

Neural Implicit Flow: A Comparative Study

Implementation Approaches and Performance Analysis

Christian Beneke

10.02.2025

Outline

- 1 Introduction
- 2 Neural Implicit Flow Framework
- 3 Implementation Approaches
- 4 Experimental Setup
- 5 Results and Analysis
- 6 Conclusions

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

Core Architecture

Two specialized networks:

- ShapeNet: Spatial complexity
- ParameterNet: Temporal evolution

Key innovations:

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

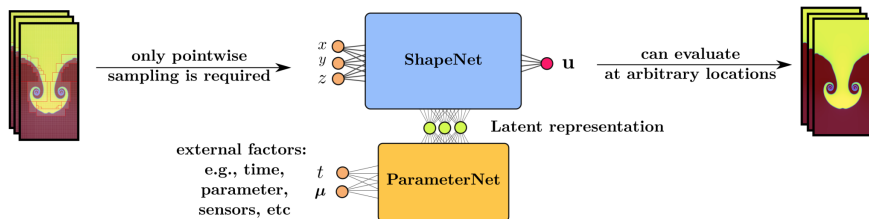


Figure: NIF Architecture

- Core mapping function:

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

where:

- $\mathbf{x} \in \mathbb{R}^d, d \in \mathbb{N}$: Spatial coordinate
- $\mathbf{t} \in \mathbb{R}^p, p \in \mathbb{N}$: Temporal/parametric input
- 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

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

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
 - Internal forward pass in a single function call

Upstream Reference Architecture

Layer (type)	Output Shape	Param #
first_dense_pnet (Dense)	(None, 30)	60
hidden_mlpshortcut_pnet_0 (MLP_SimpleShortCut)	(None, 30)	930
hidden_mlpshortcut_pnet_1 (MLP_SimpleShortCut)	(None, 30)	930
bottleneck_pnet (Dense)	(None, 1)	31
last_pnet (Dense)	(None, 1951)	3,902

Total params: 5,853 (22.86 KB)

Figure: Upstream Reference Model Summary

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

Layer (type)	Output Shape	Param #
first_pnet (Dense)	(None, 30)	60
hidden_pnet_0 (Shortcut)	(None, 30)	930
hidden_pnet_1 (Shortcut)	(None, 30)	930
bottleneck_pnet (Dense)	(None, 1)	31
last_pnet (Dense)	(None, 1951)	3,902
first_snet (HyperDense)	(None, 30)	0
hidden_snet_0 (HyperDense)	(None, 30)	0
hidden_snet_1 (HyperDense)	(None, 30)	0
last_snet (HyperDense)	(None, 1)	0

Total params: 5,853 (22.86 KB)

Figure: TensorFlow Functional API Model Summary

- 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

Low Frequency Traveling Wave

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

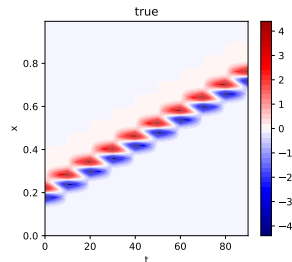


Figure: Low Frequency Example

High Frequency Traveling Wave

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

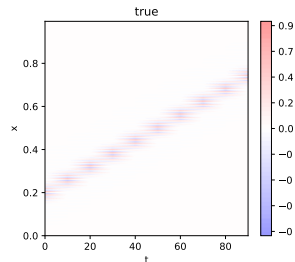


Figure: High Frequency Example

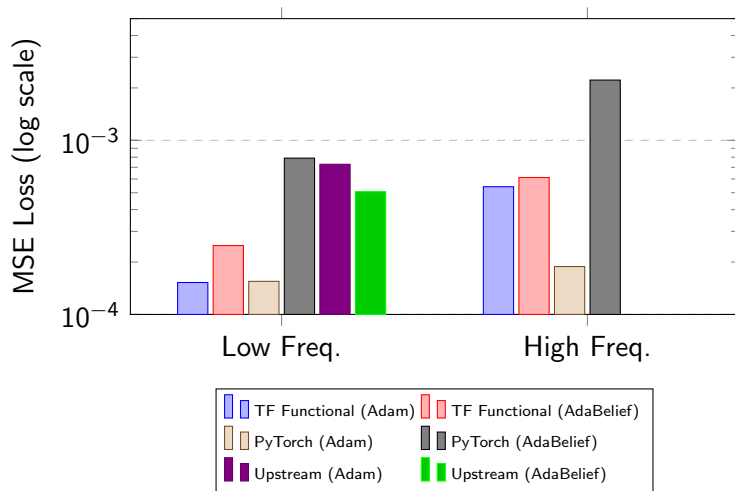
ParameterNet

- Dense input layer
- Two hidden layers (30 units)
- Bottleneck layer (1 unit)
- Adaptive output layer

ShapeNet Variants

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

Performance Overview



- 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

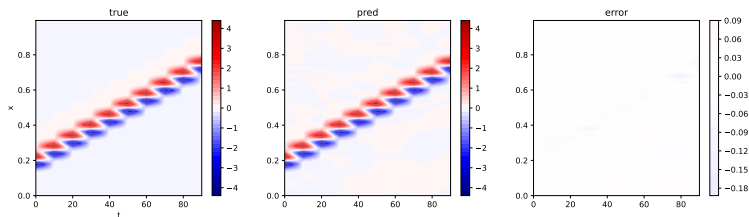


Figure: Low Frequency Predictions (TF Functional API)

Visual Results - High Frequency

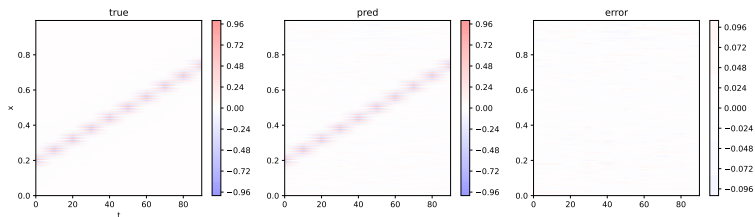


Figure: High Frequency Predictions (TF Functional API)

Key Findings

- Implementation Trade-offs:
 - Modern implementations have consistent performance
 - PyTorch: Best high-frequency accuracy
- Practical Implications:
 - All implementations achieve production-ready speed
 - Adam optimizer consistently outperforms AdaBelief

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?