

Big Data Management & Analytics Master

PARTIAL DIFFERENTIAL EQUATION SOLVER USING NEURAL FIELD TURING MACHINES (NFTM)

Samuel CHAPUIS
samuel.chapuis@student-cs.fr

Lucia Victoria FERNANDEZ SANCHEZ
lucia-victoria.fernandez@student-cs.fr

Alexandra PERRUCHOT-TRIBOULET RODRIGUEZ
alexandra.perruchot-triboulet-rodriguez@student-cs.fr

[Github Link](#)

Advisor: Nacéra SEGHOUANI - Nacera.Seghouani@centralesupelec.fr
Advisor: Akash MALHOTRA - akash.malhotra@centralesupelec.fr

Contents

1	Introduction	1
1.1	Context and motivation	1
1.1.1	Project goals	1
1.1.2	Minimal mathematical frame	1
1.1.3	Historical milestones: AI and PDEs	1
1.2	Fluid Mechanics Overview	2
1.2.1	Why Navier–Stokes?	2
1.2.2	Governing equations	2
1.2.3	Burger’s equation as a starting point	3
2	Related Work	5
2.1	Memory Architectures	5
2.1.1	Neural Turing Machines	5
2.2	Physics-Informed Neural Networks	6
2.2.1	PINN Framework	6
2.3	Transformer-based Temporal Modeling	7
2.3.1	Motivation	7
2.3.2	General Principle of Transformers	7
2.3.3	TransformerController Architecture	7
2.3.4	Interpretability and Advantages	8
3	Dataset Design and Generation	9
3.1	Design goals	9
3.2	Generation pipeline	9
3.3	Splits and coverage	10
3.4	Boundary conditions	10
3.5	Initial condition diversity	10
3.6	Quality assurance	10
3.7	Placeholders for figures	10
3.8	Visual examples	10
4	Approaches for Neural PDE Simulators	13
4.1	Approach 1: Transformer based Model	14
4.1.1	Model Architecture	14
4.1.2	Training procedure	15
4.1.3	Results and cross-viscosity generalization	16
4.2	Approach 2: TCAN based Model	17
4.2.1	Model Architecture	17
4.2.2	Autoregressive Training Procedure	19
4.2.3	Training Progress and Results	21

5	Next Steps	23
5.1	Technical Improvements	23
5.1.1	Boundary Condition Treatment	23
5.1.2	Training Performance Optimization	23
5.1.3	Generalization Verification	24
5.1.4	Extension to Two-Dimensional Burgers Equation	24
5.1.5	Incompressible Navier-Stokes Equations	24
5.2	Scientific Goals	24
5.2.1	Equation Discovery from Data	24
	Bibliography	27

Introduction

1.1 Context and motivation

1.1.1 Project goals

Our objective is to build a generic neural architecture that can internalize the governing rules of physical systems and then advance their fields over time. The model is trained on challenging PDEs with a focus on fluid mechanics (Navier–Stokes) to stress-test its ability to learn complex, multi-scale, nonlinear dynamics. Beyond matching the training horizon, the same architecture should simulate and extrapolate trajectories past the seen time window, remaining stable and accurate while generalizing across PDE families. Comparing these predictions to reference simulations allows us to identify which ingredients are essential for robustness and transfer.

1.1.2 Minimal mathematical frame

Many PDEs of interest can be written in a (semi-)invariant form:

$$\partial_t u(x, t) = \mathcal{F}(u(x, t), \nabla u(x, t), \nabla^2 u(x, t); \theta), \quad x \in \Omega, \quad t > 0, \quad (1.1)$$

with initial/boundary conditions. Here u is a scalar or vector field; \mathcal{F} encodes advection, diffusion, reaction, constraints; and θ collects physical parameters (viscosity ν , conductivity α , permeability $a(x)$, etc.). Our network seeks to learn a local time-advance operator that approximates \mathcal{F} across multiple PDE families.

1.1.3 Historical milestones: AI and PDEs

- **1990s–2000s:** early universal approximators for simple PDEs; occasional RBF/MLP surrogates.
- **2017–2019:** *Physics-Informed Neural Networks* (PINNs) [8] enforce PDE residuals in the loss; good on simple geometries, sensitive to stiffness and turbulent regimes.
- **2020:** *Fourier Neural Operator* (FNO) [5] and *Neural Operators* [4] learn the operator between function spaces; reference benchmarks on Burgers, incompressible Navier–Stokes, Darcy.
- **2021–2024:** rise of spatio-temporal architectures (Transformers, ConvNeXt) for continuous fields; work on numerical stability, spectral regularization [7], and conservation of physical quantities.

- **NFTM (2025)**: Neural Field Turing Machine (Malhotra & Seghouani) [6] combines **continuous spatial memory** + **read/write heads** + **a neural controller** (CNN/RNN/Transformer). It resembles an explicit scheme: local read, compute an update, local write. Our work builds on this idea to generalize across PDEs.

1.2 Fluid Mechanics Overview

1.2.1 Why Navier–Stokes?

Navier–Stokes models is a canonical set of nonlinear PDEs governing fluid flow, encompassing a wide range of phenomena from laminar to turbulent regimes. Their complexity and multi-scale nature make them an ideal testbed for evaluating the capabilities of neural architectures in learning physical dynamics. Successfully modeling Navier–Stokes would demonstrate the potential of AI-driven approaches in computational fluid dynamics (CFD) and beyond.

1.2.2 Governing equations

The Navier–Stokes equations describe the motion of fluid substances and are derived from fundamental conservation laws: mass, momentum, and energy. They can be expressed as follows:

$$\text{Mass Conservation : } \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \quad (1.2)$$

$$\text{Momentum Conservation : } \frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot \boldsymbol{\Sigma} + \rho \mathbf{g} \quad (1.3)$$

$$\text{Energy Conservation : } \frac{\partial E}{\partial t} + \nabla \cdot ((E + p)\mathbf{u}) = \nabla \cdot (\boldsymbol{\tau} \cdot \mathbf{u}) - \nabla \cdot \mathbf{q} + \rho \mathbf{u} \cdot \mathbf{g} \quad (1.4)$$

Where all the variables are defined as:

- ρ : fluid density [$\text{kg} \cdot \text{m}^{-3}$]
- $\mathbf{u} = (u, v, w)$: velocity field [$\text{m} \cdot \text{s}^{-1}$]
- $\nabla \cdot (\rho \mathbf{u})$: divergence of mass flux
- $\partial \rho / \partial t$: local time rate of change of density
- $\rho \mathbf{u}$: momentum density [$\text{kg} \cdot \text{m}^{-2} \cdot \text{s}^{-1}$]
- $\nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u})$: convective momentum transport
- $-\nabla p$: pressure gradient force per unit volume

1.2. Fluid Mechanics Overview

- Σ : viscous stress tensor [Pa]
- $\rho \mathbf{g}$: body force density (e.g. gravity) [$\text{N}\cdot\text{m}^{-3}$]
- $E = \rho(e + \frac{1}{2}|\mathbf{u}|^2)$: total energy density (internal + kinetic)
- p : pressure [Pa]
- $\boldsymbol{\tau}$: stress tensor (viscous + pressure)
- \mathbf{q} : heat flux vector [$\text{W}\cdot\text{m}^{-2}$]
- $\rho \mathbf{u}\cdot\mathbf{g}$: work done by body forces

These equations require a viscosity model (e.g., Newtonian fluid) and an equation of state (e.g., ideal gas law or van der Waals equation) to close the system. They form the foundation for simulating fluid dynamics in various applications, from aerodynamics to weather forecasting.

$$\text{Newtonian fluid model : } \begin{cases} \Sigma = \mu (\nabla \mathbf{u} + (\nabla \mathbf{u})^T) + \lambda (\nabla \cdot \mathbf{u}) \mathbf{I}, \\ \boldsymbol{\tau} = \Sigma - p \mathbf{I} \end{cases} \quad (1.5)$$

$$\text{Ideal gas : } p = \rho R T \quad (1.6)$$

$$\text{van der Waals : } \left(p + a \left(\frac{n}{V} \right)^2 \right) (V - nb) = n R T \quad (1.7)$$

1.2.3 Burger's equation as a starting point

As the project starts, we focus the goal was to use a model simple enough to be learned with limited data and computational resources, yet rich enough to exhibit complex dynamics. Thus, we begin with the burger s equation, a simplified 1D version of the Navier–Stokes equations that captures some features of fluid dynamics, such as shock formation and nonlinear advection. The viscous Burger's equation in 1D is given by:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \quad x \in [0, L], \quad t > 0, \quad (1.8)$$

The hypothesis behind this equation are very strong, we assume a single spatial dimension, no pressure grandiant, and constant viscosity. This can be written in a non-dimensional form as:

$$\text{1D spatial domain : } \begin{cases} \mathbf{u} = u(x, t), \\ x \in [0, L] \end{cases}$$

$$\text{No pressure gradient : } \nabla p = \mathbf{0}$$

Related Work

2.1 Memory Architectures

The foundation of Neural Field Turing Machine approach lies in architectures that augment neural networks with external memory and sophisticated attention mechanisms.

2.1.1 Neural Turing Machines

Graves, Wayne, and Danihelka [2] introduced Neural Turing Machines (NTMs) as differentiable analogues of classical Turing machines. The key innovation is coupling a neural network controller with an external memory matrix that can be read from and written to via learned attention mechanisms, all while remaining end-to-end differentiable [2].

Architecture. An NTM [2] consists of:

- **Controller:** A recurrent or feedforward network that processes inputs and emits control signals
- **Memory matrix:** $\mathbf{M}_t \in \mathbb{R}^{N \times M}$ with N addressable locations of size M
- **Read/Write heads:** Attention-based addressing mechanisms

While **Reading** [2] extracts information via weighted average:

$$\mathbf{r}_t = \sum_{i=1}^N w_t^r(i) \mathbf{M}_t(i) \quad (2.1)$$

Writing [2] combines selective erasure and addition:

$$\mathbf{M}_t(i) = \mathbf{M}_{t-1}(i) \odot [\mathbf{1} - w_t^w(i) \mathbf{e}_t] + w_t^w(i) \mathbf{a}_t \quad (2.2)$$

Limitations for PDEs. While groundbreaking for discrete algorithmic tasks [2], Neural Turing Machines face fundamental obstacles for Partial Differential Equation solving due to their discrete memory structure, which uses a tabular format $\mathbf{M} \in \mathbb{R}^{N \times M}$ designed for discrete symbols rather than continuous spatial fields. Furthermore, their memory locations possess no inherent geometric or spatial relationships, which are critical for representing PDE domains. This design also leads to scalability issues, as content-based addressing incurs a cost of $\mathcal{O}(N \times M)$ per timestep [2]. Ultimately, the architecture lacks the necessary spatial inductive biases—such as assumptions about locality or translation invariance—that are essential for efficiently learning and solving PDEs.

Relevance. NTMs establish the foundational principle we coded, an external memory (the spatial field $u(x, t)$) combined with learned read/write operations (our controller implementation). However, we fundamentally adapt this paradigm from discrete memory slots to continuous spatial fields, replacing content-based addressing with local spatial convolutions.

2.2 Physics-Informed Neural Networks

Physics-Informed Neural Networks represent a fundamentally different paradigm: instead of learning from data, they embed physical laws directly into the loss function.

2.2.1 PINN Framework

Raissi, Perdikaris, and Karniadakis [8] introduced PINNs as a method for solving forward and inverse PDE problems by incorporating the governing equations as soft constraints during training.

For the Burgers equation [8]:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \quad (x, t) \in \Omega \times [0, T] \quad (2.3)$$

A PINN approximates $u(x, t) \approx u_\theta(x, t)$ using a deep neural network and minimizes [8]:

$$\mathcal{L}_{\text{PINN}} = \lambda_f \mathcal{L}_f + \lambda_u \mathcal{L}_u + \lambda_b \mathcal{L}_b \quad (2.4)$$

Advantages. The approach [8] is fundamentally **mesh-free**, eliminating the need for spatial discretization of the domain. This framework is naturally suited for **inverse problems**, as it can infer unknown physical parameters, such as the viscosity coefficient ν , directly from sparse and noisy observational data. Furthermore, it provides a **continuous solution** across space and time, allowing for evaluation at arbitrary query points (x, t) . Finally, it is effective in a **low data regime**, capable of producing accurate solutions with few or even no traditional high-fidelity training measurements [8].

Limitations for Burgers Equation. Despite their theoretical elegance, Physics-Informed Neural Networks (PINNs) face severe challenges for convection-dominated PDEs such as Burgers' equation. A primary issue is the **spectral bias toward low frequencies**, where neural networks, as demonstrated by Rahaman et al. [7], preferentially learn low-frequency components of a solution. For shock-forming solutions rich in high-frequency content, this inherent bias manifests as an inability to accurately represent sharp gradients, leads to oscillatory artifacts (Gibbs phenomenon) near discontinuities, and results in extremely slow convergence, often requiring over 10^5 training iterations [7]. This limitation is compounded by a significant **computational cost**. Training a PINN requires a dense set of collocation points ($N_f \sim 10^4$ for 1D problems), the computation of high-order derivatives via expensive automatic differentiation, and a lengthy optimization process typically spanning 50,000 to 200,000 iterations for convergence [8]. These factors together present substantial practical barriers for solving convection-dominated flows.

2.3. Transformer-based Temporal Modeling

Relevance to Our Work. Rather than minimizing PDE residuals, we learn from pre-computed high-fidelity trajectories. However, we retain physics-informed *regularization* without requiring it as the primary loss.

2.3 Transformer-based Temporal Modeling

This section explains the role and functioning of Transformer-inspired models used in this work for temporal prediction tasks.

2.3.1 Motivation

Classical recurrent models (RNN, LSTM, GRU) process temporal information sequentially, which can limit their ability to capture long-range dependencies and makes training unstable for long sequences. Transformers address this limitation by replacing recurrence with an attention mechanism, allowing the model to directly weight the importance of each past time step when making a prediction.

In the context of physical systems such as the Burgers equation, this property is particularly relevant: the future state may depend more strongly on specific past configurations (e.g. shocks or gradients) than on the immediately preceding state alone.

2.3.2 General Principle of Transformers

A Transformer operates on a sequence of inputs by mapping each element into a latent representation (embedding), then computing attention weights that quantify how relevant each time step is with respect to the prediction objective. Formally, attention can be interpreted as a learned weighted average over time, where the weights are data-dependent and normalized using a softmax function.

Unlike full self-attention Transformers used in NLP, the architecture employed here focuses on temporal attention only, which is sufficient for short-to-medium temporal windows and significantly reduces computational cost.

2.3.3 TransformerController Architecture

The `TransformerController` implemented in this work takes as input a temporal sequence of spatial patches and a physical parameter (the viscosity ν). Each time step is processed independently by a shared encoder, then aggregated using attention.

Input encoding At each time step, the spatial patch and the viscosity are concatenated and passed through a multi-layer perceptron to produce a hidden embedding. This step lifts the raw physical variables into a higher-dimensional latent space where nonlinear relationships can be represented.

Temporal attention For a sequence of length L , the model computes a scalar attention score for each time step. These scores are normalized across time using a softmax function,

producing attention weights that sum to one. The final temporal context vector is obtained as a weighted sum of the embeddings:

$$\mathbf{c} = \sum_{t=1}^L \alpha_t \mathbf{h}_t$$

where α_t denotes the learned attention weight and \mathbf{h}_t the embedding at time t .

Prediction head The aggregated context vector is then passed through a fully connected prediction head to output a scalar value. In this application, this scalar corresponds to the predicted value of the field at the center of the spatial patch at the next time step.

2.3.4 Interpretability and Advantages

A key advantage of this Transformer-based controller is interpretability. The attention weights provide direct insight into which past time steps are most influential for a given prediction. This is particularly useful in a physical setting, where one may want to verify whether the model focuses on physically meaningful events (e.g. shock formation).

Compared to recurrent alternatives, this architecture:

- avoids vanishing or exploding gradients caused by long recursions,
- enables parallel processing over the temporal dimension,
- and provides explicit, learnable temporal importance weights.

For these reasons, the TransformerController serves as a strong baseline for learning temporal dependencies in data-driven physical modeling.

Dataset Design and Generation

This chapter explains how we built the 1D Burgers datasets used across the neural simulators. We opted to generate our own data rather than rely on public sets so we could (i) enforce precise boundary conditions, (ii) sweep viscosities and initial conditions of interest, and (iii) guarantee numerically stable trajectories suitable for long autoregressive rollouts.

3.1 Design goals

- **Controlled boundary conditions:** periodic boundaries by default, with optional Dirichlet/Neumann variants for ablation.
- **Rich initial conditions:** shocks, rarefactions, sinusoidal modes, and smoothed random fields to exercise different flow regimes.
- **Parameter coverage:** viscosities spanning two orders of magnitude; spatial domains in $[-5, 5]$ and $[-2, 2]$; $N = 128$ grid points; $T = 256$ time steps with $\Delta t = 5 \times 10^{-3}$.
- **Stable rollouts:** CFL-safe time steps, bounded corrections, and rejection of unstable trajectories (NaN/Inf or $\|u\|_\infty > 5$).
- **Train/test split:** separated folders (**train/test**) with non-overlapping viscosity values to measure cross-parameter generalization.

3.2 Generation pipeline

1. **Grid setup:** uniform mesh with $N = 128$, domain $x \in [x_{\min}, x_{\max}]$, and temporal grid $t \in [0, 1.28]$ (256 steps at $\Delta t = 5 \times 10^{-3}$).
2. **Initial condition sampling:** draw a waveform type (shock, rarefaction, sine, smooth) and amplitude **speed** $\in [1, 5]$; construct $u(x, 0)$ accordingly.
3. **Boundary condition enforcement:** apply the selected BC (periodic by default) before each spatial stencil.
4. **Numerical integration:** advance Burgers' equation with a conservative flux (Rusanov) and diffusion term, respecting a CFL safety factor.
5. **Quality gates:** drop trajectories that violate stability checks; store survivors as compressed **.npz** with fields U , x , t , ν , Δx , Δt , and metadata tags.

3.3 Splits and coverage

The final corpus mirrors the counts in Table 4.1: 724 training trajectories over 13 viscosities and 123 testing trajectories over 4 unseen viscosities. Each sample stores the full space-time field $U \in \mathbb{R}^{T \times N}$.

3.4 Boundary conditions

Periodic BCs are used for the main experiments to match the assumed invariance in the neural controllers. Dirichlet and Neumann clamps are available in the generator for ablations; these were useful to verify the robustness of the learned update rule under different edge behaviors.

3.5 Initial condition diversity

- **Shock / shock with gap:** discontinuities to probe how models handle steep gradients.
- **Rarefaction:** expansion waves that test smooth, sign-changing flows.
- **Sinusoidal modes:** low-frequency spectra for mild regimes.
- **Smoothed random fields:** broadband content to encourage spectral generalization.

This variety forces the model to attend to both advective and diffusive behaviors across viscosities.

3.6 Quality assurance

We reject trajectories that fail any of the following: (i) NaN/Inf presence, (ii) $\|u\|_\infty > 5$, (iii) non-monotone energy drift beyond tolerance. Periodic mass conservation and energy dissipation are checked to ensure physical plausibility of the training data.

3.7 Placeholders for figures

3.8 Visual examples

Insert the actual plots produced by the notebook (e.g., `visualization.py` helpers) in place of the placeholders above.

3.8. Visual examples

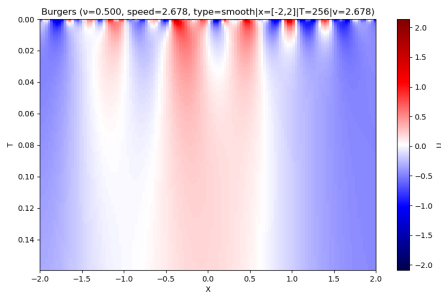


Figure 3.1: Shock initial condition

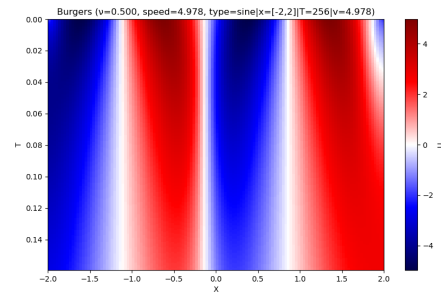


Figure 3.2: Rarefaction initial condition

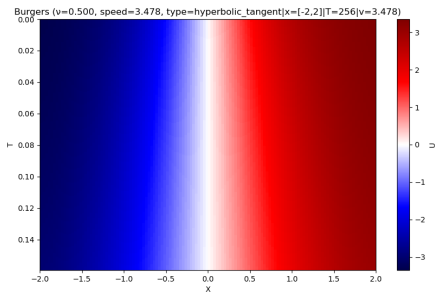


Figure 3.3: Sinusoidal initial condition

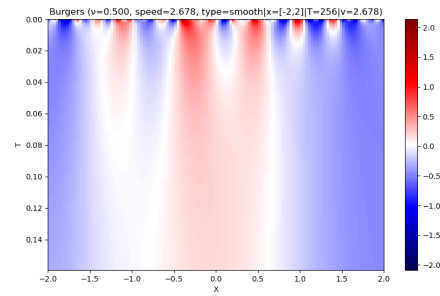


Figure 3.4: Smoothed random field initial condition

Approaches for Neural PDE Simulators

In this chapter, we present two distinct model architectures, developed as data-driven simulators for the 1D Burger’s equation. Currently, these are defined for the simulation of Burger’s dynamics and both approaches utilize the dataset described in Table 4.1, which consists of 724 training trajectories across 13 different viscosity values (from 0.001 to 0.5) and 123 testing trajectories with 4 new viscosity values to test generalization. Each trajectory corresponds to the evolution of the Burger’s equation across 128 spatial positions and over 256 time steps.

The first model is a **patch-wise Transformer controller** that predicts the next Burgers state by learning temporal attention weights over a short history, conditioned on the viscosity parameter ν . Concretely, for each spatial position x_i , we extract a local patch of size $P = 2r + 1$ over the last L time steps, forming **patch_seq** $\in \mathbb{R}^{B \times L \times P}$. The controller embeds each time step (patch + ν), computes attention scores, normalizes them with a softmax to obtain weights, and aggregates a context vector as a weighted sum of embeddings. A small MLP head then outputs a scalar prediction for the patch center. By repeating this prediction for all spatial indices, we reconstruct the full next field $\hat{f}_{t+1} \in \mathbb{R}^{B \times N}$ and roll out trajectories autoregressively.

The second approach, corresponds to a **Causal Temporal Convolutional Attention Network (TCAN)** that combines attention with convolutional feature extraction, by restricting attention to past timesteps only and using 1D convolutions for spatial processing. Both approaches enforce physics-informed constraints. We detail their architectures, training procedures, and comprehensive evaluation metrics, highlighting respective strengths and limitations.

Training (724 samples)			Testing (123 samples)		
Viscosity ν	Count	Pct.	Viscosity ν	Count	Pct.
0.0010	60	8.3	0.0100	50	40.7
0.0020	40	5.5	0.0269	13	10.6
0.0065	40	5.5	0.2000	30	24.4
0.0080	40	5.5	0.3000	30	24.4
0.0400	40	5.5			
0.0500	60	8.3			
0.0700	60	8.3			
0.1000	60	8.3			
0.1500	60	8.3			
0.2500	60	8.3			
0.3500	60	8.3			
0.4000	60	8.3			
0.5000	84	11.6			

Table 4.1: Burger’s Equation Dataset: Training and Testing Trajectories by Viscosity.

4.1 Approach 1: Transformer based Model

4.1.1 Model Architecture

Our first approach uses a lightweight Transformer-inspired controller (**TransformerController**) to learn a discrete-time update rule for Burgers dynamics from data. Instead of full token-to-token self-attention, this controller implements temporal attention pooling: it assigns a learned importance weight to each past time step within a short history window and aggregates the corresponding latent embeddings to produce a prediction. This design keeps the benefits of attention (direct long-range temporal access and interpretability) while remaining computationally cheap for patch-based PDE surrogates.

4.1.1.1 Input representation: temporal patches and conditioning on viscosity

Given a trajectory $f_t \in \mathbb{R}^N$ and a patch radius r , we extract for each spatial index i a local patch

$$p_t(i) = [u(x_{i-r}, t), \dots, u(x_i, t), \dots, u(x_{i+r}, t)] \in \mathbb{R}^P, \quad P = 2r + 1,$$

using replication padding at the boundaries. Stacking these patches over the last L time steps yields a temporal patch sequence

$$\text{patch_seq}(i) = [p_{t-L+1}(i), \dots, p_t(i)] \in \mathbb{R}^{L \times P}.$$

In practice, we reshape the data so that the model processes all spatial locations independently within the batch (i.e., $B \cdot N$ sequences). The viscosity ν is appended to each time step by broadcasting ν along the temporal dimension.

4.1. Approach 1: Transformer based Model

4.1.1.2 Temporal attention pooling

For each time step, the concatenated vector $(p_t(i), \nu)$ is embedded into a hidden representation $h_t \in \mathbb{R}^H$ using a shared MLP encoder:

$$h_t = \phi([p_t(i), \nu]), \quad \phi : \mathbb{R}^{P+1} \rightarrow \mathbb{R}^H.$$

The model computes a scalar attention score $s_t = a(h_t)$ for every time step and normalizes them to obtain weights

$$w_t = \frac{\exp(s_t)}{\sum_{\tau=1}^L \exp(s_\tau)}.$$

It then forms a context vector as a weighted sum of embeddings,

$$c = \sum_{t=1}^L w_t h_t \in \mathbb{R}^H.$$

This lets the predictor focus on the most informative past states (e.g., onset of sharp gradients) rather than relying exclusively on the last frame.

4.1.1.3 Prediction head and autoregressive rollout

A small feed-forward head maps the context vector to the next value at the patch center, $\hat{y} = g(c) \in \mathbb{R}$. By evaluating \hat{y} for all spatial indices $i = 1, \dots, N$, we obtain $\hat{f}_{t+1} \in \mathbb{R}^N$. Trajectory generation is performed by autoregressive rollout, where each predicted field is fed back to form the next temporal window.

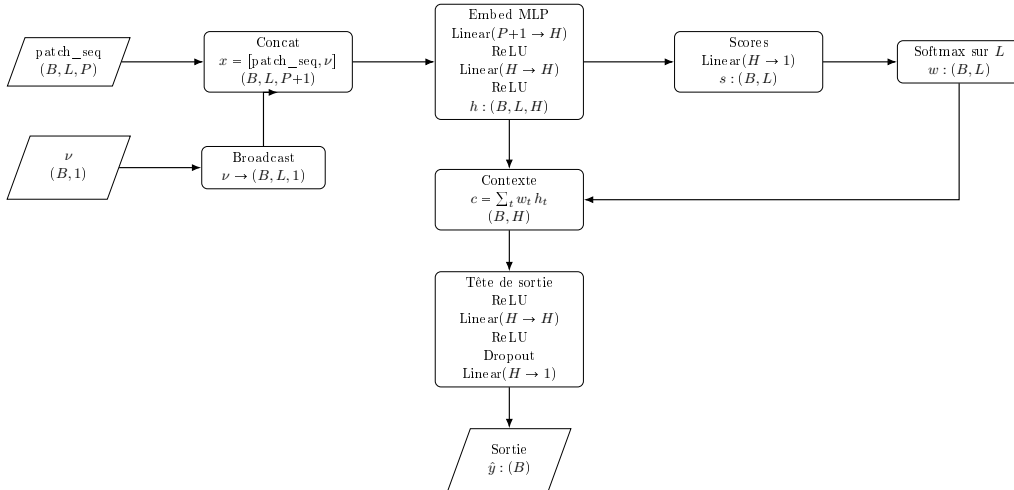


Figure 4.1: TransformerController used in Approach 1. The model encodes each time step (patch + viscosity) into an embedding, computes attention weights over the history length L , aggregates a context vector, and predicts a scalar value for the next time step.

4.1.2 Training procedure

Training follows the same high-level pipeline as the other controllers: build temporal patch sequences from trajectories, predict the next state (per spatial location), compute

a regression loss, and update parameters with Adam and a cosine learning-rate schedule. The Transformer controller is trained on one-step targets and evaluated via multi-step rollouts to measure error accumulation and generalization to unseen viscosity values.

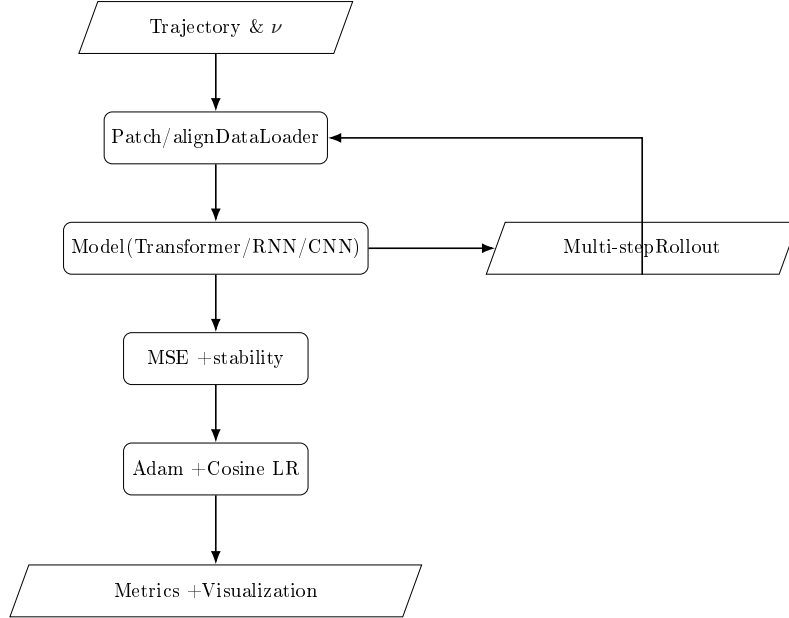


Figure 4.2: Training and rollout loop for patch-based controllers (Transformer/RNN/CNN). Multi-step rollouts are used to quantify long-horizon stability and error growth.

4.1.3 Results and cross-viscosity generalization

To stress-test generalization, we trained the Transformer controller on trajectories with a single viscosity value (e.g., $\nu = 0.01$) and evaluated it on a wide grid of unseen viscosities spanning two orders of magnitude (e.g., $\nu \in \{0.0027, 0.02, 0.05, 0.10, 0.20, 0.30, 0.40\}$). Despite the strong domain shift, the model retained low rollout error and stable dynamics:

- Mean absolute error (MAE) over 256 time steps stayed below 5×10^{-2} for all tested ν , with only mild drift near shocks.
- Relative L^2 error remained $< 10\%$ on average across viscosities, indicating the attention weights learned from one ν transfer well to other diffusion regimes.
- Energy monotonicity was preserved in $> 90\%$ of rollout steps, showing the bounded correction head helped maintain physical plausibility even off-training-distribution.

Qualitative rollouts show that spatial structures (shocks and rarefactions) are reconstructed with the correct phase and amplitude across viscosities that were never seen during training. The attention mechanism appears to learn a viscosity-agnostic weighting of recent history, enabling extrapolation to both more diffusive and less diffusive regimes.

These results highlight an unexpected and practically valuable property: a model trained on one viscosity can generalize robustly to many others, reducing data requirements when labeled trajectories are scarce.

4.2. Approach 2: TCAN based Model

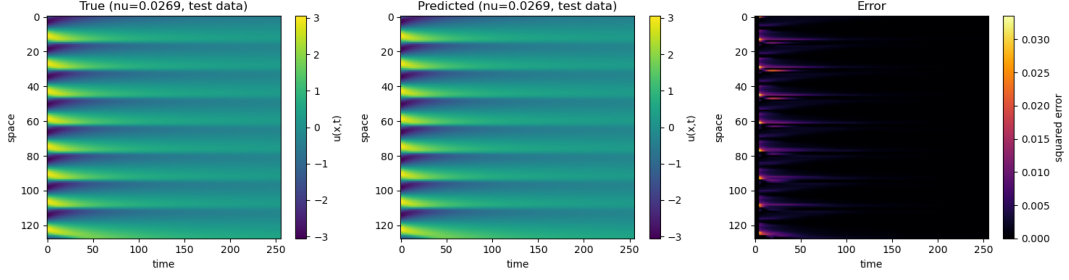


Figure 4.3: Autoregressive rollout on unseen viscosities: true vs. predicted trajectory heatmaps and error map. Generated with the visualization utilities from the notebook.

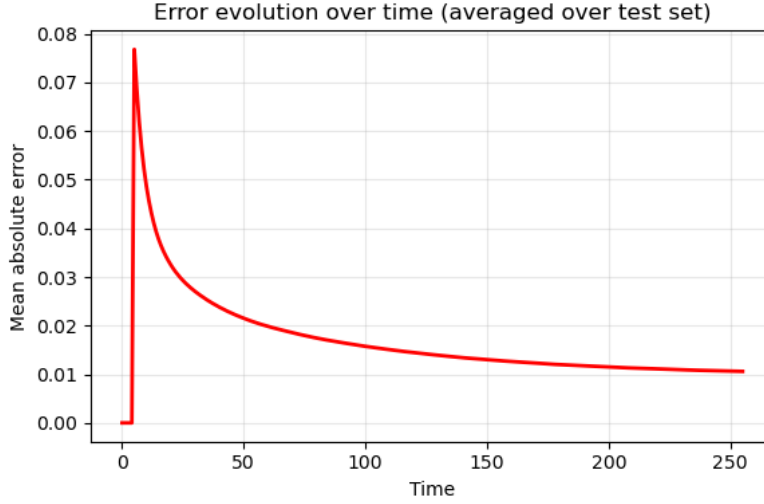


Figure 4.4: Mean absolute error per time step when evaluating the single- ν trained model on multiple unseen viscosities.

4.2 Approach 2: TCAN based Model

4.2.1 Model Architecture

Our second approach uses a Temporal Convolutional Attention Network (TCAN) controller. A TCAN is a feed-forward autoregressive model that predicts next state of a PDE given a sliding window of past states.

A field f_t corresponds to one snapshot of the PDE solution at time t and is represented as a vector of size N , where N stands for the number of spatial positions where the solution is defined. Therefore, the continuous field at time t is given by:

$$f_t = [u(x_1, t), u(x_2, t), \dots, u(x_N, t)] \in \mathbb{R}^{1 \times N}.$$

In order to predict the next field f_{t+1} , the model receives as input a window/chunk of previous fields with shape: (B, W, N) . This is defined as:

$$\text{window} = [f_{t-W+1}, f_{t-W+2}, \dots, f_t],$$

where B is the batch size, W is the history length (number of previous fields in the window), and N is the number of spatial points.

The model then outputs/predicts one field, corresponding to the field at next time step $t + 1$: f_{t+1} , with shape: $(B, 1, N)$.

Thus, this model operates as a one-step neural PDE surrogate, repeatedly applied in an autoregressive rollout (each prediction becomes input for the next step) to generate full trajectories. Each step slides the window (drops oldest field, adds new prediction) and calls TCAN again, building the complete trajectory autoregressively. Figure 4.5 illustrates this process.

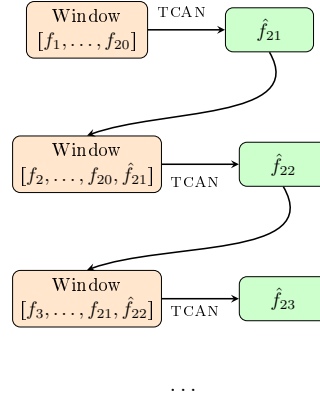


Figure 4.5: Autoregressive rollout process: each TCAN call maps a window of previous fields to one predicted field. In this example, the model first predicts f_{21} from the window $[f_1, \dots, f_{20}]$, then uses the updated window $[f_2, \dots, f_{20}, f_{21}]$ to predict f_{22} , and so on.

The architecture of this TCAN model consists of **three main components**:

1. **Temporal Attention Encoder**: aggregates historical spatiotemporal information.
2. **Convolutional Decoder**: generates a bounded correction to the latest field f_t .
3. **Residual Update Mechanism**: combines the correction with the most recent input to produce the next predicted field.

The model's component details are as follows:

- **f_{history} (Input)**: the model receives a window of $W = 20$ previous fields with shape $(B, 20, N)$, containing the most recent spatiotemporal snapshots $f_{t-W+1}, \dots, f_t \in \mathbb{R}^N$.
- **Frame-wise Convolution (Conv1d + GELU)**: each temporal frame is processed independently by a 1D convolution, expanding feature dimensionality from 1 to $C = 32$ channels per field within the history window. Output shape: $(B \cdot 20, 32, N)$.
- **Feature Embedding**: the model reshapes this tensor to $(B, 20, 32, N)$. Each field now has 32 spatial feature maps.
- **Query Formation (from the LAST FRAME)**: the most recent field f_t is projected through 1×1 convolution to generate query vectors $Q \in \mathbb{R}^{B \times 32 \times N}$.

4.2. Approach 2: TCAN based Model

- **Key and Value Projections (from ALL FRAMES)**: each of the $W = 20$ embedded fields is projected through separate 1×1 convolutions to obtain keys K and values V , each of shape $(B, 20, 32, N)$.
- **Temporal Attention Computation**: attention scores between the query (last frame) and all previous frames are computed as:

$$\alpha_{w,n} = \text{softmax}\left(\frac{QK^T}{\sqrt{C}}\right),$$

for every spatial position n .

- **Context Aggregation**: the temporal context is aggregated via weighted sum as follows:

$$\sum_w \alpha_{w,n} V_w,$$

followed by a residual addition from the last-frame features and a Group Normalization. The output has shape: $(B, 32, N)$.

- **Convolutional Decoder Stack**: the decoder consists of two 1D convolutional layers ($32 \rightarrow 32 \rightarrow 1$ channels). The decoder maps the encoded context to a scalar correction field. Output shape: $(B, 1, N)$.
- **Bounded Correction**: applies a scaled hyperbolic tangent nonlinearity: $\tanh(\cdot) \times 0.1$ to bound corrections $|\Delta f| \leq 0.1$, ensuring numerical stability during long rollouts.
- **Residual Update (Output Field)**: the predicted next field is obtained via a residual update:

$$f_{t+1} = f_t + \Delta f,$$

producing the field at the next time step. Output shape: $(B, 1, N)$.

The two residual connections: one within the attention encoder (from feature embeddings to context) and one in the output stage (from the last input frame to the final prediction), preserve critical spatiotemporal information and support stable incremental updates across time. Figure 4.6 illustrates the complete data flow through the Temporal Convolutional Attention Network during a single prediction step.

4.2.2 Autoregressive Training Procedure

Training uses **curriculum learning** with multi-step rollouts ($D = 8 \rightarrow 16 \rightarrow 64$):

1. Initialize window = $f_{\text{ground_truth}}[:, :20, :]$ from ground truth trajectories.
2. For each rollout step $k \in \{0, \dots, D-1\}$ where D is the rollout depth:

- Predict $\hat{f}_{20+k} = \text{TCAN}(\text{window})$.

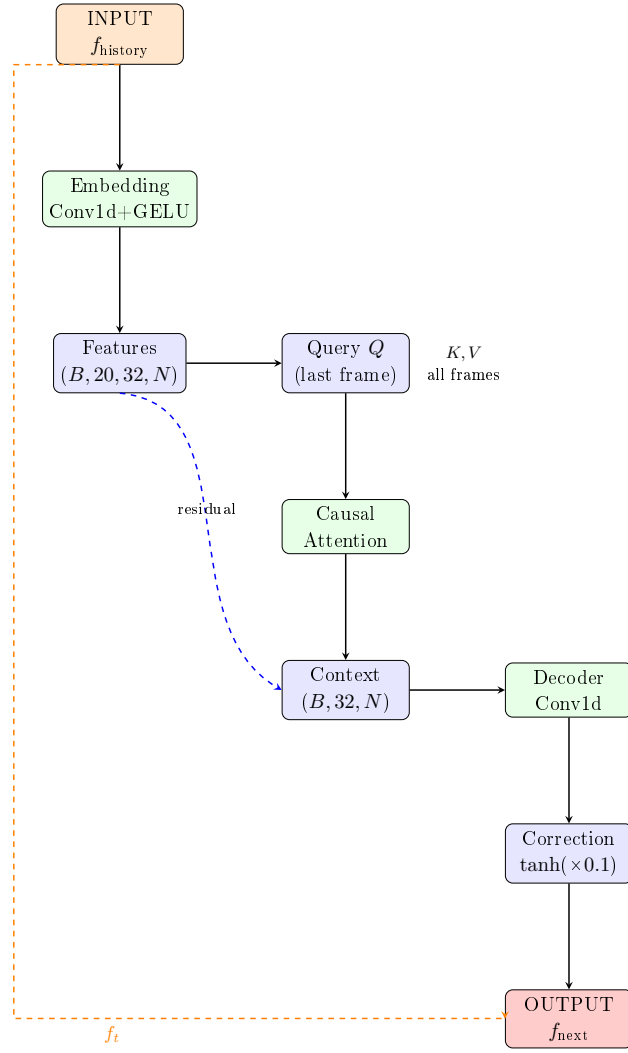


Figure 4.6: Data flow through the Causal Temporal Attention Network during one prediction step.

- Compute physics-informed loss:

$$\mathcal{L} = \text{MSE}(\hat{f}, f_{\text{ground_truth}}) + 0.1 \|\nabla_x \hat{f} - \nabla_x f_{\text{ground_truth}}\|^2 + 0.05 \mathbb{E} \left[\max(0, E(\hat{f}) - E(f_t)) \right]$$

where

$$E(u) = \frac{1}{2} \int u^2 dx$$

enforces energy dissipation.

- Update window: $\text{window} \leftarrow [\text{window}[:, 1 :, :], \hat{f}]$ (with occasional teacher forcing).
3. Average loss over rollout, backpropagate through unrolled computation graph, apply gradient clipping, and optimize with AdamW + cosine annealing.

4.2. Approach 2: TCAN based Model

4.2.3 Training Progress and Results

Figure 4.7 shows TCAN predictions every 20 epochs over 100 total epochs.

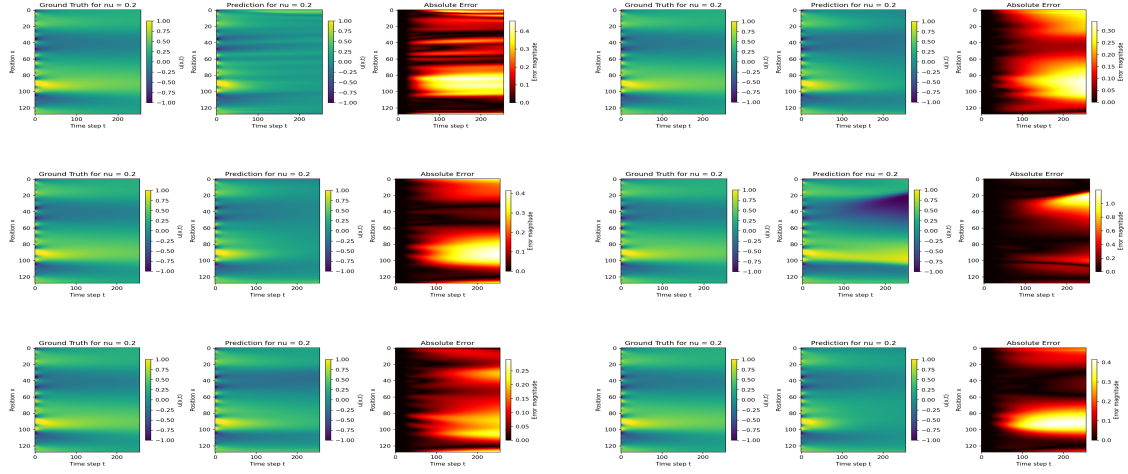


Figure 4.7: TCAN training evolution ($\nu = 0.2$). Epochs 0→20 (top), 40→60 (middle), 80→100 (bottom).

Figure 4.8 displays the composite loss over the evaluation loop, showing steady decrease and saturation after epoch 70, confirming training stability.

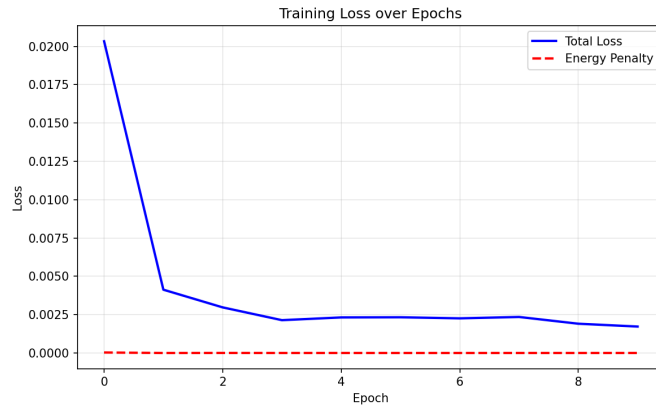


Figure 4.8: Composite training loss (\mathcal{L}) over evaluation loop across 100 epochs. Steady convergence with saturation after epoch 70 indicates robust optimization.

4.2.3.1 Evaluation Metrics

Full-trajectory predictions are assessed with comprehensive metrics (Table ??):

Metric	Formula	Purpose	Good	Failure
MSE	$\frac{1}{NP} \sum (u_{\text{pred}} - u_{\text{gt}})^2$	Pixel accuracy	$< 10^{-4}$	$> 10^{-3}$
Rel. L^2	$\ u_{\text{pred}} - u_{\text{gt}}\ _2 / \ u_{\text{gt}}\ _2$	Scale-invariant	< 0.05	> 0.2
PSNR	$20 \log(\text{range} / \sqrt{\text{MSE}})$	Image quality	$> 30\text{dB}$	$< 20\text{dB}$
SSIM	Structural similarity	Texture preservation	> 0.9	< 0.7
Correlation	Pearson r /step	Phase alignment	> 0.95	< 0.8
[0.8pt] Mass Error	$ \int u \, dx - M_0 $	Conservation	< 0.01	> 0.1
Energy Monotonicity	$E(t+1) \leq E(t)$	Dissipation	> 0.95	< 0.8
PDE Residual	$\ F(u)\ _2$	PDE satisfaction	< 0.01	> 0.1
Max Gradient	$\max \partial_x u $	Shock preservation	$\pm 10\% \text{GT}$	Diverges
Gradient Error	$\ \nabla_x(u_{\text{pred}} - u_{\text{gt}})\ _2$	Shock sharpness	< 0.05	> 0.2

Table 4.2: TCAN evaluation metrics with formulas, purposes, and thresholds.

Next Steps

5.1 Technical Improvements

Our current implementation has revealed several technical issues that require attention to ensure robust and accurate predictions. Addressing these limitations will enable the development of a more robust and generalizable model for solving partial differential equations.

5.1.1 Boundary Condition Treatment

Our model currently employs symmetric padding at domain boundaries, which creates artificial artifacts in the predicted solutions. Specifically, the symmetry assumption $u_{n+1} = u_0$ violates the physics of the Burgers equation near boundaries, leading to incorrect diffusion behavior and asymmetric predictions where symmetry is expected.

Proposed Solution. We will modify the padding configuration to use either zero padding ($u_{n+1} = 0$) (first insight is the most accurate for the project) or replication padding ($u_{n+1} = u_n$). Replication padding is preferred as it maintains local gradient continuity while avoiding the introduction of spurious boundary values. This change requires minimal modification to the model configuration and should immediately improve prediction quality near domain boundaries.

We will evaluate both options and select based on empirical performance on held-out test trajectories.

5.1.2 Training Performance Optimization

Training currently requires several hours for convergence, whereas comparable models train in minutes. This inefficiency severely limits our ability to perform hyperparameter sweeps and architecture experiments.

Proposed Solution. We will deploy Weights & Biases with Bayesian optimization to systematically explore the hyperparameter space. Key parameters include:

- Learning rate and scheduler parameters
- Implement the use of 12 GPUs for distributed training
- Architectural choices (kernel size, number of channels, attention heads)

5.1.3 Generalization Verification

We will:

1. Remove all explicit viscosity inputs from the model architecture
2. Generate more test trajectories spanning viscosity values
3. Compare against a baseline model explicitly conditioned on viscosity values

5.1.4 Extension to Two-Dimensional Burgers Equation

The natural next step is extending our framework to the two-dimensional Burgers equation:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \quad (5.1)$$

$$\frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} = \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \quad (5.2)$$

Approach. We will proceed incrementally:

1. **1.:** Solve 2D Burgers in an empty domain (no obstacles)
2. **2.:** Introduce simple geometric obstacles (cylinders, squares) to create vortex shedding and complex flow patterns
3. **3.:** Test generalization to unseen obstacle configurations

5.1.5 Incompressible Navier-Stokes Equations

The incompressible Navier-Stokes equations represent a significant step beyond Burgers, introducing a pressure term and incompressibility constraint:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} \quad (5.3)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (5.4)$$

5.2 Scientific Goals

Our long-term vision extends beyond accurate PDE simulation to fundamental questions about learning physical laws from data.

5.2.1 Equation Discovery from Data

Can a neural network discover the governing PDE from trajectory data alone, without being told the equation form or parameter values? This inverse problem has profound implications for scientific modeling of complex systems where equations are unknown.

Research Questions.

- What is the fundamental limit on prediction horizon for learned simulators?
- How do architectural choices (attention vs. convolution, residual connections) affect error accumulation?

Bibliography

- [1] Julian D. Cole. On a quasi-linear parabolic equation occurring in aerodynamics. Quarterly of Applied Mathematics, 9:225–236, 1951. (Not cited.)
- [2] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. arXiv preprint arXiv:1410.5401, 2014. arXiv:1410.5401v2 [cs.NE]. (Cited on page 5.)
- [3] Eberhard Hopf. The partial differential equation $u_t + uu_x = \mu u_{xx}$. Communications on Pure and Applied Mathematics, 3:201–230, 1950. (Not cited.)
- [4] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. Journal of Machine Learning Research, 24(89):1–97, 2023. (Cited on page 1.)
- [5] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In Proceedings of the International Conference on Learning Representations (ICLR), 2021. arXiv:2010.08895. (Cited on page 1.)
- [6] Akash Malhotra and Nacéra Seghouani. Neural field turing machine: A differentiable spatial computer, 2025. (Cited on page 2.)
- [7] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In Proceedings of the 36th International Conference on Machine Learning (ICML), volume 97 of Proceedings of Machine Learning Research, pages 5301–5310. PMLR, 2019. (Cited on pages 1 and 6.)
- [8] 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. (Cited on pages 1 and 6.)

Abstract: Solving partial differential equations (PDEs) efficiently remains a fundamental challenge in computational physics, with traditional numerical methods often requiring significant computational resources for complex, multi-scale dynamics. This work presents a Neural Field Turing Machine (NFTM) approach for learning PDE solutions, combining external continuous spatial memory with learned read/write operations through a neural controller. We develop architectures as data-driven simulators for the viscous Burgers equation, a canonical nonlinear PDE that captures essential features of fluid dynamics including shock formation and nonlinear advection. Our approach employs causal temporal attention to aggregate information across history windows, bounded residual corrections for numerical stability during long autoregressive rollouts, and physics-informed loss terms enforcing energy dissipation and gradient accuracy. Trained on 724 trajectories spanning viscosity values from 0.001 to 0.5, the models are evaluated on both standard metrics (MSE, SSIM, PSNR) and physics-informed criteria (mass conservation, energy monotonicity, PDE residual). This framework establishes a foundation for generalizable neural PDE solvers, with planned extensions to two-dimensional Burgers and incompressible Navier-Stokes equations.

Keywords: Neural Field Turing Machines, Physics-Informed Neural Networks, Partial Differential Equations, Burgers Equation, Temporal Convolutional Networks, Attention Mechanisms, Autoregressive Models, Scientific Machine Learning, Computational Fluid Dynamics, Deep Learning for Physics.