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 [kg·m^{-3}]
- $\mathbf{u} = (u, v, w)$: velocity field [m·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 [kg·m^{-2}·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.

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

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

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

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:

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

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

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 pre-computed high-fidelity trajectories. However, we retain physics-informed *regularization* without requiring it as the primary loss.

Chapter 3

Background

The objective here is to detail the main concepts/definitions existing algorithms needed to understand your work to be detailed later and to introduce the notations to be used.

Use examples

Don't forget to cite again these existing approaches

Chapter 4

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 4.1, 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.
The first model **........ EXPLAIN TRANSFORMER MODEL ....**
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

4.2 Approach 2: TCAN based Model

4.2.1 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 4.1 illustrates this process.

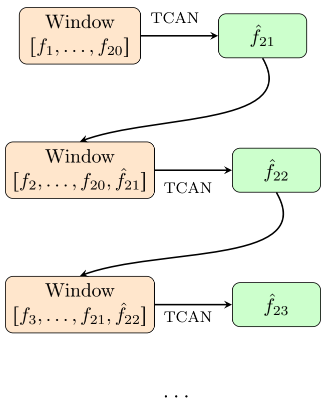

Figure 4.1: Autoregressive rollout process: each TCAN call maps a window of previous fields to one predicted field. In this example, the model first predicts f_{21} from the window $[f_1, \dots, f_{20}]$, then uses the updated window $[f_2, \dots, f_{20}, f_{21}]$ to predict f_{22}, and so on.

The architecture of this TCAN model consists of **three main components**:
1. 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.

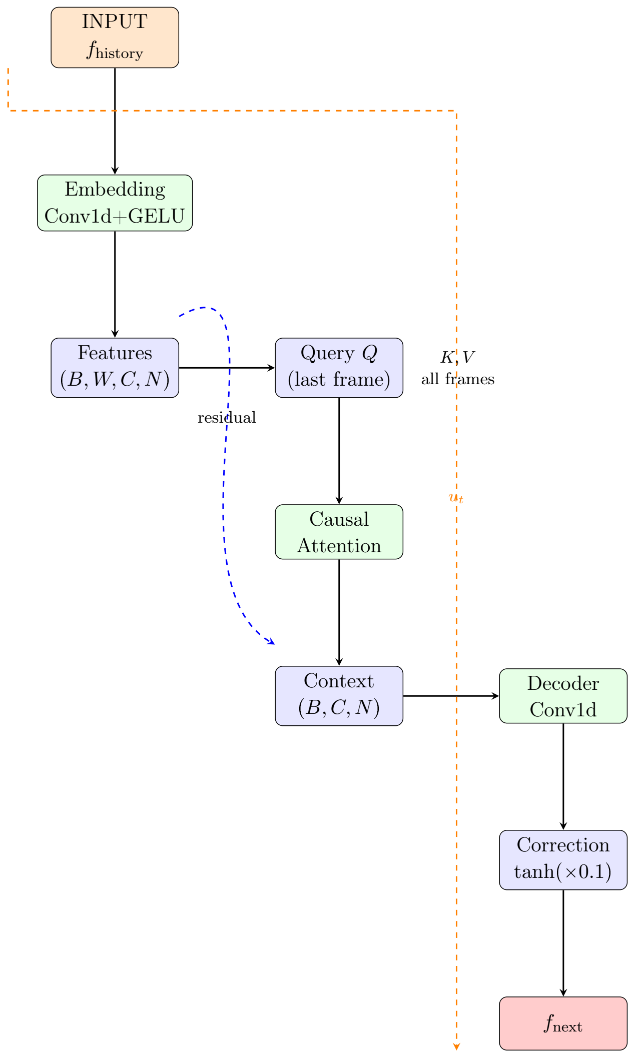

Figure 4.2: Data flow through the Causal Temporal Attention Network during one prediction step.

Figure 4.2 illustrates the complete data flow through the Causal Temporal Convolutional Attention Network during a single prediction step. Each node performs a specific transformation:

- f_{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 $Q \in \mathbb{R}^{B \times 32 \times N}$.
- **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** $\mathrm{softmax}(\frac{QK^T}{\sqrt{C}})$: Computes causal attention scores between the query (last frame) and all previous frames, producing temporal attention weights $\alpha_{w,p} = \mathrm{softmax}_w(QK^T/\sqrt{32})$ 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.
- $u_{\mathbf{next}}$ **(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 4.3 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.

4.2.2 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.

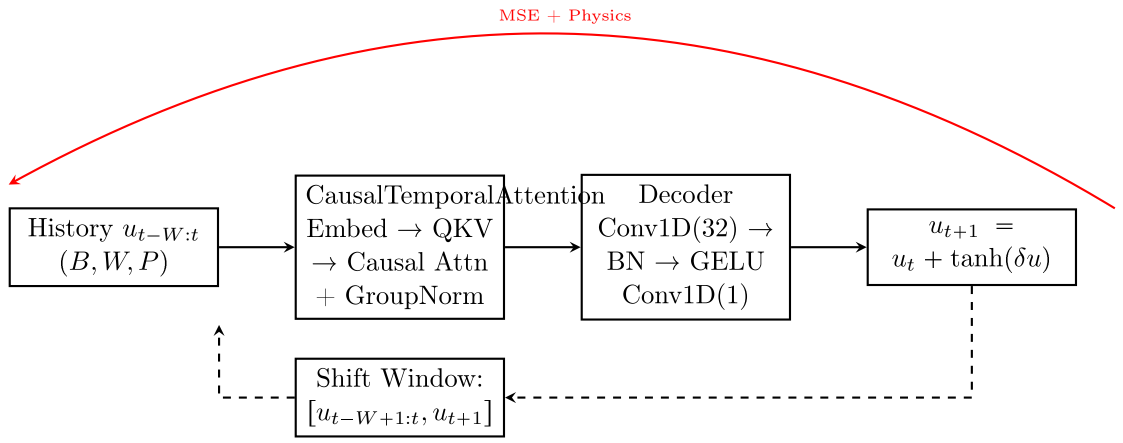
MSE + Physics
History $u_{t-W:t}$ (B, W, P)
CausalTemporalAttention Embed → QKV → Causal Attn + GroupNorm
Decoder Conv1D(32) → BN → GELU Conv1D(1)
u_{t+1} = $u_t + \tanh(\delta u)$
Shift Window: $[u_{t-W+1:t}, u_{t+1}]$

Figure 4.3: NFTM Architecture: Causal autoregressive prediction with physics-informed training.

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.

4.2.3 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 $\rightarrow$ 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.

4.2.4 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 $\hat{u}_{W+k}$ = TCAN(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.

4.2.5 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.

Table 4.2 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.

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.	$\|\int u\,dx - M_0\|$	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.	$\max \|\partial_x u\|$	Shock preserv.	$\pm 10\%$GT	Diverges
Grad. Err.	Grad. difference	Shock sharpness	< 0.05	> 0.2

Table 4.2: Evaluation Metrics for NFTM Performance Assessment

Chapter 5

Next Steps

5.1 Immediate Technical Improvements

Our current implementation has revealed several critical technical issues that require immediate attention to ensure robust and accurate predictions.

5.1.1 Boundary Condition Treatment

Current Issue. 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 ($u_{n+1} = 0$) 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.

Implementation. The padding scheme can be adjusted in the convolutional layer definitions by changing the `padding_mode` parameter from `'circular'` to `'replicate'` or `'zeros'`. We will evaluate both options and select based on empirical performance on held-out test trajectories.

5.1.2 Training Performance Optimization

Current Issue. 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.

Suspected Causes. Profiling suggests potential inefficiencies in the autoregressive rollout loop during training, possibly due to:

- Unnecessary gradient computations during intermediate rollout steps
- Inefficient tensor operations in the temporal attention mechanism
- Suboptimal batch processing in the training loop
- Redundant data transfers between CPU and GPU

Proposed Solution. We will refactor the training loop following established best practices [?]:

1. Implement gradient checkpointing to reduce memory overhead during backpropagation through time
2. Vectorize rollout operations where possible to leverage GPU parallelism
3. Profile the code systematically using PyTorch Profiler to identify bottlenecks
4. Review optimized reference implementations to identify architectural inefficiencies

5.1.3 Generalization Verification

Critical Finding. Our model demonstrates surprising generalization: after training on trajectories with a single viscosity value ν_{train}, it successfully predicts dynamics for unseen viscosity values $\nu_{\text{test}} \neq \nu_{\text{train}}$. This is unexpected, as the viscosity parameter fundamentally changes the PDE dynamics.

Verification Needed. We must rigorously verify that this generalization is genuine and not an artifact of:

- The model inadvertently accessing viscosity values from the dataset
- Similar-looking dynamics arising from different viscosity values
- Limited test set diversity failing to stress-test generalization

Experimental Protocol. We will:

1. Remove all explicit viscosity inputs from the model architecture
2. Generate test trajectories spanning viscosity values $\nu \in [0.001, 0.5]$ well outside the training range
3. Compute quantitative error metrics (relative L^2, PSNR) as a function of $|\nu_{\text{test}} - \nu_{\text{train}}|$
4. Compare against a baseline model explicitly conditioned on viscosity values

If genuine, this generalization capability represents a significant scientific finding that warrants detailed analysis and reporting.

5.2 Short-Term Research Objectives

Building on the immediate technical improvements, we outline our research agenda for the next two months.

5.2.1 Advanced Training Techniques

Teacher Forcing Implementation. Teacher forcing is a curriculum learning technique where the model is initially trained using ground truth states as input, gradually transitioning to using its own predictions. This technique is standard in the Neural ODE literature [?] and available in reference NFTM implementations. We will implement scheduled teacher forcing with an exponentially decaying schedule:

$$p_{\text{teacher}}(t) = p_0 \exp\left(-\frac{t}{\tau}\right) \tag{5.1}$$

where p_{teacher} is the probability of using ground truth at training step t, $p_0 = 0.9$ is the initial probability, and τ controls decay rate.

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

- Learning rate and scheduler parameters
- Rollout depth for curriculum learning
- Physics loss weighting coefficients λ_{energy}, λ_{grad}
- Architectural choices (kernel size, number of channels, attention heads)

The Bayesian approach constructs a surrogate model of the loss landscape to intelligently propose parameter combinations, dramatically reducing computational cost compared to grid search.

5.2.2 Extension to Two-Dimensional Burgers Equation

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

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

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{5.3}$$

Staged Approach. We will proceed incrementally:

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

The 2D case introduces significantly richer spatiotemporal dynamics including vortex formation, turbulent cascades, and boundary layer effects, providing a more stringent test of our architecture's capacity.

Architectural Modifications. The extension requires:

- Replacing 1D convolutions with 2D convolutions
- Extending attention mechanisms to handle 2D spatial dimensions
- Modifying the memory field from $\mathbb{R}^N$ to $\mathbb{R}^{N_x \times N_y \times 2}$ (two velocity components)

5.2.3 Baseline Comparisons and Alignment

To contextualize our results within the broader literature, we will implement and compare against established baseline methods:

Neural ODE. Neural ODEs [?] represent the most closely related approach. They parameterize the time derivative directly:

$$\frac{du}{dt} = f_\theta(u(t), t) \tag{5.4}$$

and use ODE solvers for integration. We will implement a CNN-based Neural ODE and compare prediction accuracy, computational cost, and stability during long rollouts.

PINNs. While PINNs [8] are known to struggle with the Burgers equation due to spectral bias [7], implementing a PINN baseline provides a reference point for data efficiency and physical consistency. We will train PINNs with identical physics constraints to isolate the effect of the training paradigm (physics loss vs. trajectory matching).

Evaluation Protocol. All methods will be trained on identical datasets and evaluated on:

- Relative L^2 error over rollout horizon
- Mass conservation accuracy
- Energy dissipation monotonicity
- Computational cost (training time, inference speed)

5.3 Medium-Term Development

Our medium-term objectives focus on demonstrating generalization across multiple PDE families and developing a unified neural solver.

5.3.1 Incompressible Navier-Stokes Equations

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

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} \tag{5.5}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{5.6}$$

Technical Challenges. The incompressibility constraint (5.6) couples the velocity and pressure fields, requiring either:

- Projection methods to enforce divergence-free velocities
- Pressure Poisson equation solvers
- Specialized loss functions penalizing divergence

We will investigate soft constraint approaches where the model is trained to minimize divergence rather than enforcing it exactly, following recent work on differentiable physics [8].

5.3.2 Multi-PDE Generalization

A key scientific question is whether a single architecture can learn dynamics across fundamentally different PDE families. We will test our framework on:

Heat Equation. The parabolic heat equation:

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u \tag{5.7}$$

represents pure diffusion without advection, testing whether the model can learn dynamics dominated by smoothing.

Reaction-Diffusion Systems. Systems like the Gray-Scott model couple diffusion with nonlinear reaction terms:

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u - uv^2 + F(1-u) \tag{5.8}$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + uv^2 - (F+k)v \tag{5.9}$$

These systems exhibit rich pattern formation (spots, stripes, spirals) that stress-test the model's ability to capture emergent spatial structures.

Evaluation Criteria. For each PDE, we will assess:

1. Whether the same architecture (without hyperparameter tuning) achieves reasonable accuracy
2. What architectural components transfer successfully across PDEs
3. Whether pre-training on one PDE improves performance on others

5.4 Long-Term Scientific Goals

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

5.4.1 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.

Approach. We will train models on trajectory data without providing:

- The explicit PDE form
- Parameter values (e.g., viscosity ν)
- Conservation laws or symmetries

Success would be demonstrated by the model learning dynamics that:

1. Match ground truth trajectories quantitatively
2. Respect underlying conservation laws (mass, energy, momentum)
3. Generalize to initial conditions and parameter regimes outside the training distribution

Interpretability. A key challenge is extracting interpretable equations from the learned model. We will investigate:

- Symbolic regression techniques to recover closed-form equations [?]
- Analysis of learned convolutional filters to identify differential operators
- Dimensionality reduction to discover latent coordinates aligned with physical variables

5.4.2 Conservation Law Enforcement

Current State. Hard-coding conservation laws (mass, momentum, energy) into neural network architectures remains an open problem. Existing approaches include:

- Hamiltonian Neural Networks [?]: Parameterize energy functions for conservative systems
- Port-Hamiltonian formulations: Extend to dissipative systems
- Projection methods: Post-process predictions to satisfy constraints

However, these methods either require restrictive assumptions (no dissipation) or conflict with gradient-based optimization when implemented as hard constraints.

Proposed Research Direction. We will develop soft constraint methods that:

1. Penalize violations of conservation laws through auxiliary loss terms
2. Learn to satisfy constraints approximately rather than exactly
3. Adaptively weight conservation vs. accuracy during training

For example, mass conservation can be enforced softly via:

$$\mathcal{L}_{\text{mass}} = \lambda_{\text{mass}} \left| \int_\Omega u_{\text{pred}}(x,t)\, dx - \int_\Omega u_0(x)\, dx \right|^2 \tag{5.10}$$

We will investigate whether soft constraints provide sufficient regularization to improve long-term stability without the optimization difficulties of hard constraints.

5.4.3 Improved Long-Horizon Prediction

Current Limitation. All current state-of-the-art methods for neural PDE solving suffer from divergence during long autoregressive rollouts. Errors accumulate, and predictions eventually become physically implausible or numerically unstable.

Research Questions.

- What is the fundamental limit on prediction horizon for learned simulators?
- Can we develop provable stability guarantees for neural time-steppers?
- How do architectural choices (attention vs. convolution, residual connections) affect error accumulation?

Proposed Investigations. We will explore:

1. **Multi-step prediction**: Train models to predict multiple steps ahead rather than single steps, reducing error accumulation
2. **Stability-aware training**: Penalize predictions that amplify high-frequency modes
3. **Hybrid approaches**: Combine learned time-steppers with traditional stabilization techniques (spectral filtering, artificial viscosity)

5.5 Summary and Timeline

We conclude by summarizing our roadmap with an approximate timeline:

Immediate (1-2 weeks).

- Fix boundary padding configuration
- Optimize training performance
- Verify generalization to unseen viscosity values
- Set up Weights & Biases for hyperparameter tuning

Short-term (1-2 months).

- Implement teacher forcing and curriculum learning
- Extend to 2D Burgers equation with obstacles
- Compare against Neural ODE and PINN baselines
- Systematic hyperparameter optimization

Medium-term (3-4 months).

- Incompressible Navier-Stokes implementation
- Multi-PDE generalization experiments (Heat, Reaction-Diffusion)
- Equation discovery investigations

Long-term (ongoing).

- Conservation law enforcement research
- Long-horizon prediction stability analysis
- Publication preparation and alignment with literature

This roadmap balances immediate practical improvements with ambitious scientific goals, positioning our work to make both methodological contributions (improved neural PDE solvers) and conceptual contributions (understanding what neural networks can learn about physical laws).

Appendix A

Appendix

The progress draft must be included in the appendix

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, 6 and 22.)

[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, 6, 22 and 23.)

Abstract:

Keywords: