Big Data Management & Analytics Master

Differential Equations Solver using NFTM for Burger Equations

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 General Context: Partial Differential Equations in Science and Engineering 1
1.1.1 Mathematical Framework of PDEs 1
1.1.2 Fundamental PDEs in Physics 2
1.1.3 Challenges in Classical PDE Solving 2
1.2 The Promise of Neural PDE Solvers 3
1.2.1 Mathematical Framework for Learning Solution Operators 3
1.2.2 Three Paradigms for Neural PDE Solving 4
1.3 The Problem: Burgers Equation as a Test Case 5
1.3.1 Mathematical Formulation 5
1.3.2 Physical Interpretation 6
1.3.3 Dimensionless Analysis and Reynolds Number 6
1.3.4 Shock Formation and the Inviscid Limit 7
1.3.5 Cole-Hopf Transformation and Exact Solutions 8
1.3.6 Why Burgers Equation for Neural PDE Research? 8
1.4 Challenges in Neural PDE Solving 9
1.4.1 Accuracy vs. Computational Efficiency Trade-off 9
1.4.2 Generalization to Unseen Parameter Regimes 10
1.4.3 Long-Time Stability and Error Accumulation 10
1.4.4 Physical Consistency and Conservation Laws 11
1.5 Project Objectives 11
1.5.1 Primary Research Questions 11

2 Related Work 13

3 Background 15

4 Our Methodology and Approach 17
4.1 Overview of the Proposed Pipeline 17
4.2 Data Generation: Stable Numerical Solver 17
4.2.1 Finite Difference Scheme 17
4.2.2 Dataset Construction 18
4.2.3 Quality Assurance 19
4.3 Neural Architecture: Improved NFTM with Temporal Attention 20
4.3.1 Input Representation and Window-Based Prediction 20
4.3.2 Causal Temporal Attention Mechanism 20
4.3.3 Convolutional Decoder 22
4.3.4 Final Prediction 22
4.4 Training Strategy 22
4.4.1 Loss Function Design 22
4.4.2 Curriculum Learning Strategy 23

- 4.4.3 Noise Injection for Robustness . . . 24
- 4.4.4 Optimization Details . . . 24
- 4.5 Evaluation Metrics . . . 25
  - 4.5.1 Accuracy Metrics . . . 25
  - 4.5.2 Physical Consistency Metrics . . . 26
  - 4.5.3 Correlation and Spectral Metrics . . . 26
- 4.6 Implementation Details . . . 26
  - 4.6.1 Software and Hardware . . . 26
  - 4.6.2 Reproducibility . . . 27
  - 4.6.3 Code Organization . . . 27
- 4.7 Summary . . . 27

5 Approaches for Neural PDE Simulators 29
- 5.1 Approach 1: Samuel's Model . . . 29
  - 5.1.1 Model Architecture . . . 29
  - 5.1.2 . . . 29
- 5.2 Approach 2: Causal Temporal Convolutional Attention Network . . . 29
  - 5.2.1 Model Architecture . . . 29
- 5.3 Temporal Attention Encoder . . . 29
- 5.4 Decoder and Residual Correction . . . 30
- 5.5 Autoregressive Training Procedure . . . 31
- 5.6 Evaluation Metrics . . . 31

6 Conclusion and Perspectives 33
- 6.1 Gl remarks . . . 33
- 6.2 Gl remarks about the presentation . . . 33

A Appendix 35

Bibliography 37

CHAPTER 1

Introduction

1.1 General Context: Partial Differential Equations in Science and Engineering

Partial differential equations (PDEs) constitute the fundamental mathematical framework for describing continuous phenomena across virtually all domains of science and engineering. From the quantum mechanics governing subatomic particles to the fluid dynamics shaping atmospheric patterns, PDEs provide the rigorous language through which we understand, predict, and control the physical world.

1.1.1 Mathematical Framework of PDEs

A partial differential equation is a relationship involving an unknown function u of multiple independent variables and its partial derivatives. The general form of a PDE can be expressed as:

$$F\left(x_1, x_2, \ldots, x_n, t, u, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \ldots, \frac{\partial^2 u}{\partial x_1^2}, \ldots, \frac{\partial^k u}{\partial x_1^k \partial x_2^l \cdots}\right) = 0 \tag{1.1}$$

where:

- $u = u(x_1, x_2, \ldots, x_n, t)$ is the unknown field variable
- $x_1, x_2, \ldots, x_n$ are spatial coordinates
- t is time
- F is a functional relationship
- $k, l, \ldots$ denote the order of partial derivatives

Classification by Order. The order of a PDE is determined by the highest derivative present. For a second-order PDE in two dimensions:

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} + Fu + G = 0 \tag{1.2}$$

The discriminant $\Delta = B^2 - 4AC$ determines the PDE type:

- **Elliptic** ($\Delta < 0$): Steady-state problems (Laplace, Poisson equations)
- **Parabolic** ($\Delta = 0$): Diffusion processes (Heat equation)
- **Hyperbolic** ($\Delta > 0$): Wave propagation (Wave equation)

1.1.2 Fundamental PDEs in Physics

1. Conservation of Mass (Continuity Equation). The principle that mass cannot be created or destroyed leads to:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{1.3}$$

where $\rho(x, y, z, t)$ is density and $\mathbf{u}(x, y, z, t)$ is the velocity field.

2. Conservation of Momentum (Navier-Stokes Equations). Newton's second law applied to a fluid element yields:

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = -\nabla p + \mu \nabla^2 \mathbf{u} + \rho \mathbf{g} \tag{1.4}$$

where:

- $p(x, y, z, t)$ is pressure [Pa]
- μ is dynamic viscosity [Pa·s]
- $\mathbf{g}$ is gravitational acceleration [m·s^{-2}]

3. Conservation of Energy (Heat Equation). Fourier's law of heat conduction combined with energy balance gives:

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) + Q \tag{1.5}$$

where:

- $T(x, y, z, t)$ is temperature [K]
- c_p is specific heat capacity [J·kg^{-1}·K^{-1}]
- k is thermal conductivity [W·m^{-1}·K^{-1}]
- Q is heat source term [W·m^{-3}]

4. Wave Propagation. The wave equation governing vibrations, acoustics, and electromagnetic radiation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u \tag{1.6}$$

where c is the wave speed [m·s^{-1}].

1.1.3 Challenges in Classical PDE Solving

Traditional numerical methods for solving PDEs face several fundamental limitations:

Computational Complexity. For a PDE discretized on an N-dimensional grid with M points per dimension and T time steps, the computational cost scales as:

$$\text{Cost} \sim \mathcal{O}(M^N \cdot T \cdot K) \tag{1.7}$$

where K is the cost per grid point update. For 3D problems with $M = 256$, this becomes prohibitive.

Curse of Dimensionality. The number of grid points grows exponentially with dimension:

$$\text{Grid points} = M^N \tag{1.8}$$

For $N = 3$ dimensions and $M = 100$ points per dimension: 10^6 grid points.

Stability Constraints. Explicit time-stepping schemes require timestep Δt to satisfy the CFL (Courant-Friedrichs-Lewy) condition:

$$\Delta t \leqslant C \cdot \frac{\Delta x}{|\mathbf{u}_{\text{max}}|} \tag{1.9}$$

where $C \leqslant 1$ for stability. For fine grids ($\Delta x \to 0$), this forces $\Delta t \to 0$, dramatically increasing computational cost.

Stiffness. Many physical systems exhibit stiffness, where multiple timescales coexist. The stiffness ratio:

$$S = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} \tag{1.10}$$

where λ are eigenvalues of the system operator. For $S \gg 1$, explicit methods require $\Delta t \sim 1/\lambda_{\text{max}}$ even when dynamics of interest occur at timescale $\sim 1/\lambda_{\text{min}}$.

1.2 The Promise of Neural PDE Solvers

1.2.1 Mathematical Framework for Learning Solution Operators

Rather than solving individual PDE instances, neural approaches learn **operators** that map between function spaces.

Operator Learning Formulation. Given a PDE of the form:

$$\mathcal{L}[u] = f \tag{1.11}$$

where $\mathcal{L}$ is a differential operator, we seek to learn the solution operator:

$$\mathcal{G} : f \mapsto u \tag{1.12}$$

such that $u = \mathcal{G}(f)$ satisfies $\mathcal{L}[u] = f$.

Universal Approximation for Operators. The neural operator framework [?] proves that certain neural architectures can approximate continuous operators between Banach spaces to arbitrary accuracy:

$$\|\mathcal{G} - \mathcal{G}_\theta\|_{L^p(\mathcal{X},\mathcal{Y})} < \varepsilon \tag{1.13}$$

where:

- $\mathcal{G}_\theta$ is the neural operator with parameters θ
- $\mathcal{X}$ is the input function space
- $\mathcal{Y}$ is the output function space
- ε is the approximation error

1.2.2 Three Paradigms for Neural PDE Solving

Paradigm 1: Physics-Informed Neural Networks (PINNs). PINNs [?] approximate the solution $u(x,t)$ directly with a neural network $u_\theta(x,t)$ by minimizing:

$$\mathcal{L}_{\text{PINN}} = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{IC}} + \mathcal{L}_{\text{BC}} \tag{1.14}$$

where:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{col}}} \sum_{i=1}^{N_{\text{col}}} |\mathcal{L}[u_\theta](x_i,t_i) - f(x_i,t_i)|^2 \tag{1.15}$$

$$\mathcal{L}_{\text{IC}} = \frac{1}{N_{\text{IC}}} \sum_{j=1}^{N_{\text{IC}}} |u_\theta(x_j,t_0) - u_0(x_j)|^2 \tag{1.16}$$

$$\mathcal{L}_{\text{BC}} = \frac{1}{N_{\text{BC}}} \sum_{k=1}^{N_{\text{BC}}} |u_\theta(x_{\partial\Omega},t_k) - g(x_{\partial\Omega},t_k)|^2 \tag{1.17}$$

Automatic Differentiation for PDE Residuals. The PDE residual is computed via automatic differentiation:

$$\frac{\partial u_\theta}{\partial t} = \frac{\partial}{\partial t}\left(\text{NeuralNet}_\theta(x,t)\right) \tag{1.18}$$

computed exactly using the chain rule through the computational graph.

Paradigm 2: Fourier Neural Operators (FNOs). FNOs [?] learn in the Fourier domain, leveraging the convolution theorem. The operator layer is:

$$v_{\ell+1}(x) = \sigma\left(W_\ell v_\ell(x) + \mathcal{F}^{-1}\left(R_\ell \cdot \mathcal{F}(v_\ell)\right)(x)\right) \tag{1.19}$$

where:

- $\mathcal{F}$ denotes Fourier transform: $\mathcal{F}(v)(k) = \int v(x)e^{-2\pi i k\cdot x}dx$
- $R_\ell(k)$ is a learnable weight in Fourier space
- W_ℓ is a local linear transformation
- σ is a nonlinear activation

Resolution Invariance. FNOs satisfy:

$$\mathcal{G}_\theta(v)|_M \approx \mathcal{G}_\theta(v|_M) \tag{1.20}$$

meaning the operator learned at resolution M generalizes to different resolutions.

Paradigm 3: Neural Field Turing Machines (NFTMs). NFTMs employ an iterative refinement strategy inspired by classical solvers:

$$u^{n+1}(x) = u^n(x) + \text{Controller}_\theta\left(\text{Patch}(u^n, x)\right) \tag{1.21}$$

where:

- u^n is the field at iteration n
- $\text{Patch}(u^n, x)$ extracts a local neighborhood around x
- Controller_θ is a neural network applying learned update rules

This formulation mimics iterative methods like Jacobi or Gauss-Seidel.

Classical Jacobi Iteration:

$$u_i^{n+1} = \frac{1}{2a}\left(f_i - bu_{i-1}^n - cu_{i+1}^n\right) \tag{1.22}$$

NFTM Analog:

$$u_i^{n+1} = u_i^n + \text{NN}_\theta\left([u_{i-k}^n, \ldots, u_i^n, \ldots, u_{i+k}^n]\right) \tag{1.23}$$

1.3 The Problem: Burgers Equation as a Test Case

1.3.1 Mathematical Formulation

The one-dimensional Burgers equation is a nonlinear PDE that serves as the simplest model capturing the essence of fluid dynamics:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{1.24}$$

with:

- Domain: $x \in [0, L]$, $t \in [0, T]$
- Initial condition: $u(x, 0) = u_0(x)$
- Boundary conditions: $u(0, t) = u_L(t)$, $u(L, t) = u_R(t)$
- Kinematic viscosity: $\nu > 0$ $[\text{m}^2{\cdot}\text{s}^{-1}]$

1.3.2 Physical Interpretation

Advection Term ($u\frac{\partial u}{\partial x}$). This nonlinear term represents self-advection or transport. A fluid element at position x with velocity $u(x,t)$ carries itself downstream. The nonlinearity leads to **wave steepening**.

Consider a smooth initial profile $u_0(x) = A\sin(kx)$ with $k = 2\pi/L$.

The characteristic curves along which information propagates satisfy:

$$\frac{dx}{dt} = u(x,t) \tag{1.25}$$

Since velocity varies spatially, different parts of the wave travel at different speeds, causing steepening and eventual shock formation.

Diffusion Term ($\nu\frac{\partial^2 u}{\partial x^2}$). This linear term represents viscous dissipation, smoothing gradients:

$$\frac{\partial u}{\partial t} = \nu\frac{\partial^2 u}{\partial x^2} \tag{1.26}$$

has the fundamental solution (Green's function):

$$G(x,t) = \frac{1}{\sqrt{4\pi\nu t}}\exp\left(-\frac{x^2}{4\nu t}\right) \tag{1.27}$$

showing that initially sharp features spread with width $\sigma \sim \sqrt{\nu t}$.

1.3.3 Dimensionless Analysis and Reynolds Number

Introduce characteristic scales:

- Length: L [m]
- Velocity: U [m·s^{-1}]
- Time: $T = L/U$ [s]

Define dimensionless variables:

$$x^* = \frac{x}{L}, \quad t^* = \frac{tU}{L}, \quad u^* = \frac{u}{U} \tag{1.28}$$

Substituting into Burgers equation:

$$\frac{U}{T}\frac{\partial u^*}{\partial t^*} + \frac{U^2}{L}u^*\frac{\partial u^*}{\partial x^*} = \frac{\nu U}{L^2}\frac{\partial^2 u^*}{\partial x^{*2}} \tag{1.29}$$

Dividing by U^2/L:

$$\frac{\partial u^*}{\partial t^*} + u^*\frac{\partial u^*}{\partial x^*} = \frac{1}{\mathrm{Re}}\frac{\partial^2 u^*}{\partial x^{*2}} \tag{1.30}$$

where the **Reynolds number** is:

$$\mathrm{Re} = \frac{UL}{\nu} = \frac{\text{inertial forces}}{\text{viscous forces}} \tag{1.31}$$

Physical Regimes.

1. **Low Reynolds Number** (Re $\ll 1$, ν large):
    - Viscosity dominates
    - Smooth, diffusion-controlled solutions
    - No shock formation
    - Analytical solutions via similarity methods
2. **Moderate Reynolds Number** (Re ~ 1):
    - Balanced advection and diffusion
    - Weak shocks with finite width
    - Transition regime
3. **High Reynolds Number** (Re $\gg 1$, ν small):
    - Inertia dominates
    - Sharp shocks and discontinuities
    - Asymptotically approaches inviscid limit
    - Highly challenging for numerical methods

1.3.4 Shock Formation and the Inviscid Limit

For the inviscid Burgers equation ($\nu = 0$):

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \tag{1.32}$$

the method of characteristics gives:

$$\frac{dx}{dt} = u, \quad \frac{du}{dt} = 0 \tag{1.33}$$

implying u is constant along characteristics: $u = u_0(x_0)$ where x_0 is the initial position.

Shock Formation Time. Consider initial condition $u_0(x) = -\alpha x$ with $\alpha > 0$. Characteristics satisfy:

$$x(t) = x_0 + u_0(x_0)t = x_0 - \alpha x_0 t = x_0(1 - \alpha t) \tag{1.34}$$

Characteristics originating from different x_0 intersect when:

$$t_{\text{shock}} = \frac{1}{\max\left(-\frac{du_0}{dx}\right)} = \frac{1}{\alpha} \tag{1.35}$$

At this time, u becomes multi-valued (unphysical), and a shock discontinuity forms.

Rankine-Hugoniot Jump Condition. Across a shock at position $s(t)$, conservation laws require:

$$\frac{ds}{dt} = \frac{1}{2}(u_L + u_R) \tag{1.36}$$

where $u_L = u(s^-, t)$ and $u_R = u(s^+, t)$ are velocities on left and right sides.

For viscous Burgers with small ν, shocks have finite width:

$$\delta_{\text{shock}} \sim \frac{\nu}{u_L - u_R} \tag{1.37}$$

This rapid variation challenges numerical methods and provides an ideal test for neural solvers.

1.3.5 Cole-Hopf Transformation and Exact Solutions

The Cole-Hopf transformation [?, ?] linearizes Burgers equation:

Let:

$$u(x,t) = -2\nu \frac{\partial}{\partial x} \ln \phi(x,t) \tag{1.38}$$

Substituting into Burgers equation yields the **heat equation**:

$$\frac{\partial \phi}{\partial t} = \nu \frac{\partial^2 \phi}{\partial x^2} \tag{1.39}$$

Solution Procedure.

1. Given $u_0(x)$, compute $\phi_0(x) = \exp\left(-\frac{1}{2\nu}\int_0^x u_0(\xi)d\xi\right)$
2. Solve heat equation: $\phi(x,t) = \int_{-\infty}^{\infty} G(x-\xi,t)\phi_0(\xi)d\xi$
3. Recover $u(x,t) = -2\nu\frac{\partial \ln \phi}{\partial x}$

Example: Gaussian Initial Condition. For $u_0(x) = A\exp\left(-\frac{x^2}{2\sigma^2}\right)$, the exact solution is:

$$u(x,t) = \frac{Ax}{1 + \frac{At}{2\nu}\exp\left(-\frac{x^2}{2\sigma^2(1+\frac{2\nu t}{\sigma^2})}\right)} \tag{1.40}$$

This provides ground truth for validation of neural solvers.

1.3.6 Why Burgers Equation for Neural PDE Research?

The Burgers equation is ideal for benchmarking neural PDE solvers because it:

1. **Captures essential physics:**
    - Nonlinearity (wave steepening)
    - Dissipation (viscous damping)
    - Shock formation
    - Parameter dependence (Reynolds number)

2. **Has analytical solutions:** Via Cole-Hopf, enabling exact validation
3. **Computationally tractable:** 1D allows rapid iteration
4. **Generalizes to Navier-Stokes:** The velocity components in 2D/3D Navier-Stokes satisfy coupled Burgers-like equations
5. **Well-studied benchmark:** Extensive literature for comparison (PINNs, FNOs, classical methods)

1.4 Challenges in Neural PDE Solving

Despite the promise of neural approaches, several fundamental challenges persist.

1.4.1 Accuracy vs. Computational Efficiency Trade-off

Quantitative Analysis. Traditional finite difference methods achieve accuracy:

$$\|u_{\text{numerical}} - u_{\text{exact}}\|_{L^2} = \mathcal{O}(\Delta x^p + \Delta t^q) \tag{1.41}$$

where p, q are the spatial and temporal orders of accuracy (typically 2-4).

For $\Delta x = 10^{-3}$, second-order method: error $\sim 10^{-6}$.

Neural methods currently achieve:

$$\|u_{\text{neural}} - u_{\text{exact}}\|_{L^2} \sim 10^{-3} \text{ to } 10^{-5} \tag{1.42}$$

Computational Cost Comparison. Finite Difference for Burgers 1D:

- Grid points: N
- Time steps: T
- Operations per step: $\mathcal{O}(N)$
- Total cost: $\mathcal{O}(NT)$

For $N = 256$, $T = 10^4$ steps: $\sim 2.5 \times 10^6$ operations

Neural Forward Pass:

- Layer evaluations: L layers
- Operations per layer: $\mathcal{O}(D^2)$ where D is hidden dimension
- Total cost: $\mathcal{O}(LD^2)$

For $L = 5$, $D = 64$: $\sim 2 \times 10^4$ operations per prediction

Speedup Potential. Once trained, neural solver can be 100× faster per forward pass, but with 1-2 orders of magnitude lower accuracy.

1.4.2 Generalization to Unseen Parameter Regimes

Statistical Learning Perspective. Given training set $\mathcal{D} = \{(\nu_i, u_0^{(i)}, u_{\text{target}}^{(i)})\}_{i=1}^N$, we learn mapping:

$$\mathcal{G}_\theta : (\nu, u_0) \mapsto u(t) \tag{1.43}$$

Generalization Error.

$$\mathbb{E}_{(\nu,u_0)\sim p_{\text{test}}}\left[\|u_{\text{pred}} - u_{\text{true}}\|^2\right] \tag{1.44}$$

For parameter extrapolation ($\nu_{\text{test}} \notin [\nu_{\text{min}}^{\text{train}}, \nu_{\text{max}}^{\text{train}}]$), generalization degrades.

Empirical observations show:

$$\text{Error}(\nu) \approx \text{Error}_{\text{train}} \cdot \left(1 + \alpha \cdot \frac{|\nu - \nu_{\text{train}}|}{\nu_{\text{train}}}\right)^\beta \tag{1.45}$$

with $\beta \approx 2 - 3$ for most neural solvers, indicating rapid degradation outside training distribution.

1.4.3 Long-Time Stability and Error Accumulation

Autoregressive Rollout. For time-dependent PDEs, neural solvers predict iteratively:

$$u^{n+1} = \mathcal{G}_\theta(u^n) \tag{1.46}$$

Error Propagation Analysis. Let $\varepsilon^n = u_{\text{pred}}^n - u_{\text{true}}^n$ be the error at step n.

Linearizing around true trajectory:

$$\varepsilon^{n+1} \approx J\varepsilon^n + \eta^n \tag{1.47}$$

where $J = \frac{\partial \mathcal{G}_\theta}{\partial u}\Big|_{u_{\text{true}}^n}$ is the Jacobian and η^n is per-step error.

Spectral Stability. If eigenvalues of J satisfy $|\lambda_i| \geqslant 1$, errors grow exponentially:

$$\|\varepsilon^n\| \sim \|\varepsilon^0\| \cdot \lambda_{\text{max}}^n + \frac{\|\eta\|}{1 - \lambda_{\text{max}}} \tag{1.48}$$

For $\lambda_{\text{max}} = 1.01$ and $n = 1000$ steps:

$$\text{Error growth} \sim (1.01)^{1000} \approx 20{,}000\times \tag{1.49}$$

Current State-of-the-Art. Most neural solvers diverge after:

$$n_{\text{stable}} \approx 10 - 100 \times n_{\text{train}} \tag{1.50}$$

This limits practical applicability to short-horizon predictions.

1.4.4 Physical Consistency and Conservation Laws

Conservation Law Violation. Consider mass conservation for Burgers:

$$M(t) = \int_0^L u(x,t)dx \tag{1.51}$$

For periodic BCs, exact evolution satisfies:

$$\frac{dM}{dt} = -\nu \left[\frac{\partial u}{\partial x}\right]_0^L = 0 \tag{1.52}$$

Neural solvers typically violate this:

$$\left|\frac{M^{\text{pred}}(t) - M(0)}{M(0)}\right| \sim 10^{-2} \text{ to } 10^{-1} \tag{1.53}$$

after long rollouts.

Energy Dissipation. For Burgers, kinetic energy:

$$E(t) = \frac{1}{2}\int_0^L u^2(x,t)dx \tag{1.54}$$

satisfies:

$$\frac{dE}{dt} = -\nu \int_0^L \left(\frac{\partial u}{\partial x}\right)^2 dx \leqslant 0 \tag{1.55}$$

Neural solvers often exhibit energy drift, with dE/dt fluctuating in sign, violating second law of thermodynamics.

1.5 Project Objectives

This thesis investigates the application of **Neural Field Turing Machines (NFTMs)** to solve the one-dimensional Burgers equation, with a focus on developing architectural innovations that address the challenges outlined above.

1.5.1 Primary Research Questions

RQ1: Architecture Design. Can a hypernetwork-based NFTM controller, conditioned on problem parameters, achieve accuracy competitive with specialized neural operators while maintaining computational efficiency?

Mathematical Formulation. Design $\text{Controller}_\theta : (\text{Patch}(u^n), \nu) \mapsto \Delta u$ where:

- Hypernetwork generates weights: $\theta = h_\phi(\nu)$
- Multi-scale processing: kernels $k \in \{3, 7, 15\}$
- Target: $\|u_{\text{NFTM}} - u_{\text{FD}}\|_{L^2} < 10^{-4}$

RQ2: Parameter Generalization. Does conditioning on viscosity ν enable extrapolation to unseen parameter values, and what are the quantitative limits of this generalization?

Experimental Protocol.

- Train: $\nu \in [0.01, 0.05]$
- Test: $\nu \in \{0.005, 0.075, 0.10\}$
- Measure: $\text{Error}(\nu_{\text{test}})/\text{Error}(\nu_{\text{train}})$

RQ3: Long-Time Stability. Can architectural choices (residual connections, normalization, multi-scale kernels) improve rollout stability beyond current state-of-the-art?

Stability Metric.

$$T_{\text{stable}} = \max\{t : \|u_{\text{pred}}(t) - u_{\text{true}}(t)\|_{L^2} < \varepsilon_{\text{threshold}}\} \tag{1.56}$$

Target: $T_{\text{stable}} > 50 \times T_{\text{train}}$

RQ4: Computational Efficiency. What is the accuracy-speed Pareto frontier for NFTM compared to classical solvers and alternative neural methods?

Metrics.

- Forward pass time (ms)
- Memory usage (GB)
- Accuracy (L^2 error)
- Amortized cost for parameter sweeps

Chapter 2

Related Work

It is expected to find:

- The research articles related to your defined problem/objective
- Try to present them by category, the main ideas behind and their limitations
- Justify your directions/choices. This part should make a link with the next chapter.

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

Our Methodology and Approach

This chapter presents our comprehensive methodology for solving the one-dimensional Burgers equation using Neural Field Turing Machines (NFTMs). We detail the complete pipeline from data generation through model architecture, training procedures, and evaluation metrics. Our approach builds upon the NFTM framework while introducing several key innovations to improve stability, physical consistency, and generalization capabilities.

4.1 Overview of the Proposed Pipeline

Our neural PDE solver pipeline consists of four main components, illustrated in Figure **??**:

1. **Physics-Based Data Generation**: High-fidelity numerical solutions of the Burgers equation using stable finite difference methods
2. **Neural Architecture**: A causal temporal attention mechanism combined with convolutional decoders
3. **Training Strategy**: Multi-stage curriculum learning with physics-informed regularization
4. **Evaluation Framework**: Comprehensive metrics assessing accuracy, stability, and physical consistency

The complete workflow operates as follows: training trajectories are generated by solving the Burgers equation numerically for various initial conditions and viscosity parameters. These trajectories are then used to train a neural network that learns to predict future states given a window of historical states. During inference, the model performs autoregressive rollout to generate long-term predictions. Finally, predictions are evaluated against ground truth using multiple metrics to assess both numerical accuracy and physical plausibility.

4.2 Data Generation: Stable Numerical Solver

4.2.1 Finite Difference Scheme

To generate high-quality training data, we implement a stable numerical solver for the one-dimensional Burgers equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{4.1}$$

We discretize the spatial domain $x \in [0, 2\pi]$ with periodic boundary conditions using $N = 128$ uniformly spaced grid points:

$$x_i = \frac{2\pi i}{N}, \quad i = 0, 1, \ldots, N-1 \tag{4.2}$$

The spatial step size is:

$$\Delta x = \frac{2\pi}{N} \approx 0.049 \tag{4.3}$$

Diffusion Term. The viscous diffusion term is discretized using central differences, which is second-order accurate in space:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_i} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} = D_i \tag{4.4}$$

This stencil is applied with periodic boundary conditions: $u_{-1} = u_{N-1}$ and $u_N = u_0$.

Advection Term. For the nonlinear advection term $u\frac{\partial u}{\partial x}$, we employ an upwind differencing scheme to maintain numerical stability:

$$\left.\frac{\partial u}{\partial x}\right|_{x_i} \approx \begin{cases} \frac{u_i - u_{i-1}}{\Delta x} & \text{if } u_i > 0 \text{ (backward difference)} \\ \frac{u_{i+1} - u_i}{\Delta x} & \text{if } u_i \leqslant 0 \text{ (forward difference)} \end{cases} \tag{4.5}$$

The upwind scheme selects the differencing direction based on the local flow direction, which prevents spurious oscillations in regions with steep gradients or shocks.

Time Integration. We use the forward Euler method for time integration with timestep $\Delta t = 0.01$:

$$u_i^{n+1} = u_i^n - \Delta t \cdot u_i^n \left(\frac{\partial u}{\partial x}\right)_i^n + \Delta t \cdot \nu \cdot D_i^n \tag{4.6}$$

To ensure stability, the timestep satisfies the CFL (Courant-Friedrichs-Lewy) condition:

$$\Delta t \leqslant \min\left(\frac{\Delta x}{|u_{\max}|}, \frac{\Delta x^2}{2\nu}\right) \tag{4.7}$$

For our parameters ($\Delta x \approx 0.049$, $\nu \in [0.01, 0.1]$, $|u_{\max}| \lesssim 1$), $\Delta t = 0.01$ satisfies this stability criterion.

4.2.2 Dataset Construction

Sampling Strategy. For each trajectory in our dataset, we randomly sample:

- **Viscosity parameter**: $\nu \sim \mathcal{U}(0.01, 0.1)$, where $\mathcal{U}(a, b)$ denotes the uniform distribution
- **Initial condition**: $u_0(x) = \sin(x)$ (smooth sinusoidal profile)

The sinusoidal initial condition is chosen because:

1. It is smooth and differentiable (realistic physical scenario)

2. It produces rich dynamics including wave steepening and eventual shock formation for small ν
3. It has zero mean: $\int_0^{2\pi} \sin(x)\, dx = 0$, which tests mass conservation
4. It provides analytical benchmarks via the Cole-Hopf transformation

Trajectory Length. Each trajectory consists of $T = 100$ timesteps, spanning total time $T_{\text{total}} = 1.0$ seconds. The state at each timestep is stored:

$$\mathcal{T}_{\text{trajectory}} = \{u^0, u^1, \ldots, u^{99}\}, \quad u^t \in \mathbb{R}^{128} \tag{4.8}$$

Dataset Split. We generate:

- **Training set**: 1000 trajectories (100,000 state snapshots)
- **Test set**: 200 trajectories (20,000 state snapshots)

No trajectory in the test set has the same viscosity parameter ν as any training trajectory, ensuring evaluation on truly unseen parameter regimes.

4.2.3 Quality Assurance

We verify the quality of generated data through:

Mass Conservation. For periodic boundary conditions, the total mass $M(t) = \int_0^{2\pi} u(x,t)\, dx$ should be conserved. Numerically:

$$M^n = \Delta x \sum_{i=0}^{N-1} u_i^n \tag{4.9}$$

We verify:

$$\left| \frac{M^{99} - M^0}{M^0 + \varepsilon} \right| < 10^{-3} \tag{4.10}$$

where $\varepsilon = 10^{-6}$ prevents division by zero for the sinusoidal IC where $M^0 \approx 0$.

Energy Dissipation. The kinetic energy:

$$E^n = \frac{\Delta x}{2} \sum_{i=0}^{N-1} (u_i^n)^2 \tag{4.11}$$

must monotonically decrease due to viscous dissipation:

$$E^{n+1} \leqslant E^n + \varepsilon_{\text{tol}} \tag{4.12}$$

where $\varepsilon_{\text{tol}} = 10^{-6}$ accounts for numerical roundoff.

Stability Check. We reject any trajectory where:

$$\max_{t,i} |u_i^t| > 5 \quad \text{or} \quad \exists\,(t,i) : \text{isnan}(u_i^t) \vee \text{isinf}(u_i^t) \tag{4.13}$$

These checks ensure only physically plausible and numerically stable trajectories enter the training dataset.

4.3 Neural Architecture: Improved NFTM with Temporal Attention

Our neural architecture enhances the basic NFTM framework with causal temporal attention and multi-scale processing.

4.3.1 Input Representation and Window-Based Prediction

Historical Window. Rather than predicting the next state from a single previous state, we use a sliding window of length W:

$$\mathbf{U}_{\text{hist}} = [u^{t-W+1}, u^{t-W+2}, \ldots, u^t] \in \mathbb{R}^{W \times N} \tag{4.14}$$

In our implementation, $W = 20$, providing the model with approximately 0.2 seconds of history to capture temporal dynamics.

Prediction Task. Given $\mathbf{U}_{\text{hist}}$, the model predicts the next state:

$$\hat{u}^{t+1} = f_\theta(\mathbf{U}_{\text{hist}}) \tag{4.15}$$

For autoregressive rollout over K steps:

$$\begin{aligned}
\hat{u}^{t+1} &= f_\theta([u^{t-W+2}, \ldots, u^t]) \\
\hat{u}^{t+2} &= f_\theta([u^{t-W+3}, \ldots, u^t, \hat{u}^{t+1}]) \\
&\vdots \\
\hat{u}^{t+K} &= f_\theta([\hat{u}^{t+K-W}, \ldots, \hat{u}^{t+K-1}])
\end{aligned} \tag{4.16}$$

This autoregressive formulation allows arbitrary-length predictions but accumulates errors over time.

4.3.2 Causal Temporal Attention Mechanism

Motivation. The temporal attention mechanism allows the model to selectively weight different historical timesteps based on their relevance to the current prediction. This is particularly important for PDEs where recent states may be more informative than distant past states, but long-range temporal dependencies still matter.

Architecture Overview. The causal temporal attention module processes the history window $\mathbf{U}_{\text{hist}} \in \mathbb{R}^{B\times W\times N}$ (batch size B, window size W, spatial points N) in three stages:

1. **Feature Embedding**: Each timestep is embedded into a high-dimensional feature space
2. **Attention Computation**: Query-Key-Value attention focuses on relevant historical information
3. **Context Aggregation**: Weighted temporal features are aggregated into a context vector

Feature Embedding. First, we embed each spatial field into a feature space using a 1D convolutional layer:

$$\mathbf{H}^t = \text{GELU}(\text{Conv1D}(u^t; \theta_{\text{emb}})) \tag{4.17}$$

where $\mathbf{H}^t \in \mathbb{R}^{d\times N}$ and $d = 32$ is the embedding dimension.

For the full window:

$$\mathbf{H} = [\mathbf{H}^{t-W+1}, \ldots, \mathbf{H}^t] \in \mathbb{R}^{W\times d\times N} \tag{4.18}$$

Query-Key-Value Projections. We use the most recent frame $\mathbf{H}^t$ to generate queries $\mathbf{Q}$, while all historical frames generate keys $\mathbf{K}$ and values $\mathbf{V}$:

$$\mathbf{Q} = \text{Conv1D}(\mathbf{H}^t; \theta_Q) \in \mathbb{R}^{d\times N} \tag{4.19}$$

$$\mathbf{K}^\tau = \text{Conv1D}(\mathbf{H}^\tau; \theta_K) \in \mathbb{R}^{d\times N}, \quad \tau \in [t-W+1, t] \tag{4.20}$$

$$\mathbf{V}^\tau = \text{Conv1D}(\mathbf{H}^\tau; \theta_V) \in \mathbb{R}^{d\times N} \tag{4.21}$$

Attention Scores. The attention scores measure similarity between the current query and past keys:

$$\alpha_i^\tau = \frac{\mathbf{Q}_i^T \mathbf{K}_i^\tau}{\sqrt{d}}, \quad i = 1, \ldots, N \tag{4.22}$$

Normalization via softmax across the temporal dimension yields attention weights:

$$a_i^\tau = \frac{\exp(\alpha_i^\tau)}{\sum_{\tau'=t-W+1}^{t} \exp(\alpha_i^{\tau'})} \tag{4.23}$$

This ensures:

$$\sum_{\tau=t-W+1}^{t} a_i^\tau = 1, \quad a_i^\tau \geqslant 0 \tag{4.24}$$

Context Aggregation. The weighted sum of value vectors produces the context representation:

$$\mathbf{C}_i = \sum_{\tau=t-W+1}^{t} a_i^\tau \mathbf{V}_i^\tau \tag{4.25}$$

Output Projection and Residual Connection. The context is projected back to the feature dimension and combined with the last frame via residual connection:

$$\mathbf{F} = \text{GroupNorm}(\text{Conv1D}(\mathbf{C}; \theta_{\text{out}}) + \mathbf{H}^t) \tag{4.26}$$

The GroupNorm stabilizes training by normalizing activations across channel groups.

4.3.3 Convolutional Decoder

The decoder transforms the attention-enriched features $\mathbf{F} \in \mathbb{R}^{d \times N}$ into spatial corrections $\Delta u \in \mathbb{R}^{1 \times N}$:

$$\begin{aligned} \mathbf{D}_1 &= \text{GELU}(\text{BatchNorm}(\text{Conv1D}(\mathbf{F}; \text{kernel} = 5))) \\ \Delta u &= \text{Conv1D}(\mathbf{D}_1; \text{kernel} = 5, \text{out\_channels} = 1) \end{aligned} \tag{4.27}$$

Multi-Scale Receptive Field. The two convolutional layers with kernel size 5 provide an effective receptive field:

$$R_{\text{eff}} = 1 + 2 \times (5 - 1) = 9 \tag{4.28}$$

This covers approximately $\frac{9}{128} \times 2\pi \approx 0.44$ spatial units, sufficient to capture local gradients and shock structures.

Bounded Correction via Tanh. To prevent instabilities, corrections are bounded:

$$\Delta u_{\text{bounded}} = \tanh(\Delta u) \cdot c_{\text{clip}} \tag{4.29}$$

where $c_{\text{clip}} = 0.1$. This ensures:

$$|\Delta u_{\text{bounded}}| \leqslant 0.1 \tag{4.30}$$

preventing single-step divergence during autoregressive rollout.

4.3.4 Final Prediction

The next state is computed via residual update:

$$\hat{u}^{t+1} = u^t + \Delta u_{\text{bounded}} \tag{4.31}$$

This formulation has two advantages:

1. **Easier Optimization**: Learning perturbations Δu rather than absolute states u^{t+1} simplifies the learning task
2. **Physical Alignment**: For small timesteps, the true dynamics satisfy $u^{t+1} \approx u^t + \Delta t \cdot f(u^t)$, matching our residual formulation

4.4 Training Strategy

4.4.1 Loss Function Design

Our loss function combines multiple terms to enforce both accuracy and physical consistency.

Mean Squared Error (MSE). The primary loss is the L2 prediction error:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{BN} \sum_{b=1}^{B} \sum_{i=1}^{N} (\hat{u}_i^b - u_i^b)^2 \tag{4.32}$$

Gradient Matching Loss. To preserve spatial structures and shocks, we match spatial gradients:

$$\mathcal{L}_{\text{grad}} = \frac{1}{BN} \sum_{b=1}^{B} \sum_{i=1}^{N} \left(\frac{\partial \hat{u}_i^b}{\partial x} - \frac{\partial u_i^b}{\partial x} \right)^2 \tag{4.33}$$

Gradients are computed via central differences:

$$\frac{\partial u_i}{\partial x} \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x} \tag{4.34}$$

Energy Dissipation Constraint. The Burgers equation dissipates energy due to viscosity:

$$E(t + \Delta t) \leqslant E(t) \tag{4.35}$$

We enforce this via a penalty:

$$\mathcal{L}_{\text{energy}} = \frac{1}{B} \sum_{b=1}^{B} \text{ReLU} \left(\frac{E(\hat{u}^b) - E(u^{t,b})}{E(u^{t,b}) + \varepsilon} \right) \tag{4.36}$$

where:

$$E(u) = \frac{\Delta x}{2} \sum_{i=1}^{N} u_i^2 \tag{4.37}$$

The ReLU ensures we only penalize energy *increases*, not decreases. Normalization by $E(u^t)$ makes the penalty scale-invariant.

Combined Loss. The total loss is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{MSE}} + \lambda_{\text{grad}} \mathcal{L}_{\text{grad}} + \lambda_{\text{energy}} \mathcal{L}_{\text{energy}} \tag{4.38}$$

with weights $\lambda_{\text{grad}} = 0.1$ and $\lambda_{\text{energy}} = 0.05$.

4.4.2 Curriculum Learning Strategy

To improve long-term stability, we employ curriculum learning that gradually increases task difficulty.

Rollout Depth Scheduling. Early in training, we use short rollouts; later, we extend to longer predictions:

$$K_{\text{rollout}}(\text{epoch}) = \begin{cases} 8 & \text{epoch} < 10 \\ 16 & 10 \leqslant \text{epoch} < 30 \\ 32 & \text{epoch} \geqslant 30 \end{cases} \tag{4.39}$$

During each training iteration, we unroll for K_{rollout} steps and backpropagate through the entire sequence:

$$\mathcal{L}_{\text{rollout}} = \frac{1}{K_{\text{rollout}}} \sum_{k=1}^{K_{\text{rollout}}} \mathcal{L}_{\text{total}}(\hat{u}^{t+k}, u^{t+k}) \tag{4.40}$$

Teacher Forcing. To prevent error accumulation during training, we occasionally use ground truth states instead of predictions:

$$u_{\text{next}}^{t+k} = \begin{cases} u_{\text{true}}^{t+k} & \text{with probability } p_{\text{TF}} \\ \hat{u}^{t+k} & \text{with probability } 1 - p_{\text{TF}} \end{cases} \tag{4.41}$$

The teacher forcing rate decays over training:

$$p_{\text{TF}}(\text{epoch}) = \begin{cases} 0.5 & \text{epoch} < 10 \\ 0.2 & 10 \leqslant \text{epoch} < 30 \\ 0.05 & \text{epoch} \geqslant 30 \end{cases} \tag{4.42}$$

This provides stability early (when the model is weak) while ensuring the model learns to handle its own predictions later.

4.4.3 Noise Injection for Robustness

After epoch 20, we inject Gaussian noise into the input history to improve robustness:

$$\tilde{\mathbf{U}}_{\text{hist}} = \mathbf{U}_{\text{hist}} + \varepsilon \mathcal{N}(0, I) \tag{4.43}$$

where:

$$\varepsilon = 0.01 \times \min\left(1.0, \frac{\text{epoch}}{50}\right) \tag{4.44}$$

This prevents overfitting to exact trajectories and improves generalization to perturbed initial conditions.

4.4.4 Optimization Details

Optimizer. We use AdamW optimizer with:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} - \lambda_{\text{WD}} \theta_t \tag{4.45}$$

where:

- Learning rate: $\eta = 0.001$
- Weight decay: $\lambda_{\text{WD}} = 10^{-4}$
- $\beta_1 = 0.9$, $\beta_2 = 0.999$
- $\varepsilon = 10^{-8}$

Learning Rate Schedule. Cosine annealing reduces the learning rate:

$$\eta_t = \eta_{\min} + \frac{\eta_{\max} - \eta_{\min}}{2}\left(1 + \cos\left(\frac{t}{T_{\max}}\pi\right)\right) \tag{4.46}$$

with $\eta_{\max} = 0.001$, $\eta_{\min} = 0$, and $T_{\max} = 50$ epochs.

Gradient Clipping. To prevent gradient explosion during backpropagation through time:

$$\|\nabla_\theta \mathcal{L}\| > 1.0 \implies \nabla_\theta \mathcal{L} \leftarrow \frac{\nabla_\theta \mathcal{L}}{\|\nabla_\theta \mathcal{L}\|} \tag{4.47}$$

Batch Size and Epochs.

- Batch size: $B = 32$
- Total epochs: 50
- Training samples per epoch: $\frac{1000}{32} \approx 31$ batches

4.5 Evaluation Metrics

We employ comprehensive metrics to assess model performance across multiple dimensions.

4.5.1 Accuracy Metrics

Mean Squared Error (MSE).

$$\text{MSE} = \frac{1}{TN}\sum_{t=1}^{T}\sum_{i=1}^{N}(\hat{u}_i^t - u_i^t)^2 \tag{4.48}$$

Relative L2 Error. Scale-invariant error:

$$E_{L^2}^{\text{rel}} = \frac{\|\hat{\mathbf{u}} - \mathbf{u}\|_2}{\|\mathbf{u}\|_2 + \varepsilon} \tag{4.49}$$

Per-Timestep Relative L2.

$$E_{L^2}^{\text{rel}}(t) = \frac{\sqrt{\sum_{i=1}^{N}(\hat{u}_i^t - u_i^t)^2}}{\sqrt{\sum_{i=1}^{N}(u_i^t)^2} + \varepsilon} \tag{4.50}$$

This reveals when predictions diverge during rollout.

Peak Signal-to-Noise Ratio (PSNR).

$$\text{PSNR} = 20\log_{10}\left(\frac{\text{RANGE}}{\sqrt{\text{MSE}}}\right) \tag{4.51}$$

where $\text{RANGE} = \max(u) - \min(u)$.

Higher PSNR indicates better reconstruction quality.

Structural Similarity Index (SSIM). Measures perceptual similarity:

$$\mathrm{SSIM}(\mathbf{u}, \hat{\mathbf{u}}) = \frac{(2\mu_u \mu_{\hat{u}} + C_1)(2\sigma_{u\hat{u}} + C_2)}{(\mu_u^2 + \mu_{\hat{u}}^2 + C_1)(\sigma_u^2 + \sigma_{\hat{u}}^2 + C_2)} \tag{4.52}$$

SSIM $\in [-1, 1]$, with 1 indicating perfect structural match.

4.5.2 Physical Consistency Metrics

Mass Conservation Error. For sinusoidal IC where $M_0 \approx 0$:

$$E_{\text{mass}}(t) = |M(t) - M_0| \tag{4.53}$$

For non-zero initial mass:

$$E_{\text{mass}}^{\text{rel}}(t) = \frac{|M(t) - M_0|}{|M_0| + \varepsilon} \tag{4.54}$$

Energy Dissipation Consistency. Verify that predicted energy decreases:

$$\text{Monotonicity} = \frac{1}{T-1} \sum_{t=1}^{T-1} \mathbb{1}[E(t+1) \leqslant E(t) + \varepsilon_{\text{tol}}] \tag{4.55}$$

Perfect monotonicity gives 1.0; violations reduce this score.

4.5.3 Correlation and Spectral Metrics

Pearson Correlation per Timestep.

$$\rho(t) = \frac{\mathrm{Cov}(\hat{u}^t, u^t)}{\sigma_{\hat{u}^t} \sigma_{u^t}} \tag{4.56}$$

High correlation ($\rho \approx 1$) indicates the model captures spatial patterns accurately.

Energy Spectrum Error. Fourier analysis reveals whether the model captures correct wavenumber content:

$$\hat{U}^t(k) = \mathrm{FFT}(u^t), \quad S(k) = |\hat{U}(k)|^2 \tag{4.57}$$

Spectrum error:

$$E_{\text{spec}} = \frac{1}{K} \sum_{k=1}^{K/2} |\log_{10} S_{\text{pred}}(k) - \log_{10} S_{\text{true}}(k)| \tag{4.58}$$

4.6 Implementation Details

4.6.1 Software and Hardware

Framework. All models are implemented in PyTorch 2.0+ with CUDA support for GPU acceleration.

Hardware. Training is performed on:

- GPU: NVIDIA A100 (40GB) or equivalent
- Training time: ~30 minutes for 50 epochs
- Inference time: ~10ms per rollout step

4.6.2 Reproducibility

To ensure reproducibility:

- Random seeds fixed: PyTorch, NumPy, Python (`seed=42`)
- Deterministic algorithms enabled where available
- Complete hyperparameter logging
- Model checkpoints saved every 10 epochs

4.6.3 Code Organization

Our implementation follows modular design:

```
burgers_solver/
|-- data_generation.py      # Finite difference solver
|-- dataset.py              # PyTorch Dataset/DataLoader
|-- model.py                # Neural architecture
|   |-- CausalTemporalAttention
|   +-- ImprovedBurgersNet
|-- training.py             # Training loop
|   |-- Loss functions
|   +-- Curriculum learning
|-- metrics.py              # Evaluation metrics
+-- visualization.py        # Plotting utilities
```

The complete source code and pre-trained models are available in our GitHub repository[1].

4.7 Summary

This chapter has presented our comprehensive methodology for neural PDE solving:

1. **High-quality data generation** using stable finite difference methods with upwind advection and central difference diffusion
2. **Novel architecture** combining causal temporal attention with convolutional decoding for effective spatiotemporal modeling

---

[1] https://github.com/Samuel-Chapuis/ML_Differential_Solver

3. **Physics-informed training** via energy dissipation constraints and gradient matching losses
4. **Curriculum learning** with scheduled rollout depth and teacher forcing for improved stability
5. **Comprehensive evaluation** across accuracy, physical consistency, and spectral metrics

The next chapter presents experimental results demonstrating the effectiveness of these design choices and comparing performance against baseline methods.

Transition to Chapter 5. Having established our complete methodological framework, we now turn to empirical validation. Chapter **??** presents comprehensive experiments evaluating our approach across multiple metrics, analyzing the contribution of each architectural component through ablation studies, and comparing performance against established baseline methods.

Chapter 5

Approaches for Neural PDE Simulators

5.1 Approach 1: Samuel's Model

5.1.1 Model Architecture

5.1.2

5.2 Approach 2: Causal Temporal Convolutional Attention Network

5.2.1 Model Architecture

The Causal 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. At each time step, the model receives a history window u_{history} of shape (B, W, P)—where B is the batch size, W is the history length (e.g., 20), and P is the number of spatial grid points—and outputs the next field u_{t+1} of shape $(B, 1, P)$. The model operates as a one-step neural PDE surrogate, repeatedly applied in an autoregressive rollout to generate full trajectories.

The architecture 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.

The full data flow for a single prediction step is illustrated in Figure 5.1.

5.3 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.

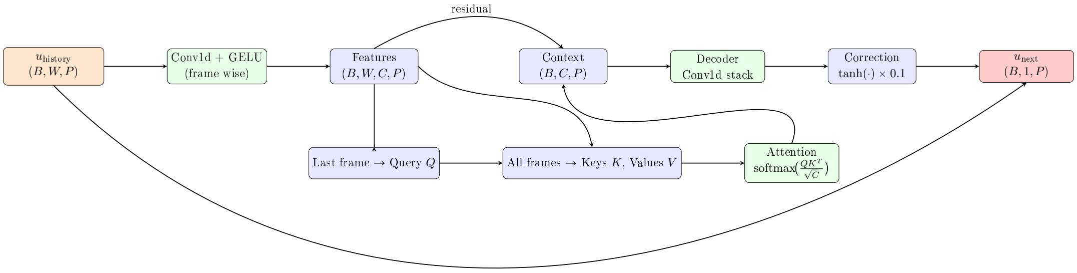
residual
u_{history} (B, W, P)
Conv1d + GELU (frame wise)
Features (B, W, C, P)
Context (B, C, P)
Decoder Conv1d stack
Correction $\tanh(\cdot) \times 0.1$
u_{next} $(B, 1, P)$
Last frame → Query Q
All frames → Keys K, Values V
Attention $\text{softmax}(\frac{QK^T}{\sqrt{C}})$

Figure 5.1: Data flow through the Causal Temporal Attention Network during one prediction step.

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.

5.4 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.

5.5 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 ← $[\text{current\_window}[:, 1 :, :], \hat{u}]$ (with occasional teacher forcing).
3. Average loss over rollout, backpropagate through unrolled computation graph, apply gradient clipping, and optimize with AdamW + cosine annealing.

5.6 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.

Chapter 6

Conclusion and Perspectives

A summary of your work. More focused on the results

The limitations of the work -> which perspectives/clues to deal with limitations, to improve your work

the last paragraph must be dedicated to the work in team

6.1 Gl remarks

GENERAL :

Each table, figure must be cited and explained in the text.

The references must be complete

Each chapter must start with a paragraph to introduce its content (no need to have a separated for that), except the introduction and the conclusion. In the same manner each chapter must finish with a paragraph to conclude and to make a link with the next one, except the introduction and the conclusion.

6.2 Gl remarks about the presentation

The slides must be numbered

The presentation follows more and less the structure of the report

No too much blabla about the the gl context you need to define the objectives of the project (with examples if possible) ...

Then how your work fits into existing works (some main related works), the overall pipeline, your main contributions in this pipeline

also your main contributions in terms of implementation

the main results

Conclusions and next ...

Then the overall

Appendix A

Appendix

The progress draft must be included in the appendix

Bibliography

Abstract:

Keywords: