CentraleSupélec

université PARIS-SACLAY

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

Chapter 1

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:

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

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:

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

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

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

Where all the variables are defined as:

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

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

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

$$\text{Ideal gas}: \quad p = \rho RT \tag{1.6}$$

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

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:

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

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:

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

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

Chapter 2

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:

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

Writing [2] combines selective erasure and addition:

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

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]:

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

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

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

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.

Relevance to Our Work. Rather than minimizing PDE residuals, we learn from precomputed 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.

Chapter 3

Approaches for Neural PDE Simulators

In this chapter, we present two distinct Neural Field Turing Machine (NFTM) 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 3.1, which consits 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.

3.1 Approach 1: Transformer based Model

3.1.1 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.

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 3.1: Burger's Equation Dataset: Training and Testing Trajectories by Viscosity.

3.1.1.1 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

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

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

$$\texttt{patch\_seq}(i) = \big[p_{t-L+1}(i), \ldots, p_t(i)\big] \in \mathbb{R}^{L \times P}.$$

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.

3.1.1.2 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:

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

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.

3.1.1.3 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 $i = 1, \ldots, N$, 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.

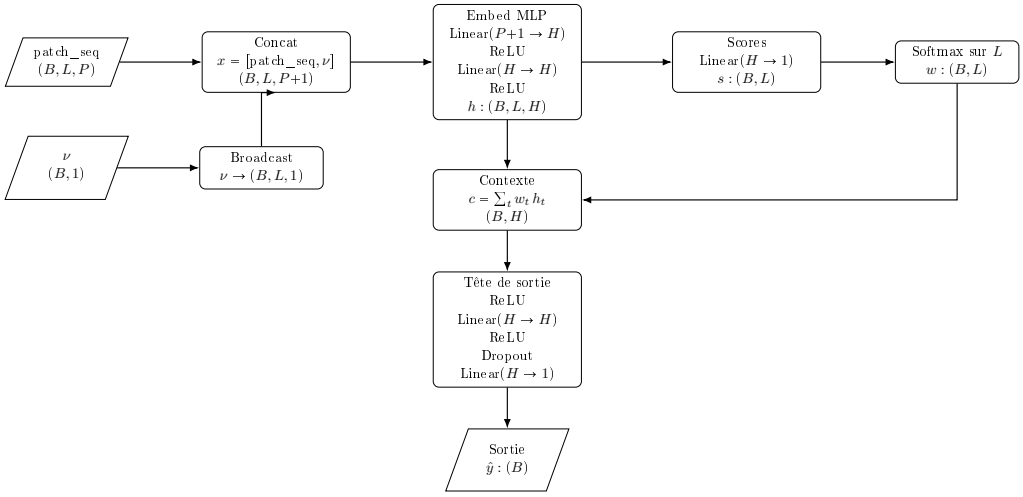

Figure 3.1: TransformerController used in Approach 1. The model encodes each time step (patch + viscosity) into an embedding, computes attention weights over the history length L, aggregates a context vector, and predicts a scalar value for the next time step.

3.1.2 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.

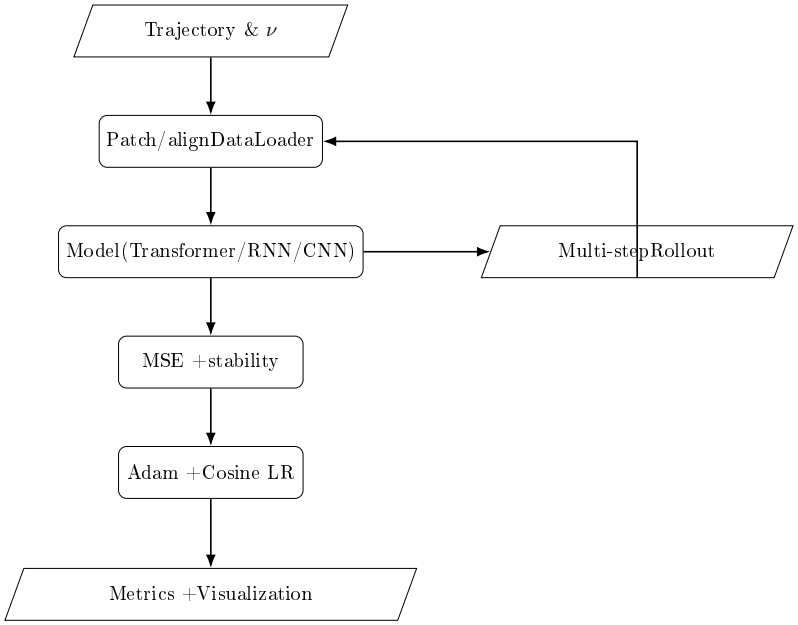

Figure 3.2: Training and rollout loop for patch-based controllers (Transformer/RNN/CNN). Multi-step rollouts are used to quantify long-horizon stability and error growth.

3.1.3 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.

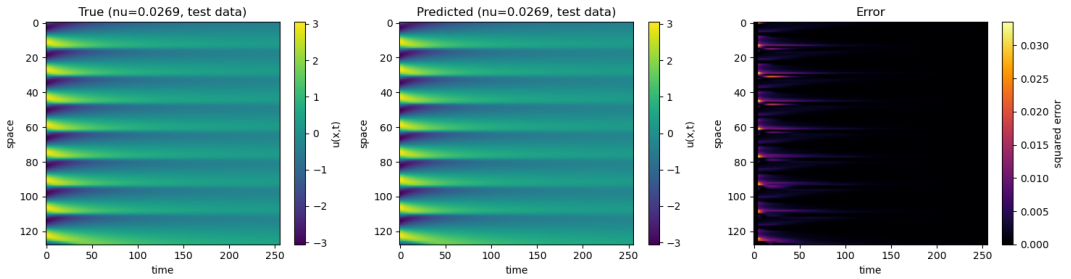

Figure 3.3: Autoregressive rollout on unseen viscosities: true vs. predicted trajectory heatmaps and error map. Generated with the visualization utilities from the notebook.

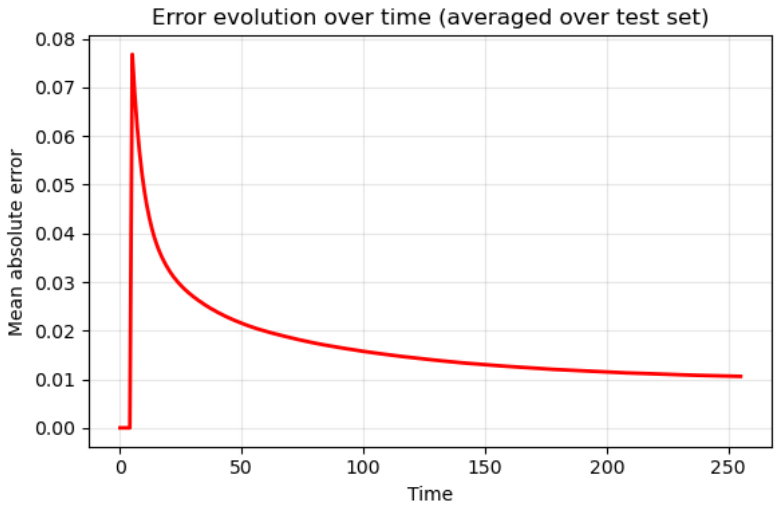

Figure 3.4: Mean absolute error per time step when evaluating the single-ν trained model on multiple unseen viscosities.

3.2 Approach 2: TCAN based Model

3.2.1 Model Architecture

The Temporal Convolutional Attention Network (TCAN) is a feed-forward autoregressive model that predicts next state of a PDE given a sliding window of past states. The TCAN serves as the *controller* in the Neural Field Turing Machine model framework, where it learns a discrete-time update rule for continuous spatiotemporal fields.
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:

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

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:

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

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 3.5 illustrates this process.

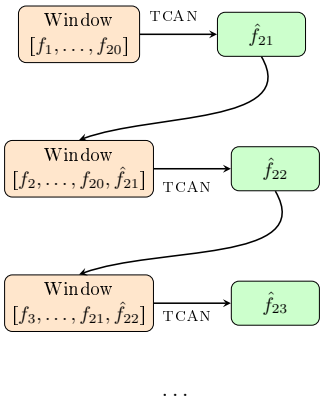

Figure 3.5: Autoregressive rollout process: each TCAN call maps a window of previous fields to one predicted field. In this example, the model first predicts f_{21} from the window $[f_1, \ldots, f_{20}]$, then uses the updated window $[f_2, \ldots, f_{20}, f_{21}]$ to predict f_{22}, and so on.

The architecture of this TCAN model consists of **three main components**:
1. A temporal attention encoder that aggregates information across the history window.
2. A convolutional decoder that produces a bounded correction.
3. A residual update that adds the correction to the last frame.

Figure 3.6 illustrates the complete data flow through the Causal Temporal Convolutional Attention Network during a single prediction step. Each node performs a specific transformation:

- $f_{\textbf{history}}$ **(Input)**: Window of $W = 20$ previous fields of shape (B, W, N), containing the most recent spatiotemporal snapshots $f_{t-W+1}, \ldots, f_t \in \mathbb{R}^N$.
- **Conv1d + GELU (frame-wise)**: Applies 1D convolution (kernel size 3) to each of the W fields individually, expanding from 1 to $C = 32$ feature channels per field. Output shape: $(B \cdot W, 32, N)$.
- **Features** (B, W, C, N): Reshaped embedded representation exposing the temporal dimension, with each of the W fields now containing 32 spatial feature maps.
- **Last frame → Query** Q: Extracts features from the most recent field f_t and applies 1×1 convolution to generate query vectors $Q \in \mathbb{R}^{B \times 32 \times N}$.

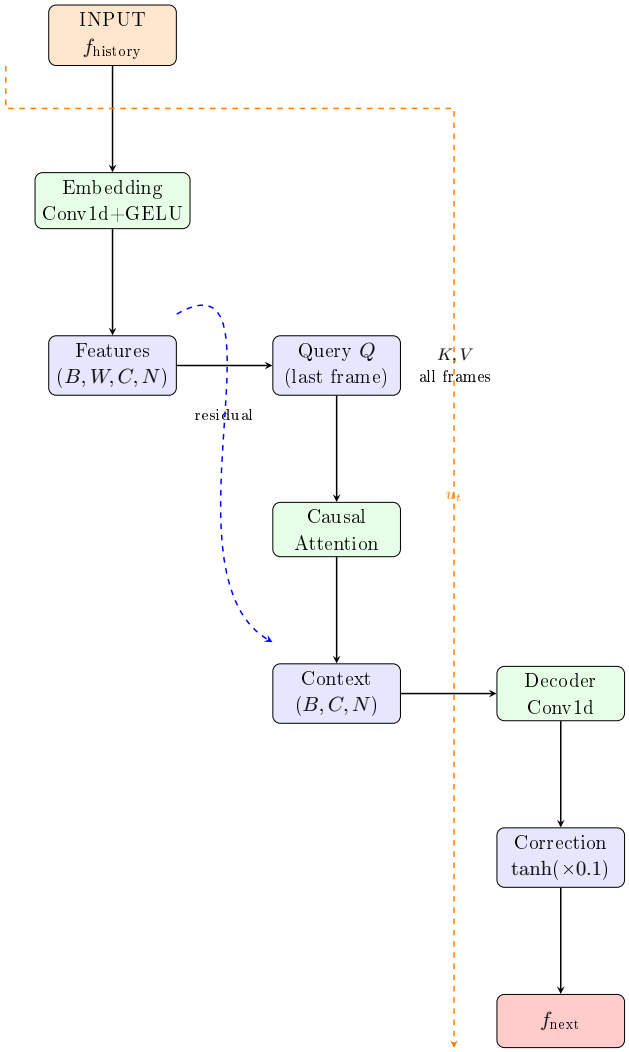

Figure 3.6: Data flow through the Causal Temporal Attention Network during one prediction step.

- **All frames → Keys K, Values V**: Projects all W embedded fields through separate 1×1 convolutions to produce keys K and values V, both of shape $(B, W, 32, N)$.
- **Attention** $\text{softmax}\left(\frac{QK^T}{\sqrt{C}}\right)$: Computes causal attention scores between the query (last frame) and all previous frames, producing temporal attention weights $\alpha_{w,p} = \text{softmax}_w(QK^T/\sqrt{32})$ for each spatial position p.
- **Context** (B, C, N): Aggregates temporal context via weighted sum $\sum_w \alpha_{w,p} V_w$, adds residual connection from last-frame features, and applies GroupNorm. Shape: $(B, 32, N)$.
- **Decoder Conv1d stack**: Two-layer convolutional decoder (32→32→1 channels, kernel size 5) that maps rich temporal-spatial context to a scalar correction field. Final layer zero-initialized for training stability.

- **Correction** $\tanh(\cdot) \times 0.1$: Applies hyperbolic tangent nonlinearity scaled by 0.1 to bound corrections $|\Delta u| \leqslant 0.1$, ensuring numerical stability during long rollouts.

- u_{next} **(Output)**: Residual update combining the last input field $u_{\text{history}}[:, -1, :]$ with the predicted correction: $f_{t+1} = f_t + \Delta u$. Shape: $(B, 1, N)$.

The two residual connections—from Features to Context (within attention encoder) and from input last frame to output—preserve critical spatiotemporal information while enabling stable incremental updates.

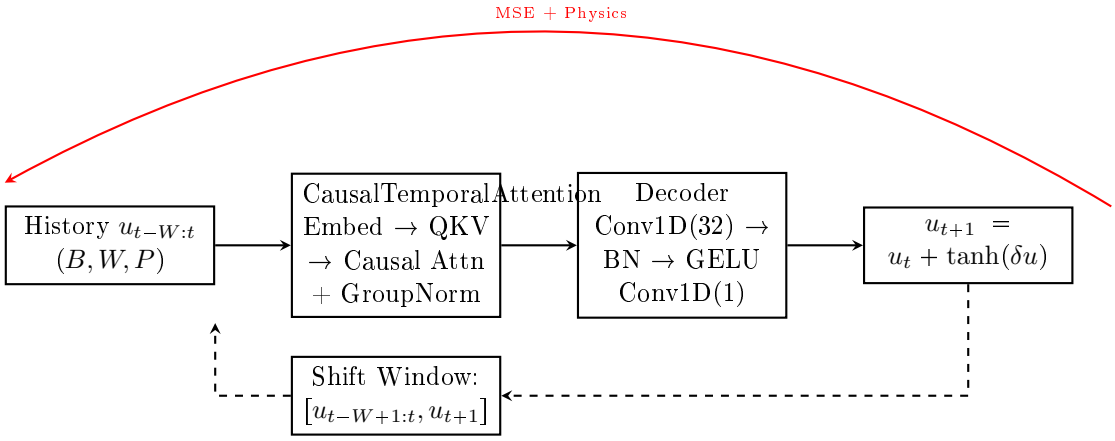

Figure 3.7: NFTM Architecture: Causal autoregressive prediction with physics-informed training.

Figure 3.7 summarizes the overall architecture of the NFTM model using the TCAN as controller. The model processes a sliding window of past fields to predict the next field, employing causal attention and convolutional decoding. The autoregressive rollout is indicated by dashed arrows, while the red feedback loop represents the physics-informed loss used during training.

3.2.2 Temporal Attention Encoder

The `CausalTemporalAttention` module processes the history window through the following steps:

1. **Frame-wise embedding**: Reshape u_{history} to $(B \cdot W, 1, P)$ and apply a 1D convolution with kernel size 3 followed by GELU activation, producing features of shape $(B \cdot W, C, P)$ where $C = 32$ is the embedding dimension.

2. **Temporal restructuring**: Reshape features back to (B, W, C, P) to expose the temporal dimension.

3. **Query computation**: Extract features of the last frame features$[:, -1, :, :] \in \mathbb{R}^{B \times C \times P}$ and apply a 1×1 convolution to obtain queries $Q \in \mathbb{R}^{B \times C \times P}$.

4. **Key and value projection**: Apply 1×1 convolutions to all W frames (flattened to $(B \cdot W, C, P)$ then reshaped) to obtain keys K and values V, both of shape (B, W, C, P).

5. **Causal attention scores**: Compute scaled dot-product attention over the temporal dimension:

$$\text{scores}_{b,w,p} = \frac{\langle Q_b, K_{b,w} \rangle}{\sqrt{C}}, \quad \alpha_{b,w,p} = \text{softmax}_w(\text{scores}_{b,w,p})$$

6. **Context aggregation**: Compute the attended context:

$$\text{context}_b = \sum_{w=1}^{W} \alpha_{b,w} \cdot V_{b,w} \in \mathbb{R}^{B \times C \times P}$$

7. **Residual projection**: Apply a 1×1 convolution to the context, add the residual connection to the last frame features, and normalize with GroupNorm.

This mechanism allows the last frame to dynamically query relevant past frames at each spatial location, producing spatially-aware temporal context features.

3.2.3 Decoder and Residual Correction

The `ImprovedBurgersNet` decoder processes the attention output context $\in \mathbb{R}^{B \times C \times P}$ as follows:

1. **Convolutional decoder stack**:

   - Conv1d: $C \to 32$ channels, kernel size 5, BatchNorm1d, GELU.
   - Conv1d: $32 \to 1$ channel, kernel size 5, producing raw_correction $\in \mathbb{R}^{B \times 1 \times P}$.

   The final layer is zero-initialized to ensure near-zero corrections during early training.

2. **Bounded correction**: Apply hyperbolic tangent clipping scaled by corr_clip $= 0.1$:

$$\Delta u = \tanh(\text{raw\_correction}) \cdot 0.1$$

3. **Residual update**: Add the correction to the last input frame:

$$u_{\text{next}} = u_{\text{history}}[:, -1:, :] + \Delta u$$

This design enforces incremental updates, preventing instability during long autoregressive rollouts.

3.2.4 Autoregressive Training Procedure

Training employs curriculum learning with multi-step rollouts:

1. Initialize current_window = $u_{\text{gt}}[:, :W, :]$ from ground truth trajectories.
2. For each rollout step $k \in \{0, \ldots, D-1\}$ where D is the rollout depth (curriculum: 8→16→64):
   - Predict $\hat{u}_{W+k} = \text{TCAN}(\text{current\_window})$.
   - Compute composite loss:

     $$\mathcal{L} = \text{MSE}(\hat{u}, u_{\text{gt}}) + 0.1 \cdot \|\nabla_x \hat{u} - \nabla_x u_{\text{gt}}\|_2^2 + 0.05 \cdot \mathbb{E}[\max(0, E(\hat{u}) - E(u_t))]$$

     where $E(u) = \frac{1}{2}\int u^2 \, dx$ enforces energy dissipation.
   - Update window: current_window ← [current_window$[:, 1:, :], \hat{u}]$ (with occasional teacher forcing).
3. Average loss over rollout, backpropagate through unrolled computation graph, apply gradient clipping, and optimize with AdamW + cosine annealing.

3.2.5 Evaluation Metrics

Full-trajectory predictions are evaluated using:

- Standard: MSE, relative L^2, PSNR, SSIM, temporal correlation.
- Physics-informed: mass conservation error, energy monotonicity fraction, mean PDE residual, spectral error, max gradient error.

This comprehensive evaluation ensures both accuracy and physical fidelity across long rollouts.

Metric		**Purpose**	**Good**	**Failure**
MSE	$\frac{1}{NP}\sum(u_{\text{pred}} - u_{\text{gt}})^2$	Pixel accuracy	$< 10^{-4}$	$> 10^{-3}$
Rel. L2	$\|u_{\text{pred}} - u_{\text{gt}}\|_2 / \|u_{\text{gt}}\|_2$	Scale-invariant	< 0.05	> 0.2
PSNR	$20 \log(\text{range}/\sqrt{\text{MSE}})$	Image quality	> 30dB	< 20dB
SSIM	Struct. similarity	Structure	> 0.9	< 0.7
Corr.	Pearson r/step	Phase alignment	> 0.95	< 0.8
[0.8pt] Mass Err.	$\lvert\int u \, dx - M_0\rvert$	Conservation	< 0.01	> 0.1
Energy Mono.	$E(t+1) \leqslant E(t)$	Dissipation	> 0.95	< 0.8
PDE Res.	PDE residual	PDE satisfaction	< 0.01	> 0.1
Max Grad.	$\max\lvert\partial_x u\rvert$	Shock preserv.	$\pm 10\%$GT	Diverges
Grad. Err.	Grad. difference	Shock sharpness	< 0.05	> 0.2

Table 3.2: Evaluation Metrics for NFTM Performance Assessment

Table 3.2 summarizes the evaluation metrics used to assess NFTM performance, detailing formulas, purposes, and thresholds for good vs. failure cases. This suite ensures comprehensive assessment of both numerical accuracy and physical fidelity.

Chapter 4

Next Steps

4.1 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.

4.1.1 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.

4.1.2 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)

4.1.3 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

4.1.4 Extension to Two-Dimensional Burgers Equation

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

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{4.1}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{4.2}$$

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

4.1.5 Incompressible Navier-Stokes Equations

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

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla p + \nu\nabla^2\mathbf{u} \tag{4.3}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{4.4}$$

4.2 Scientific Goals

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

4.2.1 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.