Neural Implicit Flow: A Comparative Study

Implementation Approaches and Performance Analysis

Christian Beneke

10.02.2025

Outline

Problem Space & Motivation

- High-dimensional spatio-temporal dynamics are computationally challenging
- Traditional methods face limitations:
  - Fixed mesh structures (SVD, CAE)
  - Poor scaling with high-dimensional data
  - Limited ability to capture nonlinear dynamics
- Real-world applications require:
  - Adaptive mesh handling
  - Efficient dimensionality reduction
  - Real-time processing capabilities

Key Challenges in Current Approaches

- Singular Value Decomposition (SVD)
  - Limited to fixed mesh structures
  - Poor scaling with dimensionality
- Convolutional Autoencoders (CAE)
  - Requires uniform grid resolution
  - Struggles with adaptive meshing
- Common Issues
  - High computational overhead
  - Limited expressiveness for complex dynamics
  - Poor generalization to new scenarios

Core Architecture

Two specialized networks:

- ShapeNet: Spatial complexity
- ParameterNet: Temporal evolution

Key innovations:

- Mesh-agnostic representation
- Efficient parameter sharing
- Adaptive complexity handling

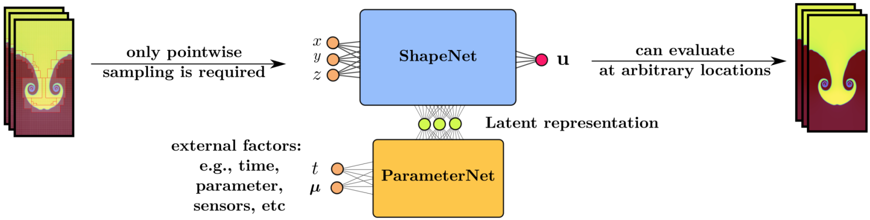


Figure: NIF Architecture

Mathematical Formulation

- Core mapping function:

$$f_\theta : (\mathbf{x}, \mathbf{t}) \mapsto u(\mathbf{x}, \mathbf{t})$$

  where:
  - $\mathbf{x} \in \mathbb{R}^d$: Spatial coordinate
  - $\mathbf{t} \in \mathbb{R}^p$: Temporal/parametric input
  - u: Field value

- Hypernetwork decomposition:

$$f_\theta(\mathbf{x}, \mathbf{t}) = \text{ShapeNet}_{\text{ParameterNet}_{\theta_p}(\mathbf{t})}(\mathbf{x})$$

Implementation Overview

- Three distinct implementations:
  - Upstream Reference (TensorFlow)
  - TensorFlow Functional API
  - PyTorch Implementation
- Key considerations:
  - Framework-specific optimizations
  - Code maintainability
  - Development ergonomics

Upstream Reference Implementation

- Based on original paper implementation
- Key modifications:
  - Migration to current TensorFlow version
  - Various bug fixes needed to run the provided examples
- Implementation details:
  - Direct TensorFlow.Keras Model inheritance
  - Static layer definitions with fixed weights
  - Manual parameter mapping and weight updates

Upstream Reference Architecture

- ParameterNet:
  - Dense (30) $\rightarrow$ 60 params
  - 2x MLP ShortCut (30) $\rightarrow$ 930 each
  - Bottleneck (1) $\rightarrow$ 31 params
  - Output (1951) $\rightarrow$ 3,902 params
- Total: 5,853 params (22.86 KB)
- ShapeNet:
  - Parameters generated by ParameterNet
  - Internal forward pass in a single function call

TensorFlow Functional API Implementation

- Complete architectural redesign
- Key features:
  - Utilises TensorFlow's functional API
  - Custom SIREN kernel initialization
  - Dynamic layer generation
- Optimizations:
  - Automatic shape inference and building
  - TF Function decoration for performance
  - Full ParameterNet output forwarded to all ShapeNet layers

TensorFlow Functional API Architecture

- ParameterNet:
  - Dense (30) $\rightarrow$ 60 params
  - 2x Shortcut (30) $\rightarrow$ 930 each
  - Bottleneck (1) $\rightarrow$ 31 params
  - Output (1951) $\rightarrow$ 3,902 params
  - Total: 5,853 params (22.86 KB)
- ShapeNet:
  - 4x HyperDense layers
  - Parameters generated by ParameterNet

PyTorch Implementation

- Core Features:
  - Mixed precision training support (FP16/BF16)
  - Custom StaticDense and StaticSIREN layers
  - Gradient scaling and automatic mixed precision (AMP)
- Implementation Details:
  - Custom AdaBelief optimizer with gradient centralization
  - Configurable learning rate scheduler with warmup
  - Dynamic device handling (CPU/GPU) with dtype management
- Network Components:
  - ResNet and Shortcut layer implementations
  - Flexible activation function mapping
  - Parameter sharing through static layers

Common Setup

- Implemented base class for common functionality
- Implemented two of the given example cases
- Made optimiser configurable (Adam, AdaBelief)
- Comprehensive training logger with visualization

Low Frequency Wave

- Domain: $x \in [0, 1]$, $t \in [0, 100]$
- 200 spatial points, 10 timesteps
- Simple periodic traveling wave
- $u(x, t) = \exp(-1000(x - x_0 - ct)^2)$

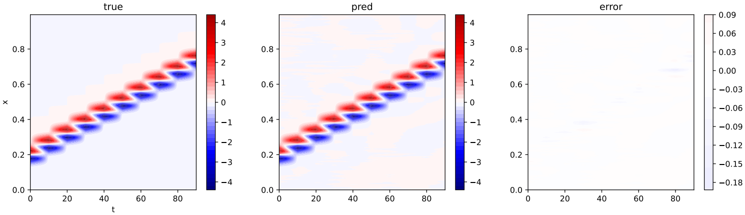


Figure: Low Frequency Example

Test Cases

High Frequency Wave

- Same spatial/temporal domain
- Added frequency $\omega = 400$
- Complex oscillatory pattern
- $u(x,t) = \exp(-1000(x - x_0 - ct)^2)\sin(\omega(x - x_0 - ct))$

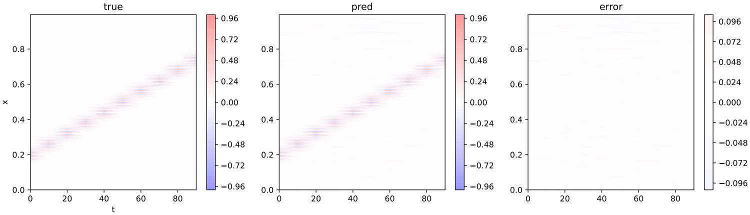


Figure: High Frequency Example

Network Architectures

ParameterNet

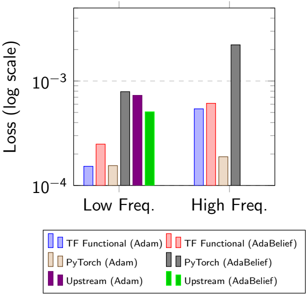
- Dense input layer
- Two hidden layers (30 units)
- Single-unit bottleneck
- Adaptive output layer

ShapeNet Variants

- Shortcut (Low frequency)
  - Skip connections
  - Swish activation
- SIREN (High frequency)
  - Sinusoidal activation
  - $\omega_0 = 30$ scaling

Performance Overview

- Low Frequency Best Result:
  - Func. API: 1.526e-04
  - PyTorch: 1.549e-04
  - Upstream: 5.073e-04
- High Frequency Best Result:
  - PyTorch: 1.884e-04
  - Func. API: 5.415e-04
  - Upstream: N/A



Processing Speed Comparison

Implementation	Low Freq.	High Freq.
TF Functional	11.5-17K pts/s	11.5-16K pts/s
PyTorch	12-15K pts/s	6-16K pts/s
Upstream	12-17K pts/s	-

Table: Processing Speed (points per second)

Optimizer Impact

- Adam vs AdaBelief comparison:
  - TF Functional API: 62.5% improvement
  - PyTorch: 5.10x improvement (low freq.)
  - PyTorch: 11.77x improvement (high freq.)
- Key findings:
  - Adam: More stable convergence
  - Better final loss values
  - Consistent across implementations

Visual Results - Low Frequency

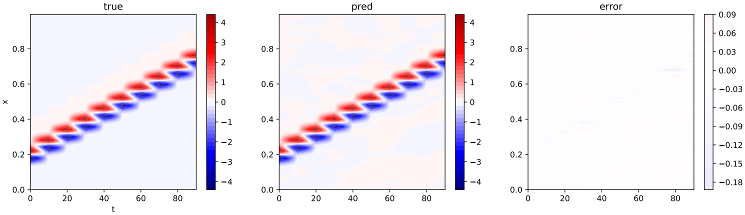


Figure: Low Frequency Predictions (TF Functional API)

Visual Results - High Frequency

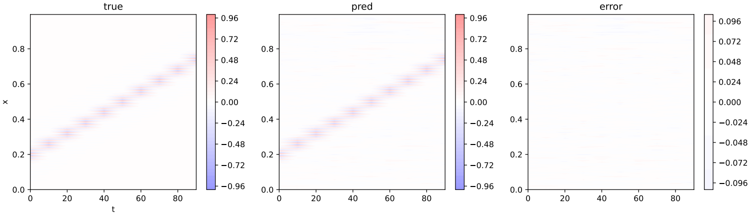


Figure: High Frequency Predictions (TF Functional API)

Key Findings

- Implementation Trade-offs:
  - Modern implementations excel at high-frequency cases
  - TF Functional API: Most consistent performance
  - PyTorch: Best high-frequency accuracy
- Practical Implications:
  - All implementations achieve production-ready speed
  - Adam optimizer consistently outperforms AdaBelief
  - Framework choice impacts development experience

Planned vs Achieved Goals

- ✓ Port from TensorFlow to PyTorch
- \+ Add TensorFlow Functional API implementation
- ✓ Implement using upstream SIREN
- ✓ Improve parallelism in training phase
- × Compare to other HyperNetworks / LoRA / etc
- ✓ Compare existing results with own implementation
- × Compare performance with very hard example (e.g. weather prediction)

Thank you for your attention!

Questions?