Compute-Adaptive Surrogate Modeling of Partial Differential Equations

Payel Mukhopadhyay[1,2,4], **Michael McCabe**[1,3], **Ruben Ohana**[1,3], **Miles Cranmer**[1,4]
[1]Polymathic AI
[2]University of California, Berkeley
[3]Flatiron Institute
[4]University of Cambridge
pmukho@berkeley.edu

Abstract

Modeling dynamical systems governed by partial differential equations presents significant challenges for machine learning-based surrogate models. While transformers have shown potential in capturing complex spatial dynamics, their reliance on fixed-size patches limits flexibility and scalability. In this work, we introduce two convolutional encoder and decoder architectural blocks—Convolutional Kernel Modulator (CKM) and Convolutional Stride Modulator (CSM)—designed for patch embedding and reconstruction in autoregressive prediction tasks. These blocks unlock dynamic patching and striding strategies to balance accuracy and computational efficiency during inference. Furthermore, we propose a rollout strategy that adaptively adjusts patching and striding configurations throughout temporally sequential predictions, mitigating patch artifacts and long-term error accumulation while improving the capture of fine-scale structures. We show that our approaches enable dynamic control over patch sizes at inference time without losing accuracy over fixed patch baselines.

1 Introduction

Numerical methods remain the gold standard for simulating partial differential equations (PDEs), offering convergence guarantees and resolution-adaptive accuracy-compute trade-offs (Morton & Mayers, 2005; Malalasekera, 2007). However, resolution choices are constrained by physical necessity and available compute (Berger & Colella, 1989; Wedi, 2014). Deep learning, and recently the class of models known as "vision transformers", has emerged as a powerful strategy for surrogate modeling of PDEs (Cao, 2021b; Li et al., 2023; McCabe et al., 2024; Herde et al., 2024). Often originating from computer vision methods, Vision Transformers (ViTs) (Dosovitskiy et al., 2020) segment discretized fields into distinct, non-overlapping patches, or tokens, serving as transformer inputs. This downsampling operation is necessary due to self-attention complexity scaling quadratically with token count. While initial training costs are expensive, the overall cost can be amortized across inference tasks such as forecasting (Bi et al., 2023), PDE-constrained optimization (Li et al., 2022), and parameter inference (Cranmer et al., 2020; Lemos et al., 2023).

However, static tokenization, as typically done in PDE surrogate models based on ViTs, limits adaptability in applications where downstream users must balance accuracy and compute. Patch size determines both accuracy and cost: smaller patches improve resolution but significantly increase compute demands (Fig. 3, 2). Fixed-patch models require separate training for different resolutions, creating an increasing burden as model scales grow. Additionally, fixed patching empirically has been shown to introduce visible artifacts during extended rollouts, where predictions recursively serve as inputs for future steps. As shown in Fig. 2, these artifacts appear spatially and as harmonic spikes in the power spectrum, degrading long-term prediction quality in vision transformer-based surrogates.

To address these concerns, we introduce *Controllable Resolution Transformers*, allowing for inference-time adaptation. To do this, we develop two strategies: *Convolutional Kernel Modulation (CKM)*, which utilizes the patch-resizing of FlexiViT (Beyer et al., 2023) in both the encoder and decoder, and *Convolutional Stride Modulation (CSM)*, which maintains a fixed convolutional kernel while modulating stride to control compression (Fig. 1). These

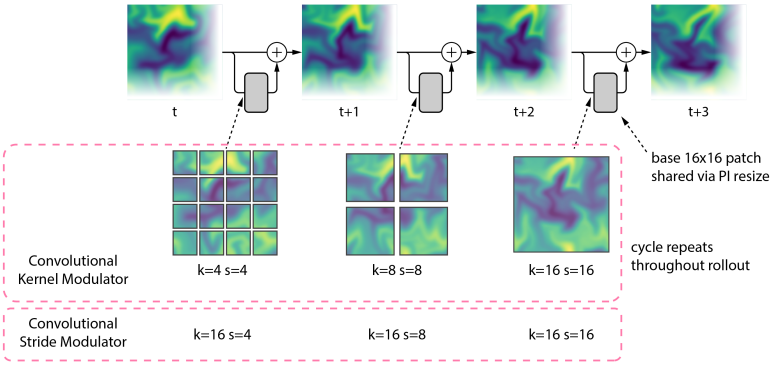

Figure 1: Overview of CKM and CSM approaches for adaptive patching in ViT-based PDE surrogates. Here, k and s denote convolutional kernel size and stride, which control downsampling. During inference, kernel or stride alternates cyclically between 4, 8, and 16 at each timestep of the rollout.

controllable resolution transformers have an encoder–processor–decoder (Sanchez-Gonzalez et al., 2020) architecture. The encoder and decoder are responsible for embedding and de-embedding of the extracted patches, and the processor is based on a ViT architecture adapted to spatio-temporal datasets (See App. B.2). CKM and CSM represent two different ways of adjusting both the encoder and the decoder to enable inference time scaling.

Contributions. Our primary contribution in this paper is to demonstrate an adaptive patching technique that can greatly benefit transformer-based surrogate models for spatio-temporal dynamics. Using two different adaptive tokenization techniques, we show that these methods enable controllable compute-accuracy trade-offs at inference, outperform fixed patch-based transformers, and improve rollout quality for transformer-based surrogate models. Our contributions are summarized below:

1. **Controllable Tokenization:** We introduce CKM and CSM for adaptive tokenization, allowing test-time control over compute-accuracy trade-offs.
2. **Patch Artifact Reduction:** We propose a new rollout strategy alternating between different patch sizes to mitigate artifacts (Fig. 2), significantly improving rollout predictions.
3. **Numerical Experiments:** We evaluate CKM and CSM on 2D and 3D physics datasets from The Well collection (Ohana et al., 2024), demonstrating improved accuracy and efficiency over fixed-patch models. See App. B.1 for dataset specifics.

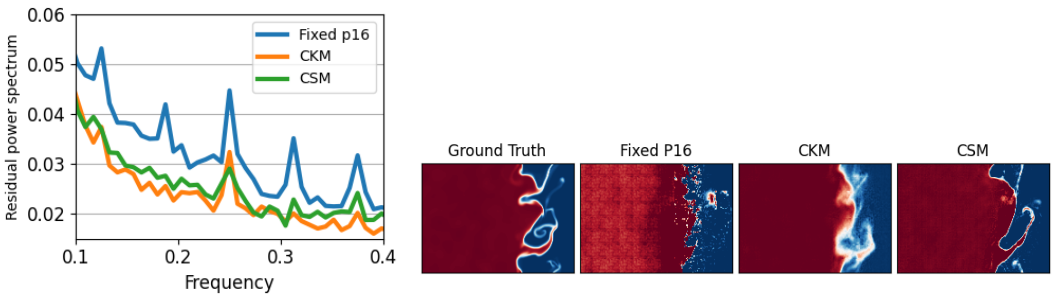

Figure 2: **Left:** Residual power spectrum of the density field at the 20th rollout step for CKM, CSM, and fixed patch 16 (P16) on the `turbulent_radiative_layer_2D` dataset, taken from (Ohana et al., 2024). See App. B.1 for dataset specifics. Harmonic spikes indicate patch-induced artifacts, significantly reduced in CKM and CSM. **Right:** Prediction at rollout step 20, where CKM and CSM mitigate visible patching effects in fixed patch 16 model.

2 Background and Related Work

Notations. We solve an autoregressive next-step prediction problem for a system $\mathcal{S}$ that evolves in space and time. Given past states $\mathbf{V}_t^{\mathcal{S}}$, the goal is to learn a model $\mathcal{F}$ that predicts the next state $\mathcal{F}(\mathbf{V}_t^{\mathcal{S}}) \approx \mathbf{v}_{t+\Delta t_{\mathcal{S}}}^{\mathcal{S}}$, analogous to video prediction. We define CKM/CSM as models using convolutional kernel/stride *modulation* in their encoder–decoder blocks. We emphasize again that both CKM and CSM form the encoder–decoder blocks of an encoder–processor–decoder architecture (See App. B.2). We will refer to the general class of models that allow controllable resolution at inference to be *compute-elastic* models.

Deep learning for PDE surrogate modeling. Several approaches leverage deep learning for PDE modeling, each with unique merits. Physics-informed neural networks (PINNs) (Raissi et al., 2019; Karniadakis et al., 2021; Hao et al., 2023) learn the solution directly, by embedding governing equations into the loss function, encouraging strict adherence to physics. CNNs and graph-based models (Brandstetter et al., 2022; Li & Farimani, 2022; Lam et al., 2022; Pfaff et al., 2021) enable learning on complex meshes. Neural operators (Li et al., 2020; Lu et al., 2019; Kovachki et al., 2021; Jin et al., 2022) approximate solution operators independently of resolution.

ViTs and compute-adaptive models for PDEs. ViTs (Dosovitskiy et al., 2020) are increasingly used in PDE surrogate modeling (Cao, 2021a; Li et al., 2023; Liu et al., 2023) but traditionally rely on fixed patch sizes, limiting flexibility and scalability. FlexiViT (Beyer et al., 2023) introduced adaptive patching to dynamically adjust resolution based on compute constraints, a concept explored in computer vision but largely absent in PDE modeling. Another important work in this area, (Zhang et al., 2024), adapts patchification based on data, compared to our approach, which provides direct control over compute requirements.

3 Flexible encoder & decoder frameworks

Transformer architecture. As mentioned before, both CKM and CSM represent two different methods of adjusting the encoder–decoder blocks of a ViT based architecture to enable inference time scaling. The transformer processor consists of spatio-temporal blocks performing temporal self-attention, spatial self-attention, and an MLP sequentially (see App. B.2 for details). Note that the both CSM and CSM patching strategies can be used with any convolution based encoder–decoder framework.

Advantages of compute-elastic models. Increasing the number of tokens, equivalent to using smaller patches, improves accuracy but increases compute cost (Fig. 3). Fixed-patch models lack inference-time scalability and introduce artifacts during long rollouts (Fig. 2), motivating compute-elastic models for ViT-based PDE modeling. Scalability is crucial for transformer-based surrogates, which benefit from parameter scaling (Zhai et al., 2022; Hoffmann et al., 2022). Instead of training separate fixed-patch models, compute-elastic architectures offer a single adaptable model, provided they match or surpass fixed-patch performance across all token counts at inference. We also assess whether variable-token training improves rollouts over static patch models. If these benefits hold, compute-elastic models will be critical for large-scale ViT PDE surrogates.

Patching. ViTs segment an input $\mathbf{x} \in \mathbb{R}^{h \times w \times c}$ into patches $\mathbf{x}_i \in \mathbb{R}^{p \times p \times c}$, where h, w, c and p denote the height, width, number of channels and patch size respectively. Convolutional patching extracts patches via a kernel-stride pair (k, s), determining token count:

$$N_h = \left\lfloor \frac{h-k}{s} \right\rfloor + 1, \quad N_w = \left\lfloor \frac{w-k}{s} \right\rfloor + 1. \tag{1}$$

where N_h and N_w are the number of patches along the height and width. The total number of patches, or tokens is $N = N_h \cdot N_w$. In standard ViTs, $k = s$, but this is not necessary. Adjusting k and s, allows control over patch size and consequently, the token count.

Convolutional Kernel Modulator (CKM). CKM dynamically adjusts convolutional kernel size, k, at each forward pass, allowing models to operate at downsampling and upsampling resolutions of 4, 8, or 16 pixels. Each forward pass randomly samples a kernel from $\{4, 8, 16\}$ for downsampling in the encoder. The kernel weight matrix of the base architecture is projected onto the sampled kernel size using PI–resize transformation (Beyer et al., 2023), denoted by $B^{T\dagger}$. This transformation, denoted by $B^{T\dagger}$, uses an interpolation matrix B —bicubic for 2D and trilinear for 3D—to resize input patches while preserving the

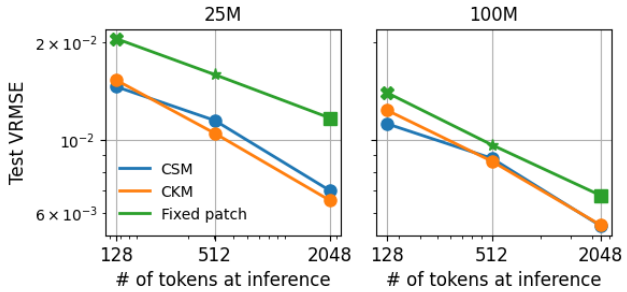

Figure 3: Comparison of next step test VRMSE as a function of the number of tokens at inference for convolutional kernel modulation models (CKM), convolutional stride modulation models (CSM), and individually trained fixed patch models (25M and 100M parameters) for the `shear_flow` dataset.

# Tokens	25M VRMSE			100M VRMSE			CNextU-net (20M) (Ohana et al., 2024)
	CKM	CSM	Fixed p.s.	CKM	CSM	Fixed p.s.	
2048	**0.0065**	0.0070	0.0117	**0.0055**	0.0055	0.0067	
512	**0.0105**	0.0115	0.0159	**0.0086**	0.0088	0.0096	0.8080
128	0.0153	**0.0146**	0.0205	0.0120	**0.0110**	0.0140	

Table 1: Next-step VRMSE test set results on the `shear_flow` dataset for models with 25M and 100M parameters, evaluated at different token counts during inference. We compare CKM, CSM, and fixed patch size (p.s.) models, alongside the CNextU-net baseline from (Ohana et al., 2024).

effects of the original convolution. By solving a least-squares optimization problem, PI–resize ensures minimal discrepancy between features extracted at different patch sizes (see App. A for details). The CKM encoder encodes *non-overlapping* tokens which gets processed by transformer blocks before reconstruction by the CKM decoder. Unlike FlexiViT (Beyer et al., 2023), CKM includes both an encoder and decoder for staged downsampling and upsampling, essential for autoregressive forecasting. See App. C.1 for algorithm.

Convolutional Stride Modulator (CSM). In CSM, the kernel size, k is kept fixed, and the stride, s is randomly adjusted at each forward pass to control downsampling and upsampling. With a fixed kernel and a randomly selected stride, the encoder generates an *overlapping* set of tokens. One limitation of this approach over CKM is that it introduces overlap with the boundary. To account for this overlap, learned padding tokens are introduced in the original space. The overlapping tokens are then processed by a transformer blocks before being reconstructed by the CSM decoder. App. C.2 shows an algorithm.

New rollout method. Unlike standard rollouts that use a fixed k and s for the encoding/decoding blocks throughout the prediction horizon, our approach dynamically alternates between different kernel or stride sizes in a cyclic fashion. Specifically, for rollouts of CKM model, we alternate between k of 4, 8, and 16 in a repeating sequence: $4 \to 8 \to 16 \to 4 \to 8 \to 16$, and so on. Similarly, CSM follows the same alternating pattern for stride values, effectively achieving downsampling factors of 4, 8, and 16 in a cyclic manner. This rollout mechanism is visualized in Fig. 1.

4 Experimental Results

To evaluate the performance of our compute-elastic CKM and CSM models, we conduct experiments on a range of complex 2D and 3D datasets from The Well (Ohana et al., 2024). See App. B.1 for dataset specifics. These experiments fall into two categories:

1. We compare the efficiency of training multiple fixed-patch models vs. a single compute-elastic model for scalable inference. To this end, we train three fixed-patch models (patch sizes 4, 8, and 16) for 200 epochs each (see Appendix B.3), then train both CKM and CSM models using the combined compute budget of these fixed-patch models. This setup reflects a practical constraint: given a fixed training budget, should resources be allocated to separate fixed-patch models or a single compute-elastic model? We evaluate inference performance to determine whether CKM and CSM offer advantages over static patching.

2. We assess whether CKM and CSM reduce visible patch artifacts in extended rollouts (Sec.4.2). In all experiments, the input spatio-temporal sequence consists of six time frames, with the loss function optimized for next-step prediction.

4.1 Prediction accuracy and inference time scalability

VRMSE error. We choose the VRMSE metric (App. D) to compare our results with recent work on these datasets (Ohana et al., 2024). Fig. 3 presents a comparative analysis of the one-step VRMSE for the `shear_flow` dataset for different number of tokens being processed. Note that the number of tokens in this plot is a measure of the compute requirement at inference. Two models of size $\sim$ 25M and 100M parameters are studied. The results show that CKM and CSM consistently outperform fixed patch-size models across a range of token counts, maintaining their advantage in both small (25M) and large (100M) models. This demonstrates that compute-elastic models capture dynamics more effectively while reducing errors compared to standard fixed patching methods. Additionally, as shown in Table 1, all our models outperform the best benchmarks from (Ohana et al., 2024) by large margins.

Dataset	# Tokens	7.6M Model			100M Model			Benchmark (20M) (Ohana et al., 2024)
		CKM	CSM	Fixed p.s.	CKM	CSM	Fixed p.s.	
`active_matter`	4096	**0.0405**	**0.0400**	0.0409	0.0192	**0.0172**	0.021	CNextU-net 0.1034
	1024	0.046	0.050	**0.043**	0.022	**0.021**	0.024	
	256	0.070	0.066	**0.050**	0.034	**0.029**	0.031	
`rayleigh_benard`	4096	0.046	**0.044**	0.061	**0.025**	**0.025**	0.031	TFNO 0.6566
	1024	**0.060**	**0.060**	0.073	**0.033**	0.035	0.036	
	256	0.080	**0.074**	0.083	0.047	**0.046**	0.047	
`turbulent_radiative_layer_2D`	3072	0.187	**0.180**	0.190	**0.133**	0.144	0.143	CNextU-net 0.1956
	768	0.210	0.210	0.210	**0.153**	0.178	0.170	
	192	0.254	**0.245**	0.26	**0.200**	0.202	0.223	
`supernova_explosion`	1024	**0.258**	0.270	0.261	–	–	–	U-net 0.3063
	256	**0.328**	0.343	0.337	–	–	–	
	64	**0.364**	0.370	0.367	–	–	–	
`turbulence_gravity_cooling`	1024	**0.096**	0.102	0.100	–	–	–	CNextU-net 0.2096
	256	0.138	0.152	**0.133**	–	–	–	
	64	0.169	0.169	**0.164**	–	–	–	

Table 2: Test VRMSE for different numbers of tokens at inference across multiple 2D and 3D datasets for CKM, CSM, and fixed patch size (p.s.) models. Results are for next-step prediction. Due to compute constraints, 3D datasets were only trained on the 7.6M model. We also include benchmark results from (Ohana et al., 2024). Note that our models use a time context size of 6, while The Well benchmarks were trained with a time context of 4.

Table 2 extends the analysis by comparing Test VRMSE performance for additional 2D datasets: `active_matter`, `rayleigh_benard`, `turbulent_radiative_layer_2D`. CKM and CSM models consistently achieve VRMSE values on par or lower compared to fixed patch models. Similarly to the `shear_flow` dataset, we note again in Table 2 that even our smallest models with $\sim$ 7.6M parameters greatly outperform or are very competitive with the best benchmarks in the Well paper (Ohana et al., 2024). Table 2 also shows results for highly complex 3D physical simulations, `supernova_explosion` and `turbulence_gravity_cooling` datasets for a model size of $\sim$ 7.6M parameters. For both of these complex 3D datasets, CKM and CSM models achieves similar accuracy to fixed patch size models, again showing that compute elastic models can enable inference time scalability without compromising accuracy.

Binned spectral error. The Binned Spectral Normalized Mean Squared Error (BSNMSE) measures how well the model predictions capture the ground truth values in frequency space, averaged in low, mid and high frequency bins. see App. E for details of the metric. Lower values imply better predictions of the ground truth in frequency space. Fig. 4 shows BSNMSE analysis on the `shear_flow` dataset, where CKM and CSM models consistently match or outperform fixed-patch models. Similar trends are observed across other datasets (see App. F), demonstrating that compute-elastic models resolve features across all frequency scales as well as, or better than, fixed-patch models—while also enabling scalable inference.
Dynamic inference-time resource scaling has not been explored in the PDE emulator literature. While large language models use optimizations like entropy-based techniques, PDE surrogates

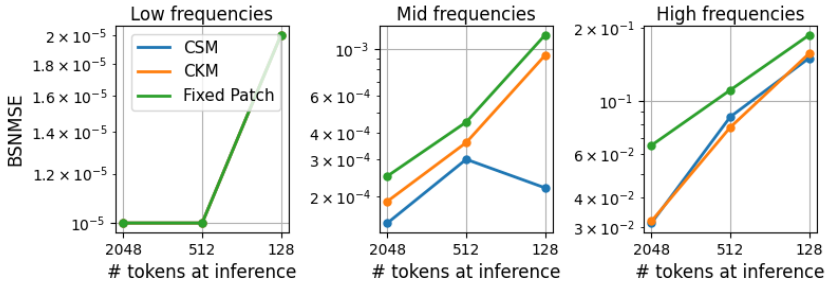

Figure 4: BSNMSE of next-step predictions on the `shear_flow` test set. The spectrum is divided into low, mid, and high frequency bins, with boundaries evenly spaced in log scale.

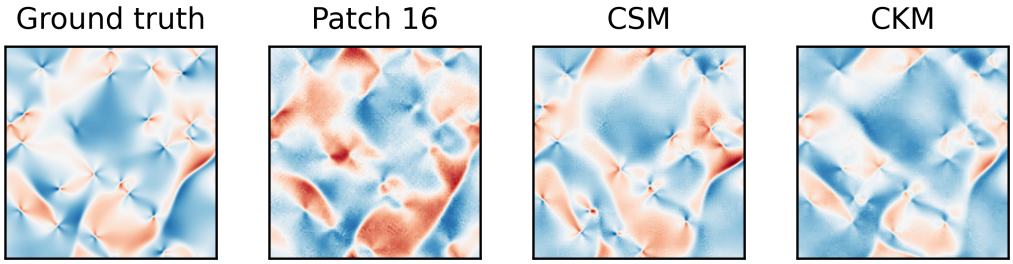

Figure 5: Step 45 rollout of the E_{xx} field (rate of strain tensor) in the `active_matter` dataset. Patch artifacts accumulate over time, degrading prediction quality, with the effect most pronounced in the fixed patch 16 model—the largest tested patch size.

lack such flexibility. Our experiments show that CKM and CSM not only allow scalable inference but also match or exceed fixed-patch performance across all tested compute regimes.

4.2 Improved rollouts with compute elastic models

During inference rollouts, we alternate at each prediction step between patch sizes and strides of 4, 8, and 16 for CKM and CSM models respectively. This alternating patching and striding strategy significantly improves rollout accuracy compared to fixed patch size models. Fig. 2 shows that fixed patch sizes lead to distinct patch artifacts visible as peaks at harmonic frequencies in the residual power spectrum. These artifacts arise from the rigid nature of fixed patching and degrade the model's ability to capture fine-scale features over extended rollouts. In contrast, by allowing to alternate patches during rollouts, CKM and CSM models drastically decrease the patch-artifacts, resulting in a suppression of the harmonic peaks and more accurate predictions. Another example of this phenomenon is shown in Fig. 5 for the `active_matter` dataset, where CSM and CKM (CSM especially in this case) produces better quality rollout at step 45 than the fixed patch 16 model (For an additional demonstration of this rollout mechanism see App. G)

Conclusion

We introduced compute-elastic techniques for surrogate modeling of PDEs, demonstrating their advantages through extensive experiments. Our findings show that CKM and CSM models enhance accuracy by outperforming fixed-patch models across various complex 2D and 3D next-step prediction tasks. Additionally, the ability to dynamically adjust computational cost at inference time provides downstream users with a flexible trade-off between speed and precision, a critical feature as ViT-based architectures scale toward foundation models for PDEs. Beyond accuracy and efficiency, these models significantly improve rollout quality by reducing artifacts inherent to fixed-patch ViTs, addressing a key limitation in vision-based PDE emulation. These results highlight the importance of compute-adaptive patching techniques as a fundamental design consideration for future transformer architectures in PDE surrogate modeling.

References

Marsha J Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of computational Physics*, 82(1):64–84, 1989.

Lucas Beyer, Pavel Izmailov, Alexander Kolesnikov, Mathilde Caron, Simon Kornblith, Xiaohua Zhai, Matthias Minderer, Michael Tschannen, Ibrahim Alabdulmohsin, and Filip Pavetic. FlexiVIT: One model for all patch sizes. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2023. doi: 10.1109/cvpr52729.2023.01393. URL `https://doi.org/10.1109/cvpr52729.2023.01393`.

Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Accurate medium-range global weather forecasting with 3d neural networks. *Nature*, 619(7970): 533–538, 2023.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Keaton J Burns, Geoffrey M Vasil, Jeffrey S Oishi, Daniel Lecoanet, and Benjamin P Brown. Dedalus: A flexible framework for numerical simulations with spectral methods. *Physical Review Research*, 2(2):023068, 2020.

Shuhao Cao. Choose a transformer: Fourier or galerkin, 2021a.

Shuhao Cao. Choose a transformer: Fourier or galerkin. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 24924–24940. Curran Associates, Inc., 2021b. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/d0921d442ee91b896ad95059d13df618-Paper.pdf`.

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Drummond B Fielding, Eve C Ostriker, Greg L Bryan, and Adam S Jermyn. Multiphase gas and the fractal nature of radiative turbulent mixing layers. *The Astrophysical Journal Letters*, 894(2):L24, 2020.

Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. Physics-informed machine learning: A survey on problems, methods and applications, 2023.

Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient Foundation Models for PDEs. *arXiv preprint arXiv:2405.19101*, 2024.

Keiya Hirashima, Kana Moriwaki, Michiko S Fujii, Yutaka Hirai, Takayuki R Saitoh, and Junichiro Makino. 3d-spatiotemporal forecasting the expansion of supernova shells using deep learning towards high-resolution galaxy simulations. *Monthly Notices of the Royal Astronomical Society*, 526(3):4054–4066, 2023a.

Keiya Hirashima, Kana Moriwaki, Michiko S Fujii, Yutaka Hirai, Takayuki R Saitoh, Junichiro Makino, and Shirley Ho. Surrogate modeling for computationally expensive simulations of supernovae in high-resolution galaxy simulations. *arXiv preprint arXiv:2311.08460*, 2023b.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Pengzhan Jin, Shuai Meng, and Lu Lu. Mionet: Learning multiple-input operators via tensor product. *SIAM Journal on Scientific Computing*, 44(6):A3490–A3514, 2022.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Alexander Pritzel, Suman Ravuri, Timo Ewalds, Ferran Alet, Zach Eaton-Rosen, Weihua Hu, Alexander Merose, Stephan Hoyer, George Holland, Jacklynn Stott, Oriol Vinyals, Shakir Mohamed, and Peter Battaglia. Graphcast: Learning skillful medium-range global weather forecasting, 2022.

Pablo Lemos, Liam Parker, ChangHoon Hahn, Shirley Ho, Michael Eickenberg, Jiamin Hou, Elena Massara, Chirag Modi, Azadeh Moradinezhad Dizgah, Bruno Regaldo-Saint Blancard, and David Spergel. Simbig: Field-level simulation-based inference of galaxy clustering, 2023.

Jichao Li, Xiaosong Du, and Joaquim RRA Martins. Machine learning in aerodynamic shape optimization. *Progress in Aerospace Sciences*, 134:100849, 2022.

Zijie Li and Amir Barati Farimani. Graph neural network-accelerated lagrangian fluid simulation. *Computers & Graphics*, 103:201–211, 2022. ISSN 0097-8493. doi: https://doi.org/10.1016/j.cag.2022.02.004. URL `https://www.sciencedirect.com/science/article/pii/S0097849322000206`.

Zijie Li, Dule Shu, and Amir Barati Farimani. Scalable transformer for PDE surrogate modeling. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=djyn8Q0anK`.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Xinliang Liu, Bo Xu, and Lei Zhang. HT-net: Hierarchical transformer based operator learning model for multiscale PDEs, 2023. URL `https://openreview.net/forum?id=UY5zS0OsK2e`.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Suryanarayana Maddu, Scott Weady, and Michael J Shelley. Learning fast, accurate, and stable closures of a kinetic theory of an active fluid. *Journal of Computational Physics*, 504:112869, 2024.

Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Prentice Hall, 2007.

Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, Francois Lanusse, et al. Multiple physics pretraining for spatiotemporal surrogate models. In *NeurIPS*, 2024. URL `https://neurips.cc/virtual/2024/poster/96095`.

Keith W Morton and David Francis Mayers. *Numerical solution of partial differential equations: an introduction*. Cambridge university press, 2005.

Ruben Ohana, Michael McCabe, Lucas Meyer, Rudy Morel, Fruzsina J. Agocs, Miguel Beneitez, Marsha Berger, Blakesley Burkhart, Stuart B. Dalziel, Drummond B. Fielding, Daniel Fortunato, Jared A. Goldberg, Keiya Hirashima, Yan-Fei Jiang, Rich R. Kerswell, Suryanarayana Maddu, Jonah Miller, Payel Mukhopadhyay, Stefan S. Nixon, Jeff Shen, Romain Watteaux, Bruno Régaldo-Saint Blancard, François Rozet, Liam H. Parker, Miles Cranmer, and Shirley Ho. The well: a large-scale collection of diverse physics simulations for machine learning. In *38th Conference on Neural Information Processing Systems*, 2024. URL `https://neurips.cc/virtual/2024/poster/97882`.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks, 2021.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020.

Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Jakob Verbeek, and Herve Jegou. Three things everyone should know about vision transformers. *arXiv preprint arXiv:2203.09795*, 2022.

Nils P Wedi. Increasing horizontal resolution in numerical weather prediction and climate simulations: illusion or panacea? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2018):20130289, 2014.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12104–12113, 2022.

Pei Zhang, M. Paul Laiu, Matthew Norman, Doug Stefanski, and John Gounley. Matey: multiscale adaptive foundation models for spatiotemporal physical systems, 2024. URL `https://arxiv.org/abs/2412.20601`.

A Derivation of the PI-Resize Matrix for Convolutional Kernel Modulators

A.1 Motivation for Kernel Resizing

The CKM-based model dynamically adjusts convolutional kernel sizes at each forward pass while maintaining a fixed architecture. This enables variable patch sizes $(4, 8, 16)$, which is crucial for handling different spatial resolutions while ensuring compatibility with the ViT. However, this flexibility introduces a problem: the CNN encoder, which extracts ViT tokens, is trained with a *fixed base convolutional kernel*. To ensure that features extracted at different patch sizes remain aligned, we use a **resize transformation**, which projects the base kernel to a dynamically selected kernel size (Beyer et al., 2023).

A.2 Mathematical Formulation

Let:

- $W^{\text{base}} \in \mathbb{R}^{k^{\text{base}} \times k^{\text{base}} \times c_{\text{in}} \times c_{\text{out}}}$ be the learned CNN weights for the **fixed base kernel size** k^{base}.
- $W \in \mathbb{R}^{k \times k \times c_{\text{in}} \times c_{\text{out}}}$ be the resized kernel for a dynamically selected kernel size k.
- B be an **interpolation matrix** (e.g., bilinear, bicubic) that resizes **input patches** from size k to size k^{base}.
- $x \in \mathbb{R}^{B \times k \times k \times c_{\text{in}}}$ be an **input patch**, where:
  - B is the batch size.
  - $k \times k$ is the **local receptive field** defined by the kernel.
  - c_{in} is the number of input channels.

Intuitively, the goal is to find a new set of patch- embedding weights W such that the tokens of the resized patch match the tokens of the original patch

$$W^{\text{base}} * x \approx W * (Bx), \tag{2}$$

where $*$ denotes convolution. The goal is to solve for W. Mathematically, this is an optimization problem.

A.3 Least-Squares Optimization for PI-Resize

A.3.1 Expanding the Objective

We aim to expand the expectation:

$$\mathbb{E}_{x \sim \mathcal{X}} \left[(x^T W^{\text{base}} - x^T B^T W)^2 \right]. \tag{3}$$

Since inner products can be rewritten as matrix-vector multiplications, we note that:

$$\langle x, W^{\text{base}} \rangle = x^T W^{\text{base}}.$$

Step 1: Expand the Square Using the identity $(a - b)^2 = a^2 - 2ab + b^2$, we expand:

$$(x^T W^{\text{base}} - x^T B^T W)^2 = (x^T W^{\text{base}})^2 - 2x^T W^{\text{base}} (x^T B^T W) + (x^T B^T W)^2. \tag{4}$$

Step 2: Take the Expectation Now, we take expectation over $x \sim \mathcal{X}$:

$$\mathbb{E}_{x \sim \mathcal{X}} \left[(x^T W^{\text{base}})^2 - 2x^T W^{\text{base}} (x^T B^T W) + (x^T B^T W)^2 \right]. \tag{5}$$

Using the definition of the covariance matrix:

$$\Sigma = \mathbb{E}_{x \sim \mathcal{X}}[x x^T], \tag{6}$$

we apply the linearity of expectation to each term:

$$\mathbb{E}_x[(x^T W^{\text{base}})^2] = W^{\text{base},T} \Sigma W^{\text{base}}, \tag{7}$$

$$\mathbb{E}_x[(x^T B^T W)^2] = W^T B \Sigma B^T W, \tag{8}$$

$$\mathbb{E}_x[x^T W^{\text{base}} (x^T B^T W)] = W^{\text{base},T} \Sigma B^T W. \tag{9}$$

Step 3: Write in Matrix Form Substituting these into the expectation:

$$\mathbb{E}_{x\sim\mathcal{X}}\left[(x^T W^{\text{base}} - x^T B^T W)^2\right] = W^{\text{base},T}\Sigma W^{\text{base}} - 2W^{\text{base},T}\Sigma B^T W + W^T B\Sigma B^T W. \quad (10)$$

The above expression can be re-written as:

$$\|W^{base} - B^T W\|_\Sigma^2 = (W^{base,T} - W^T B)\Sigma(W^{base} - B^T W). \quad (11)$$

Conclusion This result expresses the squared error between the transformed weight matrix W and the resized weight matrix $B^T W^{\text{base}}$, weighted by the covariance matrix Σ. The quadratic form measures the deviation in terms of feature space alignment, ensuring that the transformation remains consistent with the learned base kernel.
The optimal solution minimizing Eq. 11 is given by

$$W = \left(\sqrt{\Sigma}B^T\right)^\dagger \sqrt{\Sigma}W^{base} \quad (12)$$

where $\left(\sqrt{\Sigma}B^T\right)^\dagger \sqrt{\Sigma}$ is the Moore-Penrose pseudoinverse matrix.

A.4 Final Transformation and Implementation

Thus, at each forward pass, we apply the following steps:

1. **Randomly select a kernel size** $k \in \{4, 8, 16\}$.
2. **Compute the interpolation matrix** B that resizes the input patch from k to k^{base}.
3. **Transform the kernel weights** using:

$$W = (B^T)^\dagger W^{\text{base}}. \quad (13)$$

4. **Apply the resized kernel** W for convolution.

This ensures that dynamically changing kernel sizes maintain compatibility with the fixed architecture while enabling flexible patch sizes.

B Experiment details

B.1 Datasets

The datasets that we benchmarked are taken from The Well collection Ohana et al. (2024). We selected 2D and 3D datasets with complex dynamics, ranging from biology (`active_matter` Maddu et al. (2024)) to astrophysics (`supernova_explosion` & `turbulence_gravity_cooling` Hirashima et al. (2023a;b); `turbulent_radiative_layer_2D` Fielding et al. (2020)) and fluid dynamics (`rayleigh_benard` & `shear_flow` Burns et al. (2020)) . Here is information about their resolution and physical fields (equivalent to a number of channels).

Dataset	Resolution	# of fields/channels
`active_matter`	256×256	11
`rayleigh_benard`	512×128	4
`shear_flow`	128×256	4
`supernova_explosion`	$64 \times 64 \times 64$	6
`turbulence_gravity_cooling`	$64 \times 64 \times 64$	6
`turbulent_radiative_layer_2D`	128×384	5

Table 3: Specifics of the datasets benchmarked.

B.2 Model configuration

The core transformer processor consists of multiple stacked processing blocks, each composed of three key operations: (1) temporal self-attention, which captures dependencies across time steps, (2) spatial self-attention, which extracts spatial correlations within each frame, and (3) a multi-layer perceptron (MLP) for feature transformation. The number of such processing blocks varies across model scales, as detailed in Table 4. The full configuration is vizualized in Fig. 6.

Table 4: Details of the various model architectures and scales explored.

Model	Embedding Dim.	MLP Dim.	# Heads	# Blocks	Base down/up sampling rate of patch encoder/decoder
7.6M	192	768	3	12	16
25M	384	1536	6	12	16
100M	768	3072	12	12	16

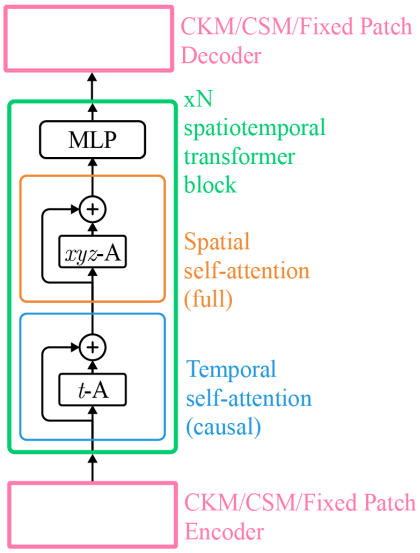

Figure 6: This visualization represents the overall architecture of our models. The CKM/CSM/Fixed patch are various strategies for the patch encoding and decoding. The patch encoder block is followed by N transformer based spatio-temporal blocks containing sequential time attention, followed by full spatial attention, followed by MLP. The embedding dimension column in table 4 represents the hidden dimension that the CKM/CSM/Fixed patch strategies embed the real space patches into.

B.3 Training configurations

In the following we list the training parameters:

1. Batch size is set at 2 and the model and we train on 8 GPUs using PyTorch FSDP.
2. Adam Optimizer. learning rate of 10^{-3}, weight decay of 10^{-4}.
3. All models and datasets were trained using Mean Squared Error averaged over fields and space during training.
4. Prediction type: delta. "delta" predicts the change in the field from the previous timestep. This only affects training since validation and test losses are computed on reconstructed fields.

C Algorithm of CKM and CSM

C.1 Convolutional Kernel Modulator (CKM)

CKM dynamically adjusts kernel sizes for patch embedding and reconstruction. At each forward pass, a kernel size $k \in \{4, 8, 16\}$ is sampled, and the base kernel weights are resized using PI–resize (Beyer et al., 2023) to ensure alignment across resolutions. Algorithm is shown below in Algorithm 1.

C.2 Convolutional Stride Modulator (CSM)

CSM dynamically adjusts stride s to vary downsampling and upsampling, while keeping the kernel fixed. See Algorithm 2.

Algorithm 1 Convolutional Kernel Modulator (CKM)

Input: $x \in \mathbb{R}^{B\times H\times W\times T\times C}$
Output: $\hat{x} \in \mathbb{R}^{B\times H\times W\times 1\times C}$ (prediction at $T+1$)
`k`: kernel size, `w`: kernel weight matrix, `s`: stride
`k`=(k_1, k_2); `w`=(w_1, w_2); `s`=(s_1, s_2). Hierarchical convolution (Touvron et al., 2022)
$\mathtt{w}^{base} = (\mathrm{w}_1^{base}, w_2^{base})$. Weight matrix of the base backbone architecture
Downsampling (CKM encoder)

1. Randomly sample (k_1, k_2) for total downsampling of 4, 8, or 16.
2. Compute resized weights: $w_1 = B^{T\dagger} w_1^{\text{base}}, \quad w_2 = B^{T\dagger} w_2^{\text{base}}$.
3. Apply convolutions:

$$x \leftarrow \mathtt{Conv}_1(x, w_1, s_1 = k_1),$$
$$x \leftarrow \mathtt{Conv}_2(x, w_2, s_2 = k_2).$$

4. Store k_1, k_2 for decoding.

Tokens processed by transformer blocks–Temporal attention, Spatial attention, MLP (App. B.2)

Upsampling (CKM decoder)

1. Retrieve kernel sizes k_1, k_2 from metadata.
2. Apply transposed convolutions:

$$x \leftarrow \mathtt{ConvTranspose}_1(x, w_2, s = k_2),$$
$$\hat{x} \leftarrow \mathtt{ConvTranspose}_2(x, w_1, s = k_1).$$

return $\hat{x}$

Algorithm 2 Convolutional Stride Modulator (CSM)

Input: $x \in \mathbb{R}^{B\times H\times W\times T\times C}$
Output: $\hat{x} \in \mathbb{R}^{B\times H\times W\times 1\times C}$ (prediction at $T+1$)
`k`: kernel size, `s`: stride
`k`=(k_1, k_2); `s`=(s_1, s_2). Hierarchical convolution (Touvron et al., 2022)
$\mathtt{k}^{base} = (\mathrm{k}_1^{base}, k_2^{base})$. Kernel sizes of the base backbone architecture
Step 1: Padding

1. Pad x with learned tokens for boundary conditions.

Step 2: Encoding (CSM encoder)

1. Randomly sample strides for total downsampling in 4, 8 or 16.
2. Apply convolutions:

$$x \leftarrow \mathtt{Conv}_1(x, k_1^{\text{base}}, s = s_1),$$
$$x \leftarrow \mathtt{Conv}_2(x, k_2^{\text{base}}, s = s_2).$$

3. Store s_1, s_2 for decoding.

Tokens processed by transformer blocks–Temporal attention, Spatial attention, MLP (App. B.2)

Step 3: Decoding (CSM decoder)

1. Retrieve stored strides s_1, s_2.
2. Apply transposed convolutions:

$$x \leftarrow \mathtt{TransposeConv}_1(x, k_2^{\text{base}}, s = s_2),$$
$$\hat{x} \leftarrow \mathtt{TransposeConv}_2(x, k_1^{\text{base}}, s = s_1).$$

return $\hat{x}$

D VRMSE

The VRMSE metric is defined as the square root of the variance scaled mean squared error (VMSE), which is the MSE normalized by the variance of the truth

$$\mathrm{VMSE}(u, v) = \frac{\langle |u - v|^2 \rangle}{(\langle |u - \bar{u}|^2 \rangle + \epsilon)}.$$

Here, v is the prediction and u is the ground truth. We chose to report its square root variant, the VRMSE:

$$\mathrm{VRMSE}(u, v) = \frac{\langle |u - v|^2 \rangle^{1/2}}{(\langle |u - \bar{u}|^2 \rangle + \epsilon)^{1/2}}.$$

Note that, since $\mathrm{VRMSE}(u, \bar{u}) \approx 1$, having VRMSE > 1 indicates worse results than an accurate estimation of the spatial mean $\bar{u}$.

E Binned spectral analysis

The **binned spectral mean squared error (BSMSE)**, introduced in (Ohana et al., 2024), quantifies the mean squared error after applying a bandpass filter to the input fields over a specific frequency band $\mathcal{B}$. It is defined as:

$$\mathrm{BSMSE}_{\mathcal{B}}(u, v) = \langle |u_{\mathcal{B}} - v_{\mathcal{B}}|^2 \rangle, \tag{14}$$

where the bandpass-filtered field $u_{\mathcal{B}}$ is given by:

$$u_{\mathcal{B}} = \mathcal{F}^{-1} \left[\mathcal{F} \left[u \right] \mathbf{1}_{\mathcal{B}} \right]. \tag{15}$$

Here, $\mathcal{F}$ represents the discrete Fourier transform, and $\mathbf{1}_{\mathcal{B}}$ is an indicator function that selects frequencies within the band $\mathcal{B}$.

For each dataset, we define three disjoint frequency bands, $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$, which correspond to low, intermediate, and high spatial frequencies, respectively. These bands are determined by partitioning the wavenumber magnitudes evenly on a logarithmic scale.

The **binned spectral normalized mean squared error (BSNMSE)** is a normalized variant of the BSMSE metric, adjusting for the energy within each frequency band:

$$\mathrm{BSNMSE}_{\mathcal{B}}(u, v) = \frac{\langle |u_{\mathcal{B}} - v_{\mathcal{B}}|^2 \rangle}{\langle |v_{\mathcal{B}}|^2 \rangle}. \tag{16}$$

A BSNMSE value of 1 or greater indicates that the model performs worse than simply predicting zero coefficients at that scale.

F Binned spectral error analysis for more datasets

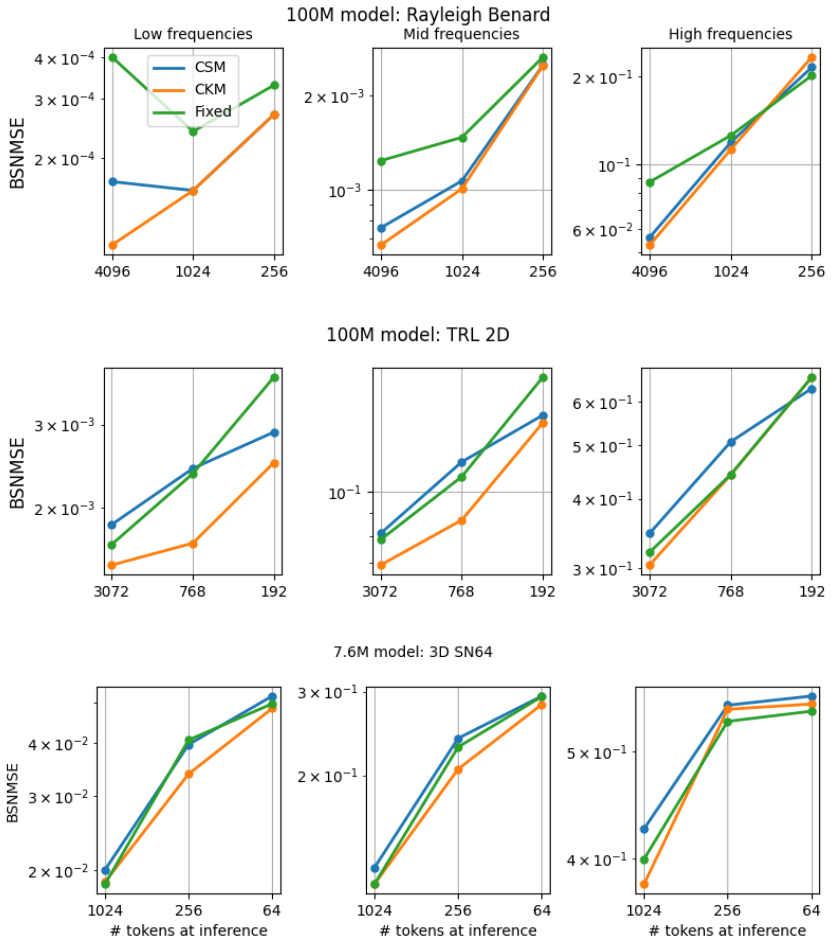

Figure 7: BSNMSE of the next-step prediction of the test set in the `rayleigh_benard`, `turbulent_radiative_layer_2D`and `supernova_explosion` datasets. The spectrum is divided into three frequency bins: low, mid and high being the smallest, intermediate and largest frequency bins whose boundaries are evenly distributed in log space.

G Shear flow rollout

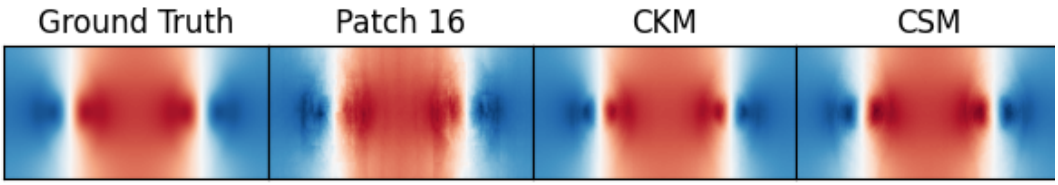

Figure 8: Step 64 rollout for the x-velocity, v_x field of the `shear_flow` dataset for the different models. Similar to other rollouts shown, the CKM and CSM-based models significantly decrease the patch artifacts compared to the fixed patch 16 model.