for CBFs of relative degree higher than one, called "nonovershooting control" back then). The newest results expand QP safety filters via Kalman-like inverse optimality, extend exponential safety to prescribed-time safety, and generalize CBF designs from ordinary differential equations (ODEs) to PDEs.

Or do I just go with what everyone I had consulted with suggested that I speak on, extremum seeking? The sheer opportunity for a double entendre in the title of the talk, "Extremum Seeking 101," was tempting, 101 meaning that the talk both covers the basics and is occasioned on the one hundred and first anniversary of the invention of ES (see "Extremum Seeking 101: Too Big of a Bite").

CONTROL AND ML: FROM RIVALS TO PARTNERS

After much consideration, I decided to create a Bode Lecture that addresses the new generation of control researchers, a lecture that, rather than looking back, hopefully opens a new horizon.

In the face of initially perceived "threats" that data-based tools may pose to theory-supported control designs, the past few years have been a testimony to the brilliance and resourcefulness of so many individuals in our community in finding synergies between control and learning. The results thus far include learning-based model predictive control (MPC); learning Lyapunov functions, control-Lyapunov functions (CLFs), and CBFs; learning controllers from data; reinforcement learning (model-based and model-free); learning in games and multiagent systems (MASs); and other topics.

For a fairly extensive list of such results and their authors, as of 2023, please see the extensive introduction in [2].

I propose here a use of ML, which, I believe, has not been envisioned before. Our community has a formidable heritage of rigorous control designs. If you think about it, what is common to our control designs is that they are all complex nonlinear mappings from models to feedback gains or feedback laws:

$$\boxed{\text{model} \mapsto \text{feedback}}.$$

So, I suggest, let ML *learn* the complete maps from models to model-based designs, automate the maps' deployment in control algorithms, and reduce the control algorithms' computational burden.

With ML and the powerful GPUs designed for neural networks (NNs), a speedup in computation in the three orders of magnitude range may be achieved. This is a game changer for control of PDEs, delay systems, and nonlinear systems.

EXAMPLE OF A MODEL-TO-FEEDBACK MAP

What do I mean by a *model-to-feedback map*?

The simplest example is the linear quadratic regulator (LQR), which is the problem of designing a control law for a linear system

$$\dot{x} = AX + Bu$$

to minimize the cost,

$$\int_0^\infty (|x(t)|^2 + |u(t)|^2)dt.$$

We all know that this problem's solution is the feedback law

$$u = -B^T Px$$

where P is the solution to the Riccati equation

$$\boxed{PA + A^T P - PBB^T P = -I}. \tag{1}$$

However, please do not think of P only as a matrix that solves the Riccati equation for one pair (A, B). Imagine the entire

Extremum Seeking 101: Too Big of a Bite

Extremum seeking was the most appealing candidate topic for a backward-looking Bode Lecture. I have worked on extremum seeking for a quarter-century now and, with many coauthors, introduced *the stability guarantees, source seeking, stochastic extremum seeking, Newton extremum seeking, Nash equilibrium seeking, and extremum seeking for delay systems and partial differential equations*.

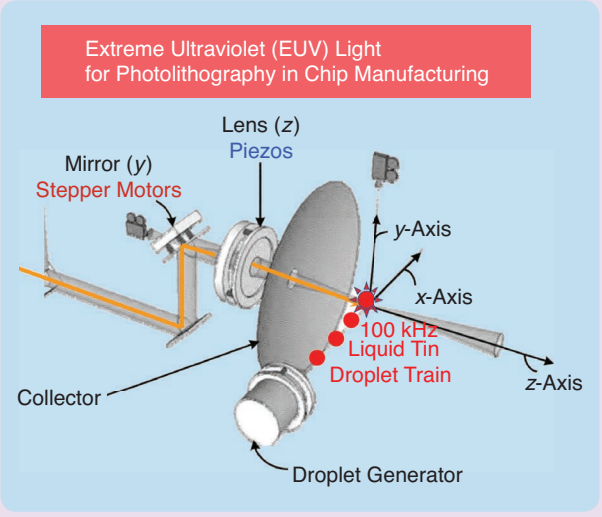

FIGURE S1 An application of extremum seeking with the highest industrial impact to date: EUV light in photolithography. Extremum seeking has stabilized the process in which laser pulses hit liquid tin droplets at the rate of tens of kilohertz, enabling the increase of chip density by two orders of magnitude and the creation of a US$10 billion per year industry (as of 2023) in light sources for photolithography.

Who would have thought, in 1997, that the proof of stability [S1], driven purely by curiosity, would bring the subject to life and lead, as of 2023, to

- 18,000 papers/articles
- nearly 3,000 patents, currently produced at the rate of more than one patent *per day*?

You can see why I wimped out of the task of surveying thousands of publications in a one-hour lecture.

However, before I move on from extremum seeking, let me mention to you its greatest industrial achievement: extreme ultraviolet (EUV) light in chip manufacturing, depicted in Figure S1. A rapid fire of laser pulses, shown with the orange lines, blasts a rapid fire of droplets of a liquid metal, tin, shown with the red dots, at a rate of 100,000 hits per second. Extremum seeking made the EUV technology *stable*, after all else had been tried.

EUV has reduced, over the last decade, the chip feature size by two orders of magnitude. EUV is now a US$10 billion per year industry in EUV (photolithography light source) equipment alone. Stable EUV was enabled by a 2013 patent of my student Paul Frihauf and his collaborators at Cymer, Inc. [S2], based on an article of another team of students in my lab [S3], a decade earlier.

REFERENCES

[S1] M. Krstic and H.-H. Wang, "Design and stability analysis of extremum seeking feedback for general nonlinear systems," in *Proc. 36th IEEE Conf. Decis. Control*, Piscataway, NJ, USA: IEEE, 1997, vol. 2, pp. 1743–1748, doi: 10.1109/CDC.1997.657809.

[S2] P. Frihauf, D. J. Riggs, M. R. Graham, S. Chang, and W. J. Dunstan, "System and method to optimize extreme ultraviolet light generation," U.S. Patent 8598552B1, 2013.

[S3] J.-Y. Choi, M. Krstic, K. B. Ariyur, and J. S. Lee, "Extremum seeking control for discrete-time systems," *IEEE Trans. Autom. Control*, vol. 47, no. 2, pp. 318–323, Feb. 2002, doi: 10.1109/9.983370.

nonlinear mapping $P = \mathcal{P}(A, B)$, namely, $\mathcal{P} : (A, B) \mapsto P$, which is implicitly defined by the Riccati equation.

This mapping is not only continuous in (A, B) but real analytic, as proven by David Delchamps [3], while a Ph.D. student at Harvard University with Chris Byrnes (the 2008 Bode Lecturer).

DEEP NEURAL OPERATORS

Neural operators are NN approximations not of functions but of nonlinear infinite-dimensional mappings of functions into functions. The excitement about neural operators came, in recent years, from their speedup of the solution of the notoriously hard nonlinear Navier–Stokes PDEs by three orders of magnitude.

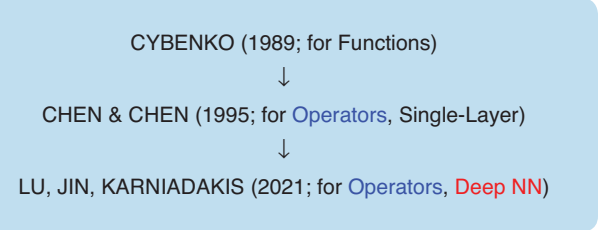

FIGURE 1 The progression of neural approximation of nonlinear mappings, from functions to operators, using deep NNs.

Let me review, with the help of Figure 1, the pathway to these deep neural operators. The first discovery of approximability by NNs was for nonlinear functions, not operators, by Cybenko in 1989 [4]. The extension by Chen and Chen in 1995 [5] to the approximation of nonlinear operators with single-layer NNs was barely noticed. Today's great impact of neural operators is due to the theory developed for *multilayer*, or deep, neural operators, a couple of years ago, by Lu, Jin, and Karniadakis [6], as well as by other teams of researchers developing a variety of methods.

The theory of neural operators, as universal approximators, has a tremendous potential for control. Let me give you a two-sentence digest, stripped of burdensome notation and technical details, of the landmark universal approximation theorem by Karniadakis and his team [6].

Let $\mathcal{G}$ denote a (nonlinear) operator. If $\mathcal{G}$ is continuous, then, for all $\epsilon > 0$, there exists a (deep) neural operator $\hat{\mathcal{G}}$ such that

$$\boxed{|\mathcal{G}(u) - \hat{\mathcal{G}}(u)| < \epsilon}$$

for all inputs $u \in \mathcal{U}$, a compact set of continuous functions.

In even simpler words, any continuous operator can be approximated by an NN, to arbitrarily tight accuracy, uniformly over a compact set of continuous input functions.

The approximating "deep neural operator" $\hat{\mathcal{G}}$ has been branded, by the authors, as a DeepONet.

OUTLINE OF THE LECTURE

In the rest of the lecture I will show you several uses of neural operators in control of PDEs. The need for a speedy approximation of feedback algorithms is particularly pronounced in control of such complex and high-dimensional dynamical systems as PDEs. Neural operators are similarly applicable in control of nonlinear ODEs, as well as in MASs and games. You may think of PDEs as MASs with infinitely many agents.

I will show you examples of the power of neural operators in control of both hyperbolic and parabolic PDEs. For pedagogical reasons, I start with *hyperbolic PDEs*, for which the operator that arises in the feedback design involves functions of only one spatial variable. In *parabolic PDEs*, functions of two variables necessarily arise.

I will show you examples of both mappings whose outputs are control *gains* and mappings whose outputs are the control *values*.

I will close the lecture with two application domains where a rapid recomputation of complex feedback laws, in real time, is critical. Rapid recomputation is critical in *gain scheduling* for nonlinear PDEs and *adaptive control* for PDEs with unknown functional coefficients.

HYPERBOLIC PDES: A PEDAGOGICAL START

I first want to ease you into the notion that the design of control gains for PDEs is a design of function-to-function operators that are *nonlinear*, even for PDEs that are linear. This is easiest to see on a class of hyperbolic PDEs in which both the model coefficient functions and the feedback gain functions have only a single (spatial) argument.

Transport PDE With Destabilizing Recirculation

Before I present stabilization of some unstable PDEs to you, let me introduce you to the baby among PDEs, the *transport PDE*. The transport PDE,

$$u_t = u_x$$

has one derivative in time t and one in space x (see "PDE Notation"). The transport PDE is nothing more than a PDE representation of a pure delay. As depicted in Figure 2, the input signal $U(t)$ is delayed by a unit of time, over a unity distance.

The simplest example of an *unstable* hyperbolic PDE has the state $u(0, t)$, from the outlet $x = 0$, fed back into the transport domain:

$$u_t(x, t) = u_x(x, t) \boxed{+ \beta(x)\, u(0, t)}. \tag{2}$$

The functional coefficient $\beta(x)$, highlighted in red, is called *recirculation*. This plant has a continuum of positive feedback loops, one at each spatial location x.

You can see the destabilizing effect of positive feedback in the simulation plot in Figure 3. The growing oscillations in this plot are similar to stop-and-go oscillations in

PDE Notation

Figure S2 is helpful in getting acquainted with the notation common for PDEs. The state of a PDE is not a vector but a function, a vector of infinite dimension. In PDEs we cannot use the symbol x for the state variable, because x is commonly used for the spatial coordinate, the continuum analog of the index of an entry of a state vector. Based on fluid dynamics, where the state is usually the fluid velocity, u, the PDE state is typically denoted by $u(x, t)$, at a location x and at time t. In PDE control, the actuation is usually at the boundary of the domain. We denote the input by U since it often has the same physical nature as the state u; it just acts at the boundary.

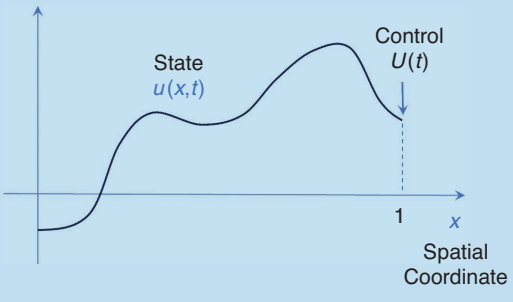

FIGURE S2 PDE notation: the state is denoted by u, the input by U, the spatial coordinate by x, and the time by t. The curve shows the state as a function of space x, at a given time t. If unfamiliar with the notion of a system state being a function, one may regard the state u as an infinite-dimensional vector, with x being a "continuum index" of the entries of the vector u.

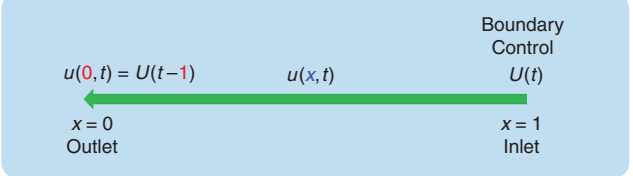

FIGURE 2 A (leftward) transport PDE representation of a unity pure delay, where $u(0,t) = U(t-1)$.

freeway traffic flow, which is modeled by PDEs somewhat more complex than (2).

The control in the transport PDE with recirculation appears at a boundary, $x = 1$, the inlet of the transport process, namely,

$$u_t(x,t) = u_x(x,t) + \beta(x)\,u(0,t), \quad x \in [0,1) \quad \text{recirculation}$$
$$u(1,t) = U(t) \quad \text{boundary control.}$$

The challenge for the control design for this system is that the positive feedback loops are closed throughout the spatial interval $[0, 1)$, while the control acts only at the *boundary* $x = 1$ of the interval, as depicted in Figure 4. So control U cannot simply cancel *any* of these positive feedback loops.

At first, it seems impossible for the scalar control U (at the boundary) to *unlink* the continuum of recirculations (throughout the domain). However, it is possible to unlink them, using PDE backstepping. In fact, that is what PDE backstepping was invented for. For PDE backstepping, this example is the simplest nontrivial benchmark.

PDE Backstepping Design

Backstepping employs a *spatial* convolution transformation of the state, $u \mapsto w$, with a convolution kernel function k, underbraced in red:

$$\boxed{w(x,t) = u(x,t) - \int_0^x \underbrace{k(x-y)}\, u(y,t)\,dy}. \tag{3}$$

The kernel k depends, in this pedagogical example, on a single spatial variable, $x - y$.

The backstepping transformation, when set to zero at the actuated boundary, provides the control law, with the same kernel k, whose argument x is set to 1, serving as the gain for the controller:

$$U(t) = \int_0^1 \underbrace{k(1-y)}\, u(y,t)\,dy. \tag{4}$$

The control law (4), along with the transformation of the state, (3), eliminates the destabilizing recirculation:

$$w_t = w_x + \cancel{\beta(x)\,u(0,t)} \tag{5}$$
$$w(1,t) = 0. \tag{6}$$

The resulting closed-loop system, called a *target system*, is the pure delay, with its zero input emphasized in red. The target system is not only exponentially stable but finite-time stable. Figure 5 shows how the state $u(x, t)$ settles to zero in finite time and how the control, shown by the red curve at the boundary $x = 1$, achieves this stabilization goal.

Backstepping Kernel Integral Equation and Nonlinear Operator

Let us recall the backstepping transformation (3) and the control law (4). The key player in both of them is the kernel function k, underbraced in red in both equations. I have not defined the function k yet. It is k, in fact, that has to be designed.

For the kernel function k to eliminate the recirculation in the target system (5), (6), kernel k must satisfy the following integral equation, for a *given* recirculation function β:

$$k(x) = -\beta(x) + \int_0^x \beta(x-y)\,k(y)\,dy, \qquad x \in [0,1]. \tag{7}$$

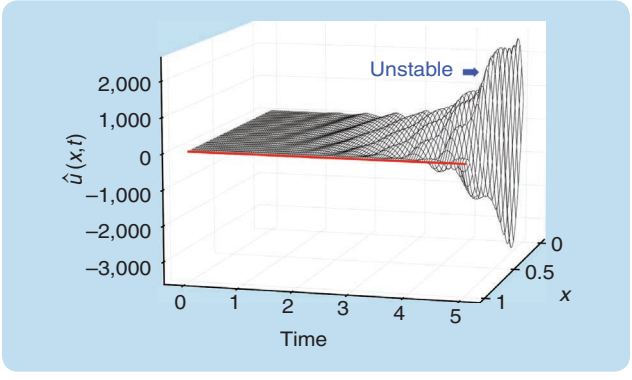

FIGURE 3 The transport PDE with recirculation, $u_t = u_x + \beta(x)\,u(0,t)$, has a continuum of positive feedback loops, recirculating the outlet state $u(0, t)$ back into the domain, which results in instability. In this simulation, only a pair of complex eigenvalues with positive real parts arises, resulting in a growing oscillation of the plant state in both space and time.

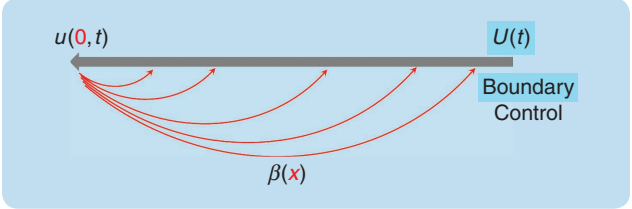

FIGURE 4 A continuum of positive-feedback recirculations in the transport PDE with recirculation, $u_t = u_x + \beta(x)\,u(0,t)$. None of these recirculations can be unlinked (cancelled) using scalar boundary actuation, $u(1,t) = U(t)$..

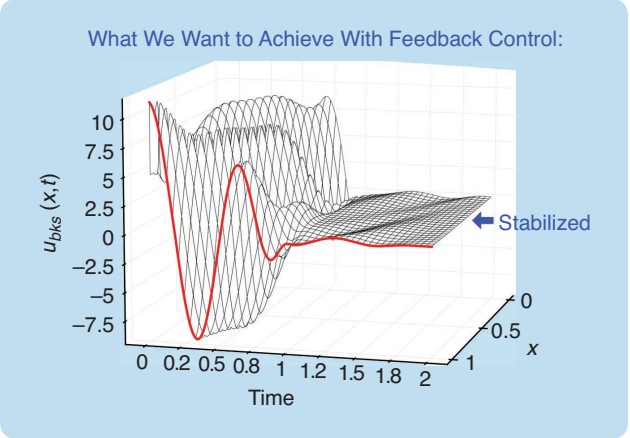

FIGURE 5 Backstepping feedback (4), whose value is highlighted in red, at the boundary $x = 1$, stabilizes the transport PDE with recirculation (2).

This integral equation serves the same purpose of a "gain design equation" for the PDE backstepping design as the Riccati equation (1) serves for the LQR design. Let us now introduce a compact, convolution notation for this integral equation, and let us suppress the spatial variable:

$$\boxed{k = -\beta + \beta * k}.$$

This is a linear equation in k, for a given β. Since a convolution appears in the equation, let us apply the Laplace transform $\mathscr{L}$, but in space (not in time), and solve for k:

$$k = \mathscr{L}^{-1}\left\{\frac{-\mathscr{L}\{\beta\}}{1-\mathscr{L}\{\beta\}}\right\}.$$

While this expression is not explicit in β, it is clear that it is nonlinear in β.

To summarize, the mapping $\mathcal{K}: \beta \mapsto k$ defined by the linear integral equation (7) is a *nonlinear operator*. Interestingly, the operator $\mathcal{K}$ happens to be its own inverse. Functions with this property are called involutions, and matrices with this property are called involutory. So, I call $\mathcal{K}$ an *involution operator*.

Learning the Nonlinear Operator $\mathcal{K}$

If one solves the integral equation (7) for a rich enough collection of functions $\beta_i(x)$, one obtains a training set of gains $k_i(x)$, as in the diagram in Figure 6. With this training set, one can train a deep NN $\hat{\mathcal{K}}$, which approximates the operator $\mathcal{K}$.

However, can $\hat{\mathcal{K}}$ approximate $\mathcal{K}$ *as closely as we like*? The answer is affirmative, using Karniadakis's DeepONet universal approximation theorem, provided we prove that this nonlinear operator $\mathcal{K}: \beta \mapsto k$ is continuous. In fact, we prove that $\mathcal{K}$ is a bit more Lipschitz.

Lemma 1 (Lipschitzness of Operator $\mathcal{K}$)
$\mathcal{K}$ is Lipschitz, with a Lipschitz constant no greater than e^{3B}, on any compact set of input functions $\|\beta\|_\infty \le B$, for any $B > 0$.

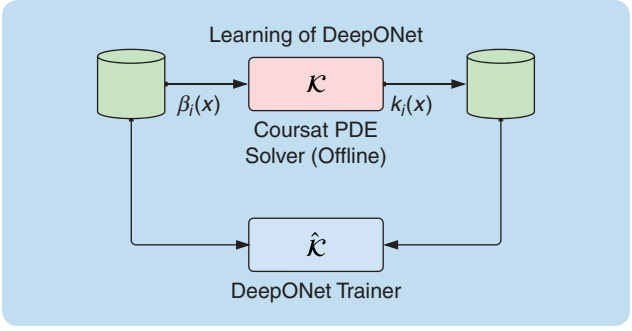

FIGURE 6 The process of constructing a neural operator (DeepONet) $\hat{\mathcal{K}}$, which approximates the exact operator $\mathcal{K}$, by solving, numerically, the integral (7) for a rich enough collection of functions $\beta_i(x)$, obtaining a training set of gains $k_i(x)$ and training an NN for $\hat{\mathcal{K}}$.

The Lipschitzness of the nonlinear operator $\mathcal{K}$ is not a trivial property. I am not talking about the Lipschitzness of the operator's output function $k(x)$, in relation to its scalar spatial argument x. I am establishing the Lipschitzness of the entire operator $\mathcal{K}(\beta)$ in relation to its functional input β.

So, having proven the Lipschitzness of $\mathcal{K}$, the DeepONet theorem guarantees that a large enough NN approximates the operator $\mathcal{K}$ arbitrarily closely, uniformly in the input functions beta.

Theorem 1 (DeepONet Approximation of $\mathcal{K}$)
For all $B, \epsilon > 0$, there exists a DeepONet $\hat{\mathcal{K}}$ satisfying

$$\boxed{|\mathcal{K}(\beta)(x) - \hat{\mathcal{K}}(\beta)(x)| < \epsilon}$$

for all $\|\beta\|_\infty \le B$ and $x \in [0, 1]$.

Figure 7(a) shows one illustrative input–output pair $(\beta, \hat{k})$ of the approximate operator $\hat{\mathcal{K}}$. A comparison of the magnitudes on the vertical axes of the two plots shows that the approximation error in Figure 7(b) is two orders of magnitude smaller than the exact operator output $k = \mathcal{K}(\beta)$ being approximated. So the DeepONet approximation $\hat{\mathcal{K}}$ of the exact operator $\mathcal{K}$ is good.

Are DeepONet Gains Stabilizing?

It is reassuring that the gain approximation in Figure 7 is good; however, approximating control gains alone is not of interest to us. Since the approximated gains are to be used for feedback, the question is whether stabilization, guaranteed under exact gains, survives the approximation.

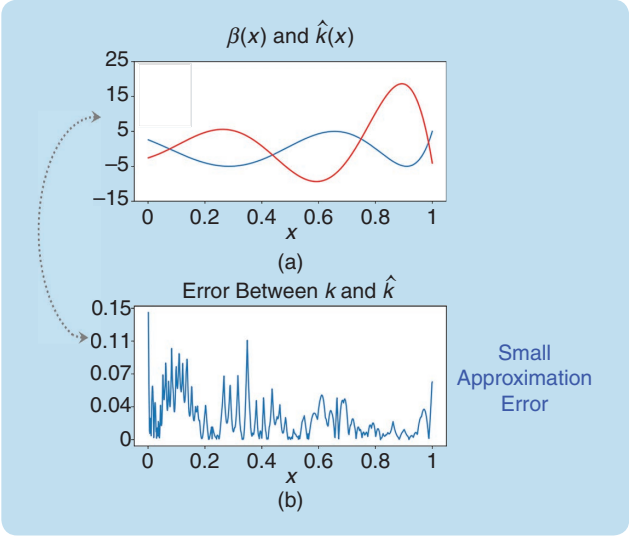

FIGURE 7 The training process in Figure 6 results in a good DeepONet approximation $\hat{\mathcal{K}}$ of the exact operator $\mathcal{K}$. For (a) an illustrative input–output pair $(\beta, \hat{k})$, (b) the approximation error is two orders of magnitude smaller than the kernel k being approximated by $\hat{k} = \hat{\mathcal{K}}(\beta)$.

To answer this question, we plug the approximate gain $\hat{k} = \hat{\mathcal{K}}(\beta)$ into the backstepping feedback law,

$$U(t) = \int_0^1 \underbrace{\hat{k}(1-y)}_{\hat{\mathcal{K}}(\beta)(1-y)} u(y,t)\,dy \tag{8}$$

and then employ this feedback as a boundary input, $u(1,t) = U(t)$. Then we proceed to study closed-loop stability.

We already know that if the exact kernel k is used in the backstepping transformation (3) and in the control law (4), the exact target system (5), (6) is obtained, resulting in finite-time stability. However, when the approximated feedback (8) is employed, along with the approximated $\hat{k} = \hat{\mathcal{K}}(\beta)$ in a backstepping transformation, meaning that (3) is replaced by

$$\hat{w}(x,t) = u(x,t) - \int_0^x \hat{k}(x-y)\,u(y,t)\,dy$$

a perturbation appears in the target system

$$\begin{aligned}\hat{w}_t(x,t) &= \hat{w}_x(x,t) + \overbrace{\delta(x)\hat{w}(0,t)}^{\text{perturbation}}\\ \hat{w}(1,t) &= 0\end{aligned}$$

where the overbraced perturbation is vanishing because it has the outlet state $\hat{w}(0,t)$ as a factor, and the perturbation coefficient δ is

$$\delta = (-1+\beta*)\underbrace{(\mathcal{K}(\beta) - \hat{\mathcal{K}}(\beta))}_{\text{bounded by }\epsilon}.$$

The boundedness of $\mathcal{K}(\beta) - \hat{\mathcal{K}}(\beta)$ by an approximately small ϵ follows from Theorem 1. In plain language, we can make the perturbation coefficient δ as small as we like with a large enough NN $\hat{\mathcal{K}}$.

Lemma 2 (Lyapunov Estimate for Target System)

For all neural operators $\hat{\mathcal{K}}$ with $\epsilon \in (0, (2/e(1+B)))$, the Lyapunov functional

$$V(t) = \int_0^1 e^{2x}\hat{w}^2(x,t)\,dx$$

satisfies the bound

$$\boxed{V(t) \le V(0)e^{-c^*t}} \qquad c^* > 0$$

for all $\|\beta\|_\infty \le B$.

What is the price for this theoretical result? From the reciprocal dependence of the upper bound on ϵ in Lemma 2 with respect to the bound B on the destabilizing recirculation function β, we note that the more unstable the open-loop plant is, the larger the NN that is needed for the kernel approximation.

Finally, we get to our main result.

Theorem 2 (Closed Loop Is *Robust* to DeepONet)

For any system with $\|\beta\|_\infty \le B$, all controllers with $\hat{\mathcal{K}}$ trained for any $\epsilon \in (0, (2/e(1+B)))$ guarantee

$$\boxed{\|u(t)\| \le Me^{-c^*t/2}\|u_0\|} \qquad \forall t \ge 0$$

with overshoot coefficient

$$M = (1 + (\bar{\beta} + (1+\bar{\beta})\epsilon)e^{(1+\bar{\beta})\epsilon})(1+\bar{\beta}e^{\bar{\beta}})\,e, \quad \bar{\beta} = \|\beta\|_\infty.$$

This theorem establishes more than exponential stability in the original state u, with an explicit (and, of course, conservative) estimate of the overshoot coefficient M. The theorem guarantees *robustness to learning*: it is a stability result

- under *all* the controllers
- approximated by all the possible NNs
- with *all* accuracy parameters ϵ in a certain range.

PARABOLIC PDES: GAIN OPERATORS WITH FUNCTIONS OF TWO VARIABLES

You might recall me emphasizing that I was starting, for pedagogical reasons, with operators whose inputs and outputs are functions of only one variable. Such operators arose in hyperbolic PDEs. Let us now advance to more complex PDEs in which the gain operator involves functions of more than one variable.

The simplest such case is the unstable *reaction–diffusion* PDE,

$$u_t(x,t) = u_{xx}(x,t) + \lambda(x)u(x,t).$$

This PDE is parabolic, with two derivatives in space. The PDE's reaction function $\lambda(x)$ causes instability (when positive).

For this system, the backstepping transform cannot be a simple convolution. It must be a general *Volterra operator*, with a kernel k that depends on two distinct variables, x and y, namely,

$$\boxed{w(x,t) = u(x,t) - \int_0^x k(x,y)u(y,t)\,dy}. \tag{9}$$

The kernel k must satisfy the following PDE:

$$k_{xx}(x,y) - k_{yy}(x,y) = \lambda(y)\ \ k(x,y) \tag{10}$$

where λ, the destabilizing reaction functional coefficient, is the PDE's input, and the backstepping kernel k the PDE's output. (There are, in addition, two boundary conditions, which I omit, and the domain is triangular, rather than rectangular.)

The mapping from λ to k, defined by the PDE (10), is hard to visualize. In Figure 8 we show one input–output pair (λ, k), where the output function $k(x, y)$ of two variables is the result of the input function $\lambda(x)$ of one variable "running through" the PDE (10). With some abuse (reuse) of notation, we denote the operator from λ to k as $\mathcal{K}$, namely, $\mathcal{K}: \lambda \mapsto k$.

With the backstepping transformation (9) and the feedback law $U(t) = \int_0^1 k(1,y)u(y,t)\,dy$, we obtain the exact target system $w_t = w_{xx}$, which is the heat equation (with suitable boundary conditions) and, therefore, exponentially stable.

When we approximate $\mathcal{K}$ by a DeepONet $\hat{\mathcal{K}}$ and use $\hat{\mathcal{K}}$ in the backstepping transformation and the feedback law, we obtain the perturbed target system

$$\hat{w}_t(x,t) = \hat{w}_{xx}(x,t) + \overbrace{\delta_{k0}(x)\,u(x,t) + \int_0^x \delta_{k1}(x,y)\,u(y,t)\,dy}^{\text{perturbation}}$$

with

$$\delta_{k0}(x) = -2\frac{d}{dx}(\mathcal{K}(\lambda)(x,x) - \hat{\mathcal{K}}(\lambda)(x,x))$$
$$\delta_{k1}(x,y) = -(\partial_{xx} - \partial_{yy} - \lambda(y))(\mathcal{K}(\lambda)(x,y) - \hat{\mathcal{K}}(\lambda)(x,y)).$$

The perturbation terms δ_{k0}, δ_{k1} include the kernel operator approximation error $\hat{\mathcal{K}}$, as well as the error's *derivatives*. I have to skip, in this time-limited lecture, the gory details of the full definition of the operator $\mathcal{K}$ and the proof of its Lipschitzness. However, let me assure you that the perturbations can be guaranteed to be small by training the neural operator to make δ_{k0}, δ_{k1} small.

Let me now show you one representative kernel approximation, in Figure 9. The approximated kernel $\hat{k}(x,y)$ looks coarse. This is expected since $\hat{k}$ is an output of an NN. You can see at the bottom of Figure 9 that the approximation error is about one tenth the size of the exact kernel k. So, the neural operator does a good enough job at what it has been trained to do.

What does it take to produce such a close approximation? It takes an NN with 76 million parameters. This is as expected for approximating PDEs that are nonlinear, in two dimensions, and with arbitrary input functions.

The training takes a mere 5 min with Nvidia RTX 3090Ti GPU (for 500 epochs, on 1,000 λ functions as Chebyshev polynomials). To generate a single kernel function $\hat{k}$ for a function λ outside the training set, it takes 25 *micro*seconds, making the entire kernel function computable at frequencies of tens of kilohertz. On an older laptop, and without a GPU, all these computations take about fifty times as long, namely, the training takes only 4–5 h, and the evaluation of $\hat{k}$ takes 0.5 ms, namely, the kernel is still computable in a real-time fashion at a kilohertz sampling frequency.

However, let us remember why we are here, not only to compute gain kernels accurately but for a "preservation of stabilization under approximation." The following theorem, for parabolic PDEs, reads almost exactly as Theorem 2 for the hyperbolic case.

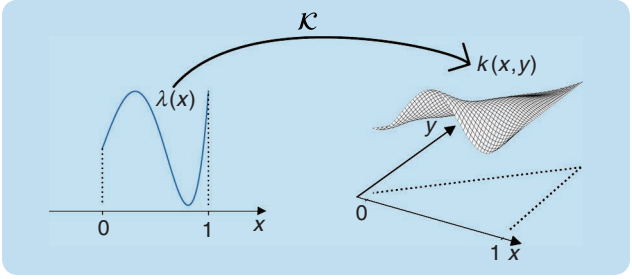

FIGURE 8 A visualization of the operator $\mathcal{K} : \lambda \mapsto k$ defined by the PDE (10).

Theorem 3 (Parabolic: *Robustness* to DeepONet)

For any system with $\bar{\lambda} := \|\lambda\|_\infty \le B_\lambda$ and $\|\lambda'\|_\infty \le B_{\lambda'}$, there exists $\epsilon^* > 0$ such that all controllers with gains $\hat{k} = \hat{\mathcal{K}}(\lambda)$ trained for any $\epsilon \in (0, \epsilon^*)$ guarantee

$$\boxed{\|u(t)\| \le Me^{-t/2}\|u_0\|} \qquad \forall t \ge 0$$

where

$$M(\epsilon, \bar{\lambda}) = (1 + \bar{\lambda}e^{2\bar{\lambda}})(1 + \bar{\lambda}e^{2\bar{\lambda}} + \epsilon)e^{\bar{\lambda}e^{2\bar{\lambda}}+\epsilon}.$$

The proof of Theorem 3 has major differences from the proof of Theorem 2, in terms of establishing both the Lipschitzness of the operator and the stability of the parabolic perturbed target system.

Before progressing with further reading of the theoretical material, the reader may refer to "Applied PDE Control and Experiments: Parabolic (Additive Manufacturing) and Hyperbolic (Traffic)."

CAN DEEPONET APPROXIMATE NOT JUST A GAIN BUT A FULL FEEDBACK LAW MAP?

You have seen me so far exploit the same idea twice: learn the mapping from a PDE model into a backstepping gain, specifically, first, the mapping $\beta \mapsto \mathcal{K}(\beta)$ for a hyperbolic PDE and then the mapping $\lambda \mapsto \mathcal{K}(\lambda)$, for an entirely different $\mathcal{K}$, for a parabolic PDE. However, in control implementation, we do not apply a gain function $k(\cdot)$ but a scalar control value U. This control value, $U(t) = \int_0^1 k(1-y)u(y,t)dy$, also written compactly, by suppressing the time variable, as $U = k * u(1) = (\mathcal{K}(\beta) * u)(1)$, is a scalar (real) output of the mapping $\mathcal{U} : (\beta, u) \mapsto U$ defined as

$$\mathcal{U}(\beta, u) = (\mathcal{K}(\beta) * u)(1).$$

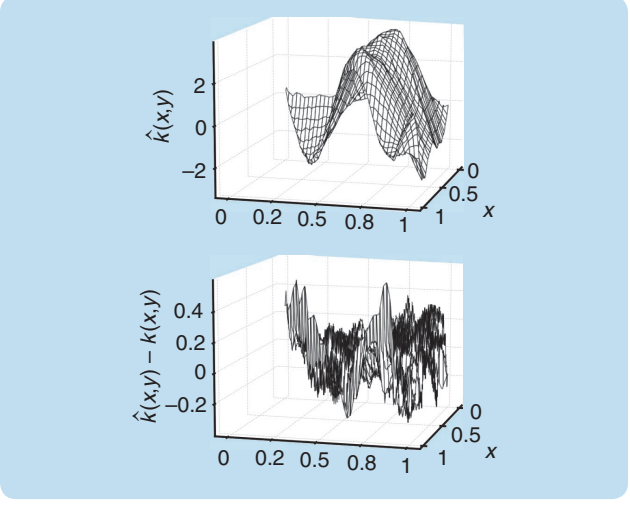

FIGURE 9 The kernel k is approximated by $\hat{k} = \hat{\mathcal{K}}(\lambda)$, using a DeepONet, to about 10% accuracy.

The operator $\mathcal{U}$ is a *functional*: it has two inputs, which are continuous functions, β and u, and a real output. The control implementation employs two such input functions into the feedback operator, β representing the *model* of the PDE and u representing the *state* of the PDE. Why not approximate this operator, $\mathcal{U}$, with a DeepONet, given $\mathcal{U}$'s importance, both practical and fundamental?

Since $\mathcal{U}$ is a continuous operator, we can approximate it. The DeepONet universal approximation theory guarantees a neural approximability of $\mathcal{U}$ with arbitrarily close accuracy.

Lemma 3 (DeepONet Approximability of $\mathcal{U}$)

There exists a neural operator $\hat{\mathcal{U}}$ such that

$$\boxed{|\mathcal{U}(\beta, u) - \hat{\mathcal{U}}(\beta, u)| < \epsilon}$$

for all $\epsilon > 0, \|\beta\|_\infty \leq B_\beta, \|u\|_\infty \leq B_u$.

It does not hurt to note, before proceeding, that the feedback law's approximation $\hat{\mathcal{U}}$, while arbitrarily accurate, is not equilibrium-preserving. The exact feedback operator $\mathcal{U}$ is linear in the state u, but $\hat{\mathcal{U}}$ is not, since it is an NN.

Is this feedback stabilizing? Yes, but not globally exponentially. The result is weaker because the approximation error acts as a nonvanishing perturbation on the system. Stability is practical and semiglobal.

Theorem 4 (Semiglobal Practical Stability)

For any system with $\|\beta\|_\infty \leq B$, all controllers $\hat{\mathcal{U}}(\beta, u)$ trained for any $\epsilon < (\sqrt{2}\, B_u)/(e(1+B_\beta))$ guarantee

$$\|u(t)\| \leq (1+B_\beta)e(1+B_\beta e^{B_\beta})e^{-t}\|u_0\| + \underbrace{(1+B_\beta)\frac{e}{\sqrt{2}}\ \epsilon}_{\text{residual value}} \tag{11}$$

for all initial conditions

$$\|u_0\| \leq \frac{1}{1+B_\beta e^{B_\beta}}\underbrace{\left(\frac{B_u}{e(1+B_\beta)} - \frac{\epsilon}{\sqrt{2}}\right)}_{>0}. \tag{12}$$

In plain English, the neural approximation of the operator from the PDE model and PDE state to the control input is stabilizing semiglobally and practically.

The *practical* nature of stability is shown in the estimate (11): after an exponential decay, the state settles to a residual set whose spatial L_2 size is proportional to the operator approximation error, ϵ, emphasized in yellow. The result is *semiglobal* in the sense that the region of attraction (12) can be arbitrarily enlarged by training the feedback operator for larger state values, B_u, emphasized in pink. To approach global exponential stability, the price is a larger NN.

The simulations in Figure 10 illustrate Theorem 4. The PDE is stabilized but with a small residual error, noticeable in the plot for control, toward the end of the time interval, on the lower left.

REAL-TIME USES OF DEEPONET

Neural operators (such as DeepONet) generate solutions of PDEs on the order of a thousand times faster than numerical PDE solvers. However, my first two examples, one hyperbolic and one parabolic, have performed the evaluation with the

Applied PDE Control and Experiments: Parabolic (Additive Manufacturing) and Hyperbolic (Traffic)

After this litany of hyperbolic and parabolic PDEs, theorems, and formulas, it may be a good moment to take a break and turn the attention to physics and practice. PDE backstepping has seen many uses in control applications, including oil drilling and lithium-ion batteries. However, let me make a brief mention of two recent books, by my Ph.D. students, on such applications of PDE control. Shumon Koga's book [S4] for parabolic models in additive manufacturing and Huan Yu's book [S5] for hyperbolic models in traffic control have both gone from theorems to experiments. Incidentally, both books are in Birkhäuser's series edited by Professor Tamer Basar.

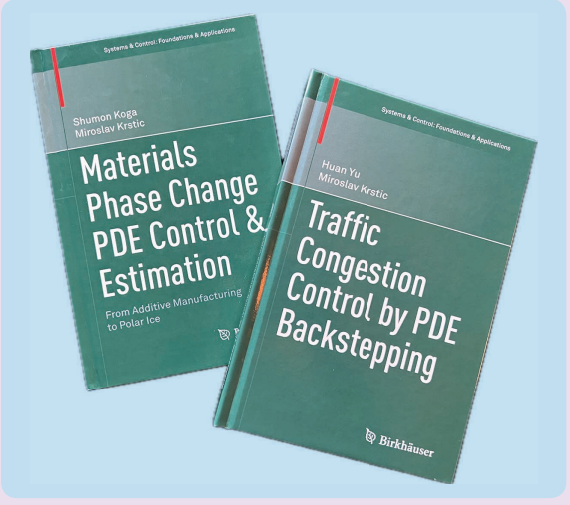

FIGURE S3 Books [S4] and [S5].

REFERENCES

[S4] S. Koga and M. Krstic, *Materials Phase Change PDE Control & Estimation*. Basel, Switzerland: Birkhäuser, 2020.

[S5] H. Yu and M. Krstic, *Traffic Congestion Control by PDE Backstepping*. Cham, Switzerland: Springer, 2022.

neural operator only offline, followed by using the controller in real time.

I will close the lecture with control problems where the operator is evaluated online repeatedly. The thousandfold computational speedup, which the NN offers over solving PDEs in real time, is crucial.

Gain Scheduling for Nonlinear PDEs

The first *online* use of a DeepONet that I show is in gain scheduling, for nonlinear PDEs. Extending linear PDE backstepping to nonlinear PDEs, using a nonlinear Volterra series, is the best approach, the PDE equivalent of the rigorous, elegant, feedback linearization (or backstepping) for ODEs but incredibly, almost hopelessly complex. Rafael Vazquez and I have done it once [7], [8] and would recommend the experience only to an enemy.

The alternative to a full-blown nonlinear design is gain scheduling. The article [9] by Rugh and Shamma, which won the IFAC High Impact Paper Award, surveys rigorous gain scheduling for ODEs. My students and I introduced such a linear parameter varying-esque framework for nonlinear PDEs in [10], a decade after [9]. This approach to nonlinear PDE control has not been revisited since, due to its theory being demanding, in spite of its straightforward implementation.

To illustrate gain scheduling for nonlinear PDEs, I return to the hyperbolic PDE with recirculation but where now the recirculation β depends on the outlet state $u(0, t)$, shown in red, in addition to depending on x:

$$u_t(x,t) = u_x(x,t) + \beta(x, u(0,t))\, u(0,t). \tag{13}$$

You can see in Figure 3 a linear version of such a plant having exponentially growing oscillations. Under a nonlinearity, the unstable oscillations settle into a limit cycle, as exhibited in Figure 11.

How does a *linear* controller, designed based on the Jacobian linearization at the origin, fare for this nonlinear system? It fares poorly, as one observes in Figure 12. Except for miniscule initial conditions, such a linear feedback makes matters even worse than they are in the open loop in Figure 3. The linear control eliminates the limit cycle and results in an exponential instability.

We now introduce a nonlinear operator for *scheduling* the backstepping gains, the *parameterized* involution operator $\mathcal{K}$:

$$\mathcal{K}(\beta)(x,\nu) = -\beta(x,\nu) + \int_0^x \beta(x-y,\nu)\mathcal{K}(\beta)(y,\nu)\,dy$$

where $\nu \in \mathbb{R}$. The input functions β of this operator depend on *two* variables, the spatial variable x, as before, in (7), and

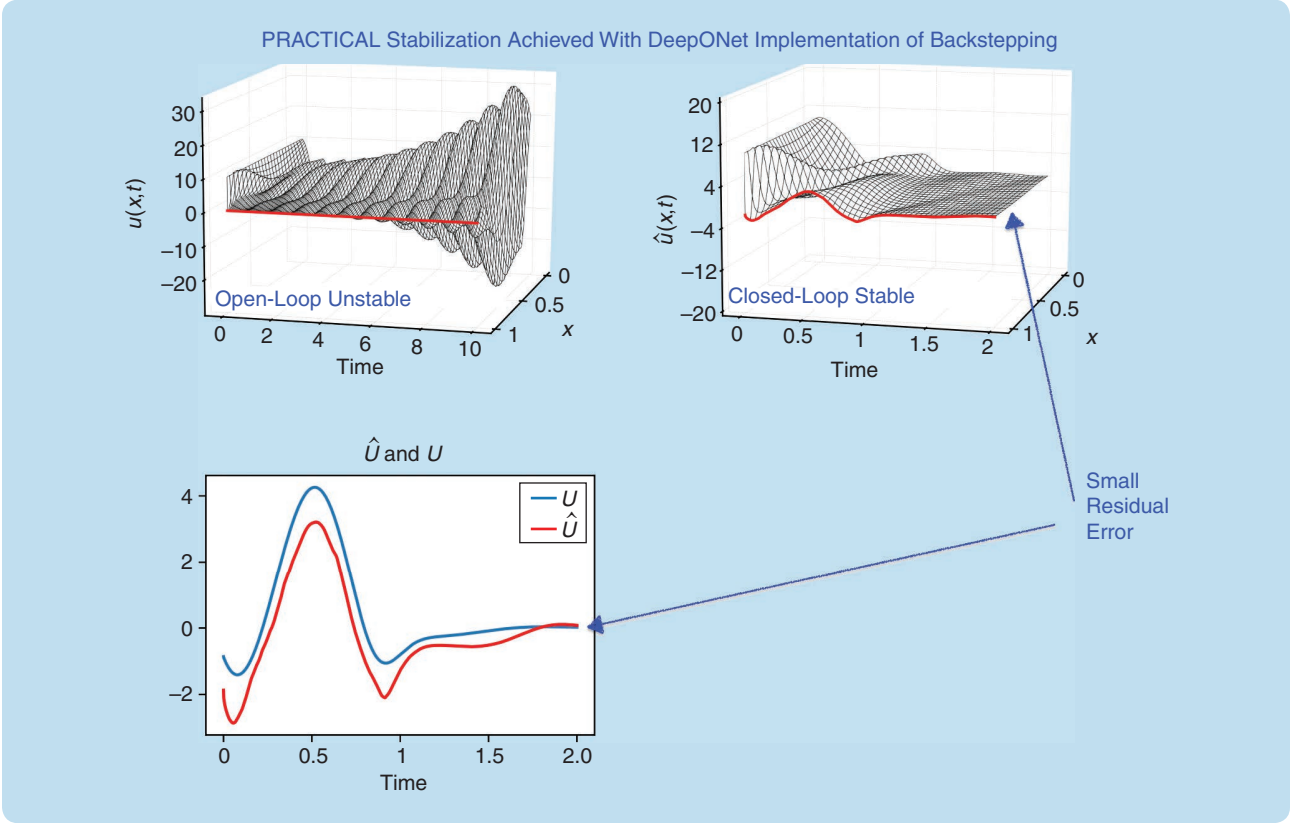

FIGURE 10 Practical stabilization, using the DeepONet approximation $U = \widehat{\mathcal{U}}(\beta,u)$ of the feedback law, with a small noticeable residual error toward the end of the plot for control on the lower left.

the variable ν, which stands in for the outlet state $u(0,t)$. We employ this new operator in a gain scheduling nonlinear feedback law, with approximation $\hat{\mathcal{K}}$,

$$U(t) = \int_0^1 \hat{\mathcal{K}}(\beta)(1-y, u(0,t))\, u(y,t)\, dy. \tag{14}$$

For simulation results, let us turn our attention to Figure 13. Figure 13(a) illustrates the stabilization with the gain scheduling controller, implementing the DeepONet $\hat{\mathcal{K}}(\beta)(1-y,u(0,t))$ as a gain. Figure 13(b) shows how the spatially dependent gain $\hat{k}(x,t) = \hat{\mathcal{K}}(\beta)(1-y,u(0,t))$ evolves over time, in response to the variation of the scheduling state $u(0,t)$.

Can we prove anything for a DeepONet-enabled gain scheduling controller, such as the one illustrated by Figure 13? Yes, we can prove stability, stated in the next theorem. However, the result is not easy, and its proof approaches 20 pages. This is because even in the absence of a perturbation induced by the DeepONet approximation, perturbations caused by derivatives both in time (as in gain scheduling for ODEs [9]) and in space need to be dominated in the analysis.

Theorem 5 (Local Stabilization in H_1 Norm)

For all $c > 0$, there exist $\epsilon^*, \Omega_0, M > 0$ such that with any neural operator $\hat{\mathcal{K}}$ approximated to any accuracy $\epsilon \in (0, \epsilon^*)$,

$$\Omega(u_0) \le \Omega_0 \;\Rightarrow\; \boxed{\Omega(u(t)) \le M\Omega(u_0)e^{-\frac{c}{2}t}}$$

for all $t \ge 0$, where

$$\Omega(u(t)) \;:= u^2(0,t) + \|u(t)\|^2 + \|u_x(t)\|^2.$$

In this theorem, our estimate of a region of attraction Ω_0, shown in yellow, in terms of the H_1 spatial norm of the state, shown in blue, depends on the accuracy ϵ of the neural operator $\hat{\mathcal{K}}$.

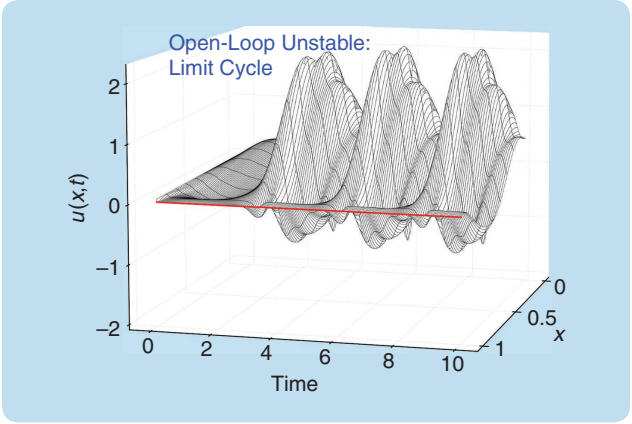

FIGURE 11 The transport PDE with a *nonlinear* recirculation (13) is unstable at the origin but settles into a (spatiotemporal) limit cycle.

Adaptive Control for Unknown Functional Coefficients

The last result I have, for today, is the most exciting, at least to me: in online adaptive control. You will see offline learning and online learning working *in tandem*.

I return to the example (2), repeated here for convenience:

$$u_t(x,t) = u_x(x,t) + \beta(x)\, u(0,t). \tag{15}$$

However, $\beta(x)$ denotes now an *unknown* functional coefficient. The online estimate of $\beta(x)$ is denoted by $\hat{\beta}(x,t)$, and its updating includes projection in order to guarantee that $\|\hat{\beta}(\cdot,t)\|_\infty \le B$ for all $t \ge 0$. The neural operator $\hat{\mathcal{K}} \approx \mathcal{K}$ has already been trained; it produces the adaptive gain $\hat{\mathcal{K}}(\hat{\beta})$ by being fed the estimate $\hat{\beta}$ and is employed in the *indirect adaptive control law*,

$$U(t) = \int_0^1 \hat{\mathcal{K}}(\hat{\beta})(1-y,t)\, u(y,t)\, dy.$$

The estimation of β with $\hat{\beta}$ is performed online. The update law is designed by Lyapunov approach and given by

$$\frac{\partial}{\partial t}\hat{\beta}(x,t) = \underbrace{\frac{\gamma}{1+\|w(t)\|_c^2}}_{\text{normalization}} \times \underbrace{\left[e^{cx}w(x,t) - \int_x^1 e^{cy}\hat{\mathcal{K}}(\hat{\beta})(y-x,t)\, w(y,t)\, dy\right]}_{\text{regressor}} \underbrace{u(0,t)}_{\substack{\text{regulation}\\ \text{error}}} \tag{16}$$

where

$$w(x,t) = u(x,t) - \int_0^x \hat{\mathcal{K}}(\hat{\beta})(x-y,t)\, u(y,t)\, dy$$

$$\|w(t)\|_c^2 = \int_0^1 e^{cx}w^2(x,t)\, dx.$$

The structure of the update law (16) is conventional: a product of a regressor with the regulation error, divided by normalization, and employing the backstepping transformation and its weighted norm.

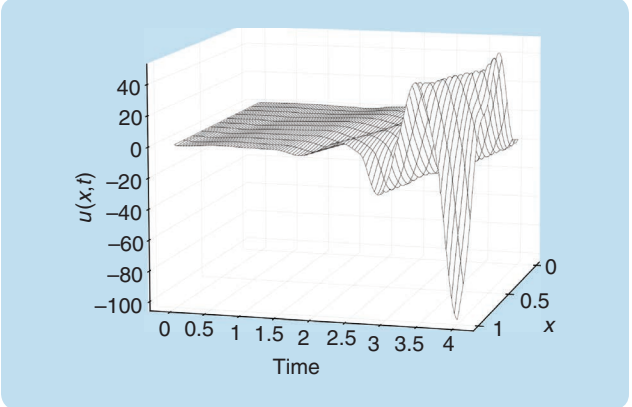

FIGURE 12 A controller based on linearization not only fails to stabilize (13) but destroys its boundedness (limit cycle) and induces an exponential growth. This is the result of acting aggressively in response to the locally destabilizing recirculation and disregarding the bounding effect of the nonlinearity in $\beta(x,u(0,t))$.

The update law (16) looks like a PDE, but it is not. It has a derivative in time but no derivative in x. Instead, it has integration in x. The right-hand side of (16) depends nonlinearly on $\hat{\beta}$ through the neural operator $\hat{\mathcal{K}}$. In today's parlance, this is a nonlinear *ensemble* system, in the same mathematical family with mean-field games.

Let me illustrate this online learning-based control, enabled by offline learning, using the plots in Figure 14. The plant is open-loop unstable. The adaptive controller stabilizes the plant, in spite of the β function being unknown. How, exactly?

Adaptive control is a rather complex process, so let me explain it step by step. The plant's instability drives the estimation of β. By about 10 s, the estimate $\hat{\beta}$ of β, shown in Figure 14(c), is good enough to produce a controller that is stabilizing, as shown in Figure 14(b), where the state $u(x, t)$ rapidly decays after about 10 s. The stabilization of $u(x, t)$ then eliminates the persistency of excitation (PE) in the update law (16). This loss of PE automatically freezes the updating of $\hat{\beta}$, from about 10 s onward.

Figure 14(c) shows that $\hat{\beta}$ has settled to the red profile, but it has not converged to the *true* blue profile. This lack of exact convergence does not matter. The red profile for $\hat{\beta}$ is stabilizing, which is the sole task of adaptive *control* (system *identification* is not the task).

So, if you are uninitiated in the miracle of adaptive control, the simulation in Figure 14 introduces it to you: succeeding at the task of control, without paying the price (of inducing PE) for unnecessary complete learning of the model. In fact, unless tracking of a complex reference is

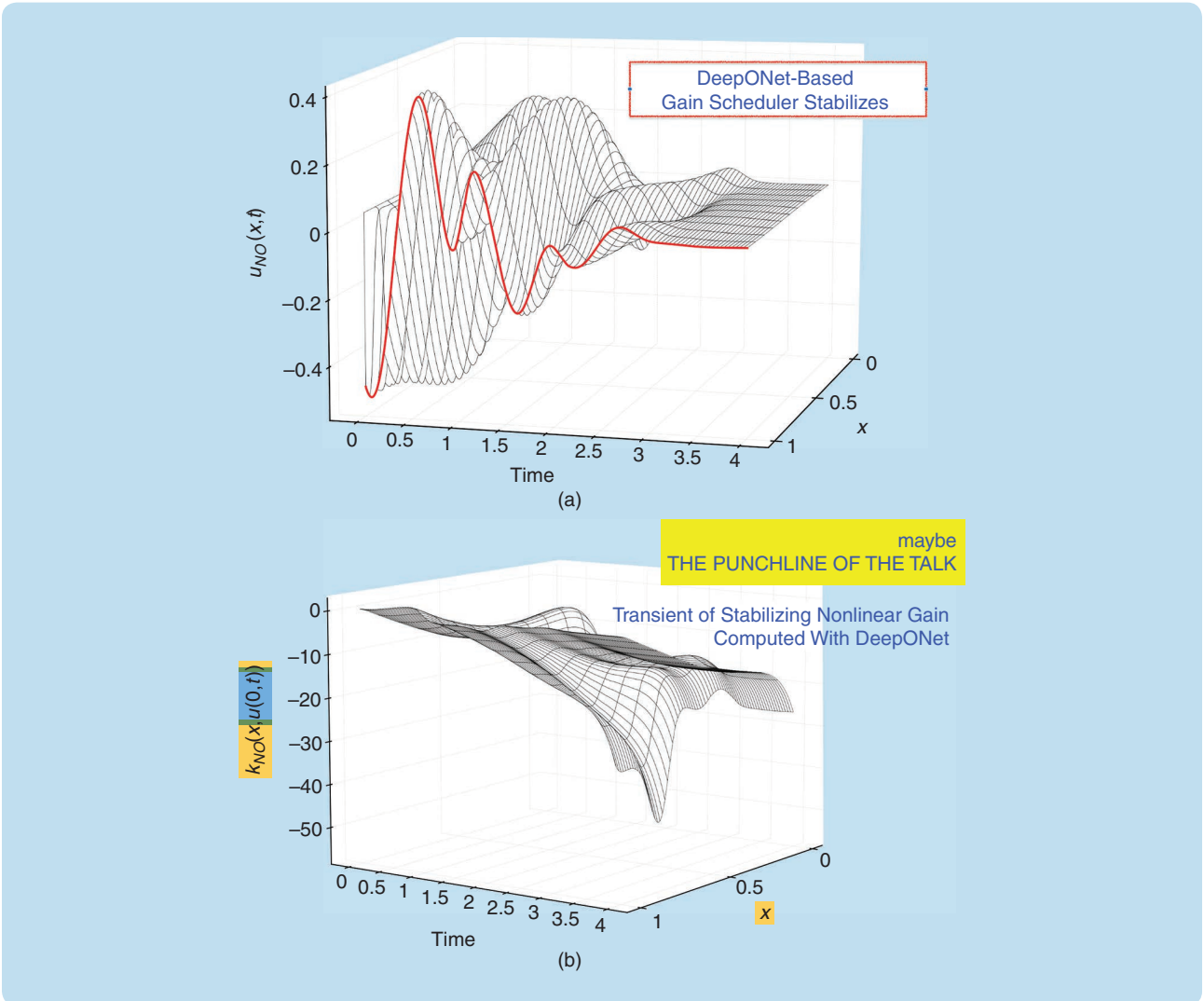

FIGURE 13 (a) The nonlinear gain-scheduling controller (14) stabilizes the nonlinear PDE (13), with the control input shown in red. (b) The spatially dependent gain $\hat{k}(x,t) = \hat{\mathcal{K}}(\beta)(1 - y, u(0,t))$ evolves over time, in response to the variation of the scheduling state $u(0, t)$. It is interesting how the gain profile grows more negative, overall, over time. This is consistent with the fact that the open-loop system has a limit cycle, shown in Figure 11. The controller is more cautious initially, given the bounding open-loop benefit of the limit cycle, and grows more aggressive as the state gets smaller. The DeepONet-enabled gain scheduling controller acts in a meaningful, theory-interpretable fashion.

desired, "incidental PE" and perfect model identification are synonymous with bad transient performance. We see that avoided in Figure 14.

Is the adaptive control experiment in Figure 14 backed by theory? Yes.

Theorem 6 (Global Stabilization, Pointwise Regulation)
There exist $R, \rho > 0$ such that

$$\Gamma(t) \leq R(e^{\rho\Gamma(0)} - 1), \forall t \geq 0$$
$$\Gamma(t) = \int_0^1 [u^2(x,t) + (\beta(x) - \hat{\beta}(x,t))^2]dx$$

and

$$\lim_{t\to\infty} u(x,t) = 0, \quad \forall x \in [0,1].$$

Global stability in the L_2 norm of the state and parameter error, with an exponential-in-initial-condition estimate, along with a pointwise-in-space regulation of the state u, is proven.

SUMMARY AND PERSPECTIVES

Let me close with a question and an answer to it. Is this learning-based approach to control model-free? It is absolutely not; the role of ML is not to learn the unknown. ML's role is to encode, once and for all, in an NN, the model-based backstepping design for a class of PDEs. If it is model-free learning and control you are after, you might want to check out extremum seeking instead.

To recap, a real-time speedup of three orders of magnitude in producing feedback gains, achieved with a very reasonable training time (minutes), enables control of PDEs that are nonlinear or have unknown functional parameters.

What problems open up for future research? This entire list does:

- 2D and 3D PDEs
- coupled + ensemble PDEs
- other nonlinear PDE classes
- PDE observers
- nonlinear ODE delay systems–predictor feedback
- *applications*.

I am particularly excited about applications but also about predictor feedback for nonlinear delay systems, which requires an approximation of the open-loop *flow map* of the nonlinear ODE.

However, why should *you* care, if you do not work on PDE control? Many designs for nonlinear controllers and

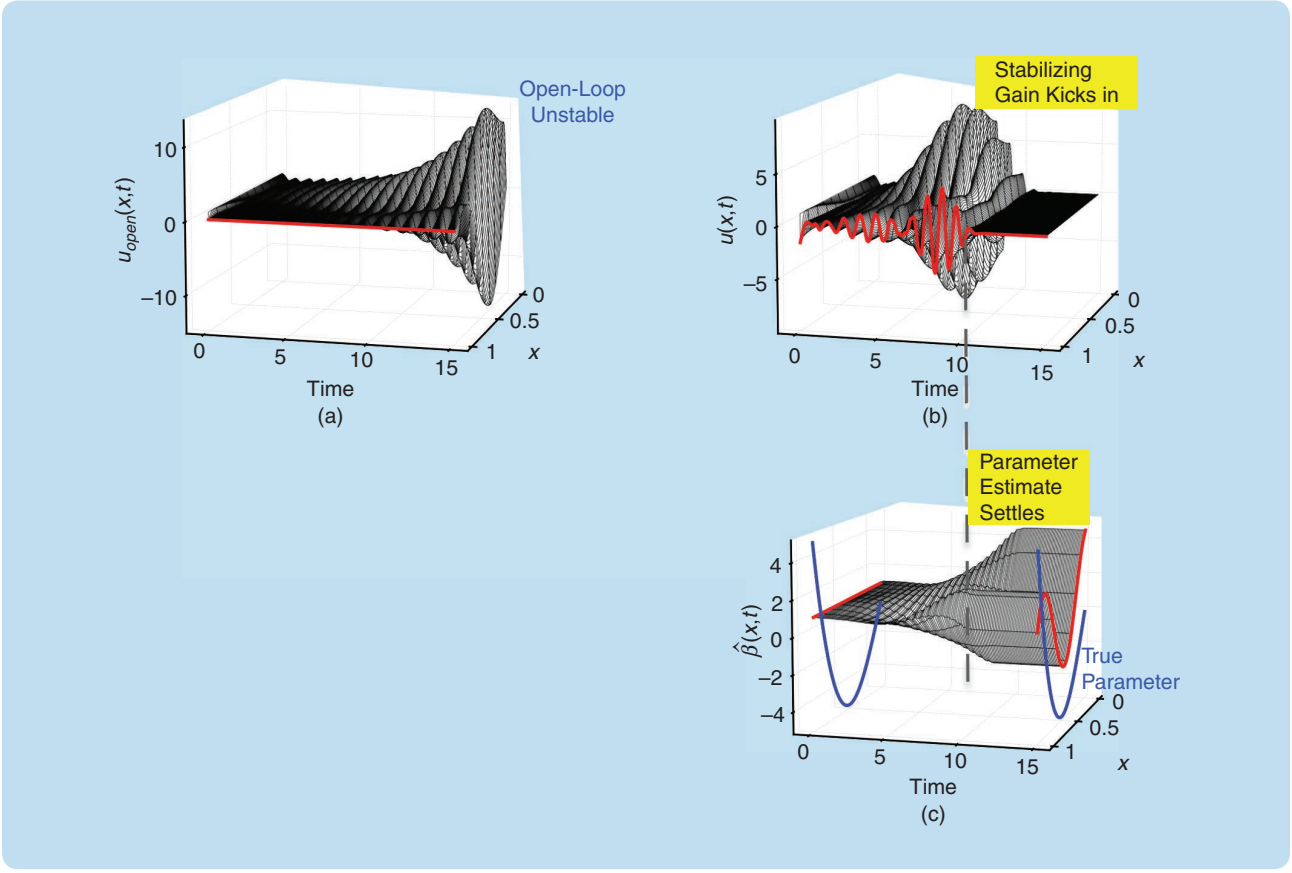

FIGURE 14 (a) The plant is open-loop unstable. (b) The adaptive controller stabilizes the plant, once the estimate $\hat{\beta}$ has become good enough, around 10 s, to provide a stabilizing feedback gain. (c) The parameter updating ends around 10 s, with a profile $\hat{\beta}(x)$ in red, which differs from the true profile $\beta(x)$ in blue, since the stabilization of the state $u(x, t)$ terminates the persistence of excitation and does not require perfect identification of $\beta(x)$.

PDE Backstepping: 77 Notable Contributors

I would be absolutely remiss not to acknowledge the PDE backstepping community in this lecture. This is a community of exceptional talent. It has revolutionized a branch of control theory, PDE control, which is physically ubiquitous and highly necessitated in technology.

Seventy-seven notable contributors to PDE backstepping are: Aamo, Ahmed-Ali, Alalabi, Alleaume, Andrade, Anfinsen, Ascencio, Astolfi, Baccoli, Balogh, Bastin, Bekiaris-Liberis, Bernard, Bhan, Bresch-Pietri, Bribiesca-Argomedo, Burkhardt, Cai, Cerpa, Chen (several), Cochran, Coron, Demir, Deutscher, Diagne, Di Meglio, Espitia, Frihauf, Gehring, Giri, Guan, Guo, Hasan, Hashimoto, Hayat, Hu, Karafyllis, Koga, Lamnabhi-Lagarrigue, Liu (several), Magnis, Meurer, Morris, Moura, Nguyen, Olive, Oliveira, Orlov, Parisini, Pisano, Polyakov, Prieur, Qi, Rathnayake, Redaud, Ren, Sanz-Diaz, Schuster, Shi, Siranosian, Smyshlyaev, Steeves, Su, Susto, Tang, Tsubakino, Vazquez, Wang (several), Xu, Yu, Zhang, and Zhu.

estimators, for systems such as $\dot{x} = f(x) + g(x)u$, are predicated on solving a PDE:

- dynamic programming: DeepONet for $(f,g) \mapsto V$
- output regulation
- Kravaris-Kazantzis-Luenberger (KKL) observers
- Immersion & Invariance (I&I) adaptive control.

However, I do not promise that DeepONet will remove the curse of dimensionality. Approximating the Hamilton–Jacobi–Bellman map for an ODE with more than a few state variables will always be very hard, possibly harder than designing controllers for linear PDEs in dimensions not much higher than three.

THANKS

As I thank you for your attention, and my sponsors, the National Science Foundation, Air Force Office for Scientific Research, and Office of Naval Research, for their kind support, let me also thank my collaborators:

1) Luke Bhan (UC San Diego)
2) Yuanyuan Shi (UC San Diego)
3) Maxence Lamarque (Mines-Paris)
4) Mamadou Diagne (UC San Diego)
5) Rafael Vazquez (University of Seville)
6) Jie Qi (Donghua University, China)
7) Shanshan Wang (University of Shanghai for Science and Technology, China)
8) Jing Zhang (Donghua University, China).

Starting on this topic in January this year, they have produced many more results than I am able to show on this day in December, toppled some amazing technical barriers, and shown that ML is *no bane* but a *boon* for control.

ACKNOWLEDGMENT

This work was supported by NSF Grants CMMI-2228791, ECCS-2151525, and ECCS-2210315; ONR Grant N00014-23-1-2376; and AFOSR Grant FA9550-23-1-0535.

AUTHOR INFORMATION

Miroslav Krstic (mkrstic@ucsd.edu) received the undergraduate degree in electrical engineering from the University of Belgrade and the M.S. and Ph.D. degrees from the University of California, Santa Barbara, in 1992 and 1994, respectively. Miroslav Krstic is a distinguished professor and senior associate vice chancellor for research at the University of California, San Diego, La Jolla, CA 92093 USA. He has received the IEEE Hendrik Bode Lecture Prize; the Richard Bellman Control Heritage Award; the SIAM Reid Prize; the ASME Rufus Oldenburger Medal; IFAC TC Awards for Nonlinear Control, Distributed Parameter Systems, and Adaptive Systems; IFAC's Harold Chestnut Textbook Prize; the Ragazzini Education Award; the inaugural AV Balakrishnan Award for the Mathematics of Systems; and other recognitions. He has been elected Fellow of seven scientific societies and a member of the Serbian Academy of Sciences. Krstic serves as editor in chief of *Systems & Control Letters* and senior editor of *Automatica* and is editor in chief-designate for *IEEE Transactions on Automatic Control* (2026).

REFERENCES

[1] P. V. Kokotovic, "The joy of feedback: Nonlinear and adaptive," *IEEE Control Syst. Mag.*, vol. 12, no. 3, pp. 7–17, Jun. 1992, doi: 10.1109/37.165507.

[2] L. Bhan, Y. Shi, and M. Krstic, "Neural operators for bypassing gain and control computations in PDE backstepping," *IEEE Trans. Autom. Control*, early access, Dec. 26, 2023, doi: 10.1109/TAC.2023.3347499.

[3] D. F. Delchamps, "Analytic stabilization and the algebraic Riccati equation," in *Proc. 22nd IEEE Conf. Decis. Control*, Piscataway, NJ, USA: IEEE, 1983, pp. 1396–1401, doi: 10.1109/CDC.1983.269767.

[4] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control, Signals Syst.*, vol. 2, no. 4, pp. 303–314, 1989, doi: 10.1007/BF02551274.

[5] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Trans. Neural Netw.*, vol. 6, no. 4, pp. 911–917, Jul. 1995, doi: 10.1109/72.392253.

[6] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nature Mach. Intell.*, vol. 3, no. 3, pp. 218–229, 2021, doi: 10.1038/s42256-021-00302-5.

[7] R. Vazquez and M. Krstic, "Control of 1-D parabolic PDEs with Volterra nonlinearities, part I: Design," *Automatica*, vol. 44, no. 11, pp. 2778–2790, 2008, doi: 10.1016/j.automatica.2008.04.013.

[8] R. Vazquez and M. Krstic, "Control of 1D parabolic PDEs with Volterra nonlinearities, part II: Analysis," *Automatica*, vol. 44, no. 11, pp. 2791–2803, 2008, doi: 10.1016/j.automatica.2008.04.007.

[9] W. J. Rugh and J. S. Shamma, "Research on gain scheduling," *Automatica*, vol. 36, no. 10, pp. 1401–1425, 2000, doi: 10.1016/S0005-1098(00)00058-3.

[10] A. A. Siranosian, M. Krstic, A. Smyshlyaev, and M. Bement, "Gain scheduling-inspired boundary control for nonlinear partial differential equations," *ASME J. Dyn. Syst., Meas., Control*, vol. 133, no. 5, 2011, Art. no. 051007, doi: 10.1115/1.4004065.