CAPE: Channel-Attention-Based PDE Parameter Embeddings for SciML

Makoto Takamoto[*]
NEC Labs Europe

Francesco Alesiani
NEC Labs Europe

Mathias Niepert
University of Stuttgart

Abstract

Scientific Machine Learning (SciML) is concerned with the development of machine learning methods for emulating physical systems governed by partial differential equations (PDE). ML-based surrogate models substitute inefficient and often non-differentiable numerical simulation algorithms and find multiple applications such as weather forecasting and molecular dynamics. While a number of ML-based methods for approximating the solutions of PDEs have been proposed in recent years, they typically do not consider the parameters of the PDEs, making it difficult for the ML surrogate models to generalize to PDE parameters not seen during training. We propose a new channel-attention-based parameter embedding (CAPE) component for scientific machine learning models. The CAPE module can be combined with any neural PDE solver allowing it to adapt to unseen PDE parameters without harming the original model's performance. We compare CAPE using a PDE benchmark and obtain significant improvements over the base models.

1 Introduction

Many real-world phenomena, ranging from macroscopic weather forecasts to microscopic molecular dynamics, can be modeled with partial differential equations (PDEs). To obtain those PDEs' solution, numerical simulation methods have been developed for many years and have achieved a high level of accuracy in solving these equations. However, numerical methods are resource intensive and time-consuming even when run on larger supercomputers to obtain sufficiently accurate results.

Recently, there has been a rapidly growing interest in machine learning methods for the problem of solving PDEs due to their various applications in science and engineering Guo et al. (2016); Lusch et al. (2018); Sirignano & Spiliopoulos (2018); Raissi (2018); Kim et al. (2019); Hsieh et al. (2019); Bar-Sinai et al. (2019); Bhatnagar et al. (2019); Pfaff et al. (2020); Wang et al. (2020); Khoo et al. (2021). A considerable number of papers have shown the advantage of ML-based surrogate models. The majority of these methods, however, are purely data-driven, which does not allow us to change PDE parameters. Although a few models are taking into account PDE parameters, they are tailored to specific neural networks and cannot be used with other state-of-the-art methods. This makes it difficult for the SciML community to develop models with high generalization capacity not only for the initial conditions but both for different types of PDEs *and* PDE parameters.

To overcome the shortcomings of existing data-driven SciML models, we propose a new and effective parameter embedding module by utilizing the channel-attention method. The crucial idea is that a neural network generates intermediate (approximated) field data for future time steps which are then interpolated by a BASE model such as the FNO (Li et al., 2021a) to predict the field data for the next time step. CAPE can be combined with any existing autoregressive neural PDE solvers. Figure 1 illustrates the proposed CAPE framework. We performed extensive experiments using various PDEs

[*]E-mail:Makoto.Takamoto@neclab.eu

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

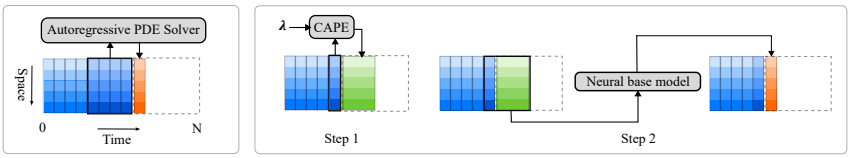

Figure 1: The standard autoregressive approach (left) and the proposed CAPE approach (right) which consists of two interdependent steps.

with a large number of different parameters evaluating the effectiveness and efficiency of the proposed method in comparison with popular state-of-the-art methods.

2 CAPE: A Framework for Neural PDE Solvers

Problem Definition We consider PDEs: $\partial_t u = F(t, x, u, \partial_x u, ...)$ whose solution is described as a temporal sequence of field data $\left\{u^k\right\}_{k=0,...,N} := u^0, u^1, ..., u^N$ where u^k is the field data at time step t_k, that is, the state of the physical system governed by the PDE under consideration at time t_k discretized using $\Delta t = T/N$. Each $u \in \mathcal{X} \subseteq^{c \times x_1, ..., x_D}$ represents the field tensor data with c the number of physical variables such as density and velocity, and x_i the spatial dimensions of the i-th coordinate. We will often refer to c as the channel dimension. We aim to emulate numerical simulators of PDEs which iteratively map $\mathcal{M} : \mathcal{X} \to \mathcal{X}$ from u^k to u^{k+1}. The emulator (or surrogate model) is a learnable function modeled as a neural network NN with weights θ. We refer to the parameters of a neural network as weights to avoid a conflict in terminology with the parameters of PDEs. In the following, we denote the emulator's prediction at time index k as $\tilde{u}^k$. Auto-regressive neural networks predict the next time step's field data based on a sequence of field data tensors of length ℓ $\tilde{u}^{k+1} = \mathrm{NN}(\tilde{u}^{k-\ell+1}, ..., \tilde{u}^k; \theta)$. Given the length of the input sequence $N \in \mathbb{N}$, and an initial input sequence $\left(u^0, ..., u^{\ell-1}\right) = \left(\tilde{u}^0, ..., \tilde{u}^{\ell-1}\right)$ of length $\ell < N$, the ML model auto-regressively generates the remaining sequence $\left(\tilde{u}^\ell, ..., \tilde{u}^N\right)$. The training loss is typically the normalised mean-squared error (nMSE) (Takamoto et al., 2022) between the predicted and the true field data tensors Since we are training an auto-regressive neural network, the gradients of the above loss can be backpropagated in time in various ways. We discuss this in the following sections. Figure 1(left) illustrates this auto-regressive approach to solving PDEs. In the vast majority of experimental setups, the assumption is made that $\ell > 1$, and, therefore, an initial input sequence of length ℓ is available to the model; in practice, this would require a numerical simulation to be run for $\ell - 1$-time steps from the initial condition and for each PDE parameter λ. The main idea of CAPE is to learn to generate these sequences based on the current field data and parameter values λ and use those as input to an off-the-shelf neural surrogate model such as an FNO (Li et al., 2021a) or U-Net (Ronneberger et al., 2015) to perform a complex interpolation.

Combining Neural PDE Solvers with the CAPE Module The proposed approach is motivated by the need for neural PDE solvers to generalize to PDE parameters unseen during training. We propose CAPE, a novel neural network architecture that takes the prior state of the system $\tilde{u}^k$ and PDE parameters λ as input and predicts the ℓ-*intermediate* future states $\left\{\hat{u}_{\text{cape}}^{k \to k+i}\right\}_{i=1,...,\ell} = \text{CAPE}(u^k, \lambda; \theta_{\text{CAPE}})$The output of CAPE is then used by the BASE network. The overall structure is provided in Figure 1(right). The intuition behind this approach is that the intermediate future states capture information about the PDE parameters' impact by attending to the results of the convolutional operations. While we do not change the architecture of the base neural PDE solvers, we propose to use them to predict, given the past temporal states and the intermediate future states, the state for the next time step. This is contrary to the typical use of neural PDE solvers. The base network is trained jointly with the CAPE module. As shown in section 3, this choice improves the prediction capability of the BASE network.

During training, the output of CAPE is regularized by the additional loss term

$$\mathbf{L}_{\text{cape}}(\theta_{\text{CAPE}}) = \sum_{k=\ell}^{N} \sum_{i=1}^{\min(\ell, N-k)} \text{nMSE}\left(\hat{u}_{\text{cape}}^{k \to k+i}, u^{k+i}\right), \tag{1}$$

which forces the CAPE module to predict a temporal sequence of future field data $\left\{u^{k+i}\right\}_{i=1,...,\ell}$.

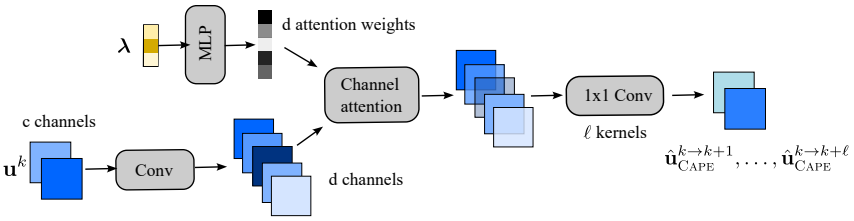

Figure 2: The CAPE module for one type of convolution (residual connections are omitted).

Finally, the intermediate sequence $\left\{\hat{u}^{k\to k+i}_{\text{cape}}\right\}_{i=1,\ldots,\ell}$ is concatenated with u^k, the field data at time t_k, and given to the base network to make the final prediction. In summary, the CAPE module transforms the input variables $\left\{u^k, \lambda\right\}$ into temporal-sequential intermediate field data $\{u^k, \hat{u}^{k\to k+1}_{\text{cape}}, \ldots, \hat{u}^{k\to k+\ell}_{\text{cape}}\}$ which is then interpolated by the base neural network. Before we introduce the inductive bias of the CAPE module, we motivate the general approach from a classical numerical simulation perspective.

Channel-Attention-Based Parameter Embedding (CAPE Module) CAPE computes 3 different d-dimensional channel attention masks $a_\alpha \in^d, \alpha = 1, 2, 3$ from the parameters of the PDE λ using a 2-layer MLP: $a_\alpha = W_{2,\alpha}\sigma(W_{1,\alpha}\lambda)$, where d is the channel dimension in the feature space and σ is the GeLU activation function (Hendrycks & Gimpel, 2016). $W_\alpha = (W_{1,_\alpha}, W_{2,_\alpha})$ are the weights associated with three operators: a 1×1-convolution (g_1), a depth-wise convolution (g_2), and a spectral convolution (Li et al., 2021a) (g_3), that are used to compute the tensor representations $z^k_\alpha \in^{d\times n_x\cdots}$ as $z^k_\alpha = g_\alpha(u^k, W_\alpha)$. The tensors are then multiplied by the attention

$$v^k_\alpha = a^k_\alpha \odot_1 z^k_\alpha \tag{2}$$

using the Hadamard operator ($\odot_1$) over the first dimension (the channel dimension) which is equivalent to the broadcast operation of ML programming languages. A similar mechanism has been proposed for visual tasks, called the squeeze-and-excitation networks (Hu et al., 2018) which enhances useful channels of the feature vector of convolutional networks through an attention mechanism. The feature $v^k_\alpha \in^{d\times n_x\cdots}, \alpha = 1, , 2, 3$ are combined to form an intermediate feature $y^k \in^{c\times\ell\times n_x\cdots}$ as

$$y^k = h_{1\times1,d\to c\times\ell}\left(\sigma\left(h_{1\times1,c\to d}(u^k) + \sum_\alpha v^k_\alpha\right)\right) \tag{3}$$

where $h_{1\times1,*}$ are 1×1 convolutions that adjust the number of dimensions, in particular $h_{1\times1,c\to d}$: $c\times n_x\cdots\to d\times n_x\ldots$, while $h_{1\times1,d\to c\times\ell} : d\times n_x\cdots\to c\times\ell\times n_x\ldots$. Finally, the sequence of predictions is computed

$$\left\{u^{k\to k+i}_{\text{cape}}\right\}_{i=1,\ldots,\ell} = (u^k + \text{LayerNorm}(y^k_i))_{i=1,\ldots,\ell} \tag{4}$$

where y^k_i is the i-th element of the data tensor y^k, selected from the second dimension . For simplicity, we omitted the batch dimension. Figure 2 illustrates the architecture of the CAPE module.

3 Experiments

We used datasets provided by PDEBench (Takamoto et al., 2022) a benchmark for SciML from which we selected the following PDEs: 1D Advection equation, 1D Burgers equation, 2D Compressible Navier-Stokes equations (2D NS). For 1-dimensional PDEs, we used $N = 9000$ training instances and 1000 test instances for each PDE parameter with resolution 128. For 2-dimensional NS equations, we used $N = 900$ training instances and 100 test instances for each PDE parameter with spatial resolution 64×64. The training was performed on GeForce RTX 2080 GPU for 1D PDEs and GeForce GTX 3090 for 2D NS equations.

Experiment Setup. We evaluated the neural models U-Net (Ronneberger et al., 2015) and FNO (Li et al., 2021a) with some datasets provided by PDEBench (Takamoto et al., 2022) for various parameters for the 1D Advection equation, 1D Burgers equation, and 2D compressible Navier-Stokes equations. We trained each of the neural models (1) **Base**: without any changes (vanilla model), (2)

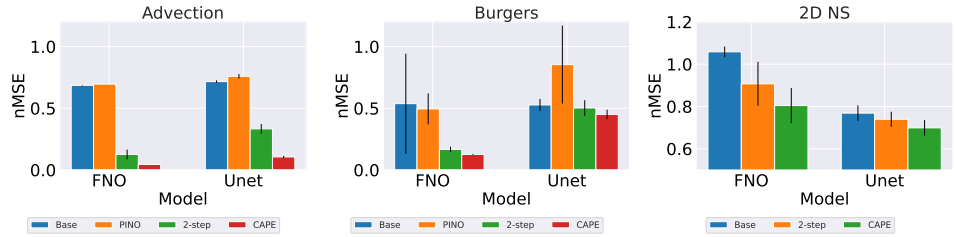

Figure 3: Plots of the normalized MSE (smaller is better) with an error bar for Advection eq. (Left), Burgers eq. (Middle), and 2D Compressible NS equations (Right). **The smaller, the better.**

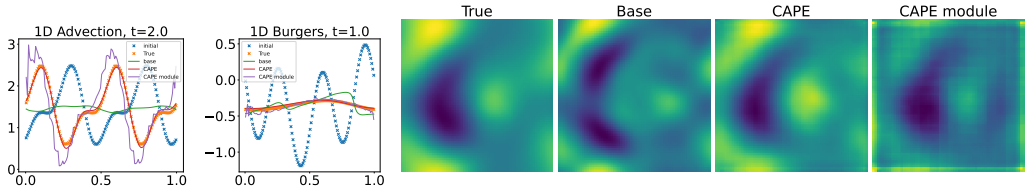

Figure 4: Visualization of the results: Advection eq. at the final time-step ($t = 2.0$) (Left), Burgers eq. at $t_k = 20(t = 1.0)$ (2nd-left) at the final time-step, and V_x of 2D NS equations at $t_k = 5(t = 0.25)$ (Right). Here "CAPE module" is the direct output from CAPE module only.

PINO: with a PINO loss (Li et al., 2021b), (3) **2-step**: with the field data for the current and previous time-steps as input (u^k, u^{k-1}), and (4) **CAPE**: with the CAPE module. Other than case (3), we only provided field data for one time step to the models and, therefore, the models cannot obtain PDE parameters' information from the given data. Since the solutions of each PDE are not normalized, we measure the normalized MSE (nMSE). The optimization was performed with Adam (Kingma & Ba) for 100 epochs. The learning rate was divided by 2.0 every 20 epochs. For a fair comparison, we made the model size of the different methods as similar as possible.

Varying the parameter values. Figure 3 shows bar plots comparing the BASE models with and without CAPE module, the models with PINO loss, and the models with the 2-steps as input. The CAPE module results in the lowest error in all cases. In particular, the CAPE module leads to an impressible error reduction ranging from 20 % (2D NS equation) to 95 % (1D Advection). We partly attribute this to the BASE network's ability to capture physical dynamics from the PDE parameter-dependent data provided by the CAPE module. The vanilla FNO is a state-of-the-art model and is superior to the U-net as a BASE model . Interestingly, the CAPE module provides either comparable or a little better results than the case with 2-step information. This indicates that the CAPE module succeeded in providing equivalent and even more useful information to the BASE network.

Qualitative analysis of the CAPE module. In Figure 4 we plot some representative outputs of the vanilla FNO, the CAPE module, and the overall CAPE model, and compare them with the true solutions. Interestingly, we can see that the BASE network often interpolates a higher noise approximation of the CAPE module into the typical shape (style) of the final solution.

4 Conclusion

This paper proposed a channel-attention-based parameter embedding (CAPE) module which allows any data-driven SciML models to incorporate PDE parameters. We also propose a simple but effective curriculum training strategy that allows us to bridge teacher-forcing and auto-regressive learning. We performed an extensive set of experiments and showed the effectiveness and efficiency of our method from various aspects: generalization of seen/unseen PDE parameters during training. For the moment, our method is limited to hydrodynamic-type field equations with structured meshes, and only PDE parameters can be taken into account.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Please see the last part of Section 4.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We believe that our work does not have any potential negative societal impacts as it does not contain confidential or private data.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We only used data of general physical systems with no potential ethical issues or severe environmental damage.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provided our anonymous Github URL.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We tried to include them as much as possible.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Please see Figure 3.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] They are provided in the first part of Section 3.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We adopted the implementation of the baseline models with some modifications, with proper citation and credits to the authors, as well as existing software packages
   (b) Did you mention the license of the assets? [Yes] All the appropriate licenses are provided or mentioned in our repository.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] All the code and a part of the data are included in the code repository
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We used the data generated by numerical simulations which is accessible to the public, and we include citations to the original authors.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We do not include any personal information or offensive content in our codes.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 481–490, 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural pde solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer graphics forum*, volume 38, pp. 59–70. Wiley Online Library, 2019.

Diederik P. Kingma and Jimmy Ba. In *ICLR (Poster)*.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *International Conference on Learning Representations (ICLR)*, 2021a.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021b.

Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):1–10, 2018.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Makoto Takamoto, Timothy Pradita, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: A diverse and comprehensive benchmark for scientific machine learning, 2022. URL `https://darus.uni-stuttgart.de/privateurl.xhtml?token=1be27526-348a-40ed-9fd0-c62f588efc01`.

Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1457–1466, 2020.