Neural Field Turing Machine: A Differentiable Spatial Computer

Akash Malhotra
LISN
Université Paris-Saclay
9 Rue Joliot Curie, Orsay
akash.malhotra@lisn.fr

Nacéra Seghouani
LISN
Université Paris-Saclay
9 Rue Joliot Curie, Orsay
nacera.seghouani@lisn.fr

Abstract

We introduce the Neural Field Turing Machine (NFTM), a theoretical framework for computation on spatial fields through local, iterative updates. NFTM defines a computational model where a stateless neural controller performs read-write operations on a neurally continuous spatial memory field via movable heads with bounded support regions. Unlike Neural Turing Machines, which access discrete memory locations globally, or Neural Cellular Automata, which operate on discrete grids with fixed update rules, NFTM formalizes computation on neurally continuous domains, fields with finite parameterization, continuous evaluation, and differentiable dynamics. The framework supports both sequential and parallel update schedules, with Turing completeness under bounded error established through Rule 110 emulation. We demonstrate NFTM's practical realizability through three minimal instantiations using standard neural architectures: cellular automata (Rule 110), physics-informed PDE solvers (heat equation), and image inpainting (CIFAR-10). These examples, operating on discrete grids as practical approximations, show that the framework captures real computational patterns across symbolic, physical, and perceptual domains. NFTM provides a compelling perspective on local iterative computation over spatial fields.

1 Introduction

Many problems in computer vision, robotics, and physics involve computation over spatially structured data that evolves iteratively. While various architectures address such problems, Transformers Vaswani et al. [2017] with global attention, Neural Cellular Automata Mordvintsev et al. [2020] with parallel local updates, PINNs Raissi et al. [2019] and neural operators Li et al. [2020] for physical dynamics, there is no unifying theoretical framework that formalizes computation on parametrized continuous spatial fields through local operations with learned update rules.

We introduce the *Neural Field Turing Machine (NFTM)* as such a framework. NFTM is not a specific architecture but a computational abstraction: it defines how stateless neural controllers can perform computation through sequential or parallel read-write operations on neurally continuous spatial fields, fields with finite parameterization, continuously queryable (i.e. queryable at arbitrary spatial coordinates), and differentiable dynamics.

The key distinguishing characteristics of NFTM are:

- *Stateless controller*: All computation state resides in the spatial field; the controller itself maintains no internal memory, distinguishing NFTM from recurrent architectures.

Machine Learning and the Physical Sciences Workshop, NeurIPS 2025.

- *Local operations with bounded support*: Computation proceeds through fixed-radius neighborhoods rather than global attention, enabling efficient scaling while maintaining expressivity.
- *Turing completeness*: We prove NFTM is Turing complete under bounded error through construction via Rule 110 emulation, establishing theoretical universality.

We demonstrate NFTM's practical realizability through three minimal instantiations using standard neural components (MLPs, CNNs): cellular automata (Rule 110), physics-informed PDE solvers (heat equation), and image inpainting (CIFAR-10). These examples operate on discrete grids as practical approximations of the continuous ideal, showing that the theoretical framework captures real computational patterns independent of specific implementation choices. Code and experiments are available at `https://github.com/akashjorss/NFTM`.

2 Related Work

Relevant prior work includes neural cellular automata, memory-augmented models, and neural differential equation solvers. Neural Cellular Automata Mordvintsev et al. [2020] demonstrate local differentiable updates but operate on discrete grids with fixed neighborhoods and lack explicit control flow. Neural Turing Machines Graves et al. [2014] provide differentiable memory via soft attention over discrete memory locations accessed globally, while NFTM defines memory as continuous spatial fields accessed locally. Neural ODE solvers Chen et al. [2018] and related iterative methods (including GNN-based message passing Liu et al. [2025]) typically use synchronous updates across all spatial locations; NFTM treats the field as a tape with movable read/write heads, enabling computation patterns through learned update rules. This Turing machine perspective formalizes computation through directed read/write operations on spatial fields. Unlike Transformers Vaswani et al. [2017], which perform global computations that become prohibitive as spatial fields grow large, NFTM's local operations with bounded support regions scale naturally with field size. NFTM also connects to classical analog computation models like GPAC Shannon [1941] and BSS machines Blum et al. [1989], but differs fundamentally through finite parameterization and gradient-based learning rather than idealized real-number arithmetic, situating it within the regime of *neural continuity*: computation that is simultaneously continuous in representation and practical (i.e. discrete) in implementation, such as the ones represented by SIREN Sitzmann et al. [2020], INRs Grattarola and Vandergheynst [2022] and NeRFs Mildenhall et al. [2021].

3 Neural Field Turing Machine: Definition and Universality

Neural Continuity. Before defining NFTM, we introduce the conceptual framework it instantiates. *Neural continuity* describes a computational regime between discrete and continuous systems, characterized by three defining properties:

- **(NC1) Finite parameterization**: a finite vector of parameters θ encodes both the field representation and update rules, ensuring practical implementability.
- **(NC2) Continuous Querying**: the state is defined as a function over a continuous domain (e.g., $f_t : \mathbb{R}^d \to \mathbb{R}^m$), though it may be discretized for computation.
- **(NC3) Differentiable dynamics**: field evolves in a differentiable way, such that a neural controller can learn the rules of its evolution through examples.

Neurally continuous systems differ from classical Turing machines (purely discrete) and BSS machines (idealized unit-cost real arithmetic) by combining theoretical expressivity with practical realizability through modern neural architectures. The NFTM formalizes one such system for spatial computation.

NFTM Definition. A *Neural Field Turing Machine (NFTM)* embeds computation into a spatial domain via movable read/write heads operating over local neighborhoods (Fig. 1). The system maintains a spatial field f_t and head positions h_t. Updates are computed over support regions $S(h_t) = \{x \in \mathcal{X} : \|x - h_t\| \leq r\}$, though arbitrary locality kernels are possible.

A controller C operates on local features to produce updates:

$$C(f_t[S(h_t)]) \mapsto (A_t(x, y), \Delta h_t), \tag{1}$$

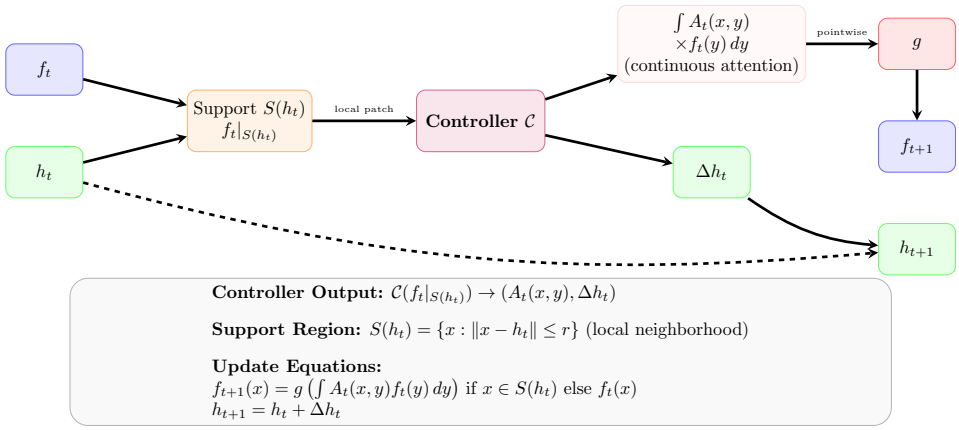

Figure 1: Neural Field Turing Machine (NFTM).

where $A_t(x, y)$ defines spatial attention and Δh_t updates head position. The dynamics are:

$$f_{t+1}(x) = g\left(\int A_t(x,y) f_t(y)\,dy\right) \text{ if } x \in S(h_t) \text{ else } f_t(x), \tag{2}$$

$$h_{t+1} = h_t + \Delta h_t, \tag{3}$$

with g a pointwise nonlinearity.

Proposition 1. *NFTMs are Turing complete under bounded error.*

Proof. (1) Discretization: Sample the field on a lattice of spacing Δ and interpret $f_t(n\Delta) \in \{0, 1\}$ as the state of cell n.

(2) Rule-110 Emulation: Configure the controller so that the support region $S(h_t)$ covers three adjacent sites (left, center, right). Given these inputs, C implements the Rule 110 update, known to be Turing complete Cook [2004].

(3) Sequential Sweep: Set $h_{t+1} = h_t + \Delta$ so the head scans the lattice and applies the local Rule 110 update at each site. The induced evolution reproduces Rule 110 dynamics on the discretized lattice up to $O(\Delta)$ quantization error.

Since the NFTM simulates Rule 110, it is Turing complete under bounded error. □

4 Methodology

We demonstrate three minimal instantiations of the NFTM framework using standard neural architectures. Each instantiation operates on discrete grids as practical approximations to neurally continuous fields, showing that the framework can be realized with existing components across symbolic, physical, and perceptual domains.

Cellular Automata We instantiate NFTM for cellular automata by using an MLP controller that learns rule truth tables from local neighborhoods. The controller reads a cell and its neighbors, computes the next state, and writes it back:

$$f_{t+1}(i) = \mathcal{C}(\text{read\_neighborhood}(f_t, i)) \tag{4}$$

For binary cellular automata (Rule 110), we use Straight-Through Estimator binarization: $\text{STE}(x) = \lfloor x + 0.5 \rfloor$ (forward), x (backward), enabling gradient-based learning of discrete update rules.

Physics-Informed PDE Solvers We instantiate NFTM for the 2D heat equation $\partial_t u(x, y, t) = \alpha(x, y)\nabla^2 u(x, y, t)$ by encoding the Laplacian operator as local neighborhoods accessed by the controller. The controller learns to apply finite-difference-like updates, with the spatial diffusion coefficient $\alpha(x, y)$ learned from data via standard MSE loss between predicted and ground-truth field evolution. Training proceeds in two phases: teacher forcing on single-step transitions, followed by rollout training to ensure stability under autoregression.

Image Inpainting We instantiate NFTM for CIFAR-10 inpainting using a CNN controller that predicts per-pixel corrections iteratively. Images are corrupted by masking 25-50% of pixels. The controller reads local patches around each pixel and predicts updates:

$$I_{t+1} = \text{clamp}(I_t + \beta \cdot g \cdot \Delta I_t) \tag{5}$$

where g is a learned gating function and β controls step size. Training uses MSE loss with curriculum learning over rollout depth (gradually increasing the number of iterative refinement steps).

5 Results

We evaluate three instantiations of the NFTM framework across symbolic, physical, and perceptual domains.

Cellular Automata. The MLP-based instantiation successfully learns Rule 110's local truth table and reproduces rollouts faithfully over 100 steps, extrapolating beyond the training horizon (Figure 2). The learned controller captures underlying update rules rather than memorizing trajectories.

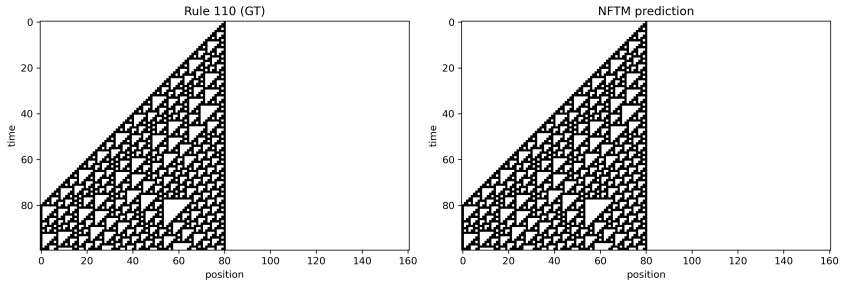

Figure 2: Rule 110 cellular automaton. Left: ground-truth evolution. Right: learned controller predictions showing successful learning of symbolic update rules.

Physics-Informed PDE Solvers. The heat equation instantiation learns spatially varying diffusion coefficients $\alpha(x, y)$ from data. For a central square region ($\alpha = 0.15$) embedded in background ($\alpha = 0.05$), the learned field matches ground truth with mean PSNR of 40.89 dB (Figure 3). Table 1 shows the NFTM-based solver achieves 38.48 dB with only 2,914 parameters, outperforming UNet (30.12 dB, 117K params) and ConvNet (25.72 dB, 27K params) with 40$\times$ fewer parameters.

Table 1: Heat equation with spatially-varying $\alpha(x, y)$. Local learned updates achieve strong accuracy with minimal parameters.

Model	Parameters	PSNR ↑	Inference (s)
Finite-difference	—	28.63	0.32
NFTM instantiation	2,914	**38.48**	3.62
ConvNet	27,265	25.72	2.19
UNet	116,753	30.12	10.79

Image Inpainting. The CNN-based instantiation for CIFAR-10 inpainting trained for 20 steps generalizes to 30 steps at test time. Table 2 shows competitive reconstruction (25.18 dB overall PSNR) with lowest LPIPS perceptual distance (0.0101 vs 0.0113/0.0126). PSNR improves monotonically from 15.2 dB to 24.5 dB (Figure 4), demonstrating test-time scaling where learned local rules continue improving predictions when rolled out further.

Across instantiations, learned local update rules compose into stable global dynamics that generalize beyond training horizons, with test-time scaling appearing consistently across domains.

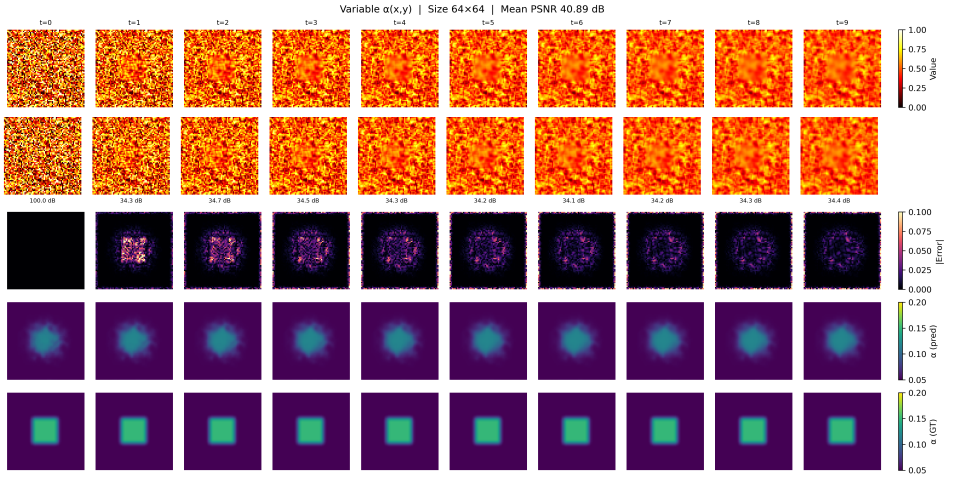

Figure 3: Learning spatially varying $\alpha(x, y)$. Top: ground truth and predictions. Middle: errors. Bottom: learned vs. ground-truth coefficients.

Table 2: CIFAR-10 inpainting (25–50% corruption). Iterative updates achieve competitive quality with lowest perceptual distance.

Method	PSNR (All) ↑	PSNR (Miss) ↑	LPIPS ↓	Params
UNet	**26.77**	**23.29**	0.0113	46,479
TV-L1	25.55	21.65	0.0126	—
NFTM instantiation	25.18	21.27	**0.0101**	46,375

6 Conclusion

We introduced the Neural Field Turing Machine (NFTM), a theoretical framework for computation on spatial fields through local iterative updates with stateless neural controllers operating on neurally continuous fields. We formalize neural continuity as a computational regime characterized by finite parameterization, continuous querying, and differentiable dynamics, bridging discrete and continuous computation. NFTM achieves Turing completeness under bounded error, demonstrated through Rule 110 emulation. Three instantiations using standard architectures (cellular automata, PDE solvers, image inpainting) show this framework captures real computational patterns across symbolic, physical, and perceptual domains, consistently exhibiting test-time scaling where predictions improve beyond training horizons. NFTM generalizes Neural Cellular Automata through neurally continuous fields and movable heads, differs from Neural Turing Machines through spatially bounded rather than global memory access, and extends Neural ODE methods by learning update rules that determine where and how computation proceeds at each step.

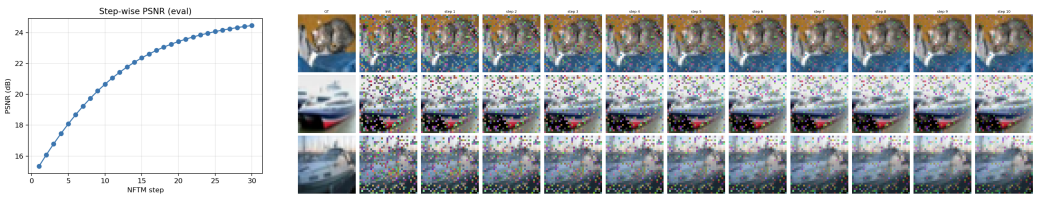

Figure 4: CIFAR-10 inpainting showing test-time scaling. Left: PSNR improvement beyond training horizon. Right: progressive reconstruction.

References

Lenore Blum, Michael Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. In Ciprian Foias and Georg Sell, editors, *Complexity of Computation*, pages 170–195. Springer, 1989.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.

Daniele Grattarola and Pierre Vandergheynst. Generalised implicit neural representations. *Advances in Neural Information Processing Systems*, 35:30446–30458, 2022.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

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.

Zewen Liu, Xiaoda Wang, Bohan Wang, Zijie Huang, Carl Yang, and Wei Jin. Graph odes and beyond: A comprehensive survey on integrating differential equations with graph neural networks. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 6118–6128, 2025.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 2020. doi: 10.23915/distill.00023. URL `https://distill.pub/2020/growing-ca`.

Maziar Raissi, Paris Perdikaris, and George Em 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.

Claude E. Shannon. Mathematical theory of the differential analyzer. *Journal of Mathematics and Physics*, 20:337–354, 1941.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

A Complexity Analysis of NFTM Instantiations

The NFTM framework does not prescribe specific computational complexity, as this depends on implementation choices. Here we analyze the three instantiations presented in this paper, all of which use synchronous updates on discrete grids.

For these grid-based instantiations with synchronous parallel updates, let N denote the number of spatial sites, T the rollout horizon, and P the cost of one controller forward pass. Each timestep involves:

1. Reading local neighborhoods (all sites in parallel)
2. Applying the controller to compute updates
3. Writing updates back to the field

Table 3: Asymptotic complexity of our instantiations. N = field size, T = rollout horizon, P = controller cost, r = patch radius.

Instantiation	Complexity	Notes
Cellular Automata	$\mathcal{O}(N \cdot T \cdot P)$	MLP controller, STE binarization
PDE Solvers	$\mathcal{O}(N \cdot T \cdot (P + k))$	5-point stencil
Image Inpainting	$\mathcal{O}(N \cdot T \cdot P)$	CNN controller
With Local Attention	$\mathcal{O}(N \cdot r^2 \cdot T)$	Patch radius r

For neighborhoods of fixed size k, the read/write cost is $\mathcal{O}(kN)$ and the controller adds $\mathcal{O}(NP)$, giving total time complexity

$$\mathcal{O}(N \cdot T \cdot (k + P)). \tag{6}$$

Memory usage is $\mathcal{O}(N)$ for forward simulation, or $\mathcal{O}(N \cdot T)$ if storing full trajectories for backpropagation through time. With truncated BPTT over window w, this reduces to $\mathcal{O}(N \cdot w)$.

For instantiations using local attention over patches of radius r, each site aggregates $\mathcal{O}(r^2)$ neighbors, giving complexity

$$\mathcal{O}(N \cdot r^2 \cdot T). \tag{7}$$

For small r (e.g., 1-3), this remains efficient and GPU-parallelizable.

Table 3 summarizes these instantiations. The linear scaling in N and T is comparable to standard CNNs and finite difference methods. These results demonstrate the practical viability of our specific implementations, though alternative instantiations of the NFTM framework (e.g., sequential updates, truly continuous representations) would have different complexity characteristics.

B Neural Continuity

We introduce neural continuity as a computational regime characterized by finite parameterization, continuous evaluation, and differentiable dynamics. This concept distinguishes systems like NFTM from both classical discrete computation (Turing machines) and idealized continuous computation (BSS machines, GPAC).

Definition. A system is *neurally continuous* if it satisfies three properties:

- **(NC1) Finite parameterization.** A finite vector of parameters θ encodes both the state representation and update rules. The system can be implemented on finite computational hardware.
- **(NC2) Continuous evaluation.** The state is defined as a function over a continuous domain (e.g., $f_t : \mathbb{R}^d \to \mathbb{R}^m$). While practical implementations may discretize for computation, the representation itself is continuous.
- **(NC3) Differentiable dynamics.** State transitions $f_{t+1} = \Phi_\theta(f_t)$ are differentiable with respect to both f_t and θ, enabling gradient-based learning.

Relation to Other Computational Models. Neurally continuous systems occupy a distinct position relative to classical models:

Discrete computation (Turing machines, cellular automata) operates on symbolic states with finite alphabets and no continuous structure. Neurally continuous systems extend this through continuous evaluation (NC2) while maintaining finite parameterization (NC1).

Real-RAM and BSS machines assume idealized unit-cost arithmetic over exact real numbers, requiring infinite precision. Neurally continuous systems avoid this idealization through finite parameterization and are implementable with standard floating-point arithmetic.

Analog computation (GPAC, differential analyzers) performs continuous computation through physical mechanisms but lacks differentiable dynamics and gradient-based learning (NC3). Neurally continuous systems add trainability through backpropagation.

NFTM as Neurally Continuous. NFTM satisfies all three properties: the spatial field f_t provides continuous evaluation (NC2), the controller $\mathcal{C}$ has finite parameters (NC1), and both field updates and

head movements are differentiable (NC3). The Turing completeness proof (Proposition 1) shows that neural continuity does not sacrifice computational universality, while the three instantiations demonstrate practical realizability across different domains.

Open Questions. Several theoretical questions remain about neural continuity: What are its closure properties under composition? What computational complexity classes does it characterize? How does it relate formally to neural ODEs and operator learning? Can neurally continuous systems simulate each other efficiently? A rigorous computational theory of neural continuity is left to future work.