MECHANISTIC INTERPRETABILITY FOR PHYSICS-INFORMED NEURAL NETWORKS

A Comprehensive Research Project Document

Reverse-Engineering the Computational Mechanisms
Learned by Neural Networks for Solving Differential Equations

Duration: 8 Weeks
Type: Research Project
Target: AI Research Labs (Anthropic, OpenAI, DeepMind)

Version 1.0
February 2026

Table of Contents

1. Executive Summary

This research project aims to apply mechanistic interpretability techniques, originally developed for understanding large language models at Anthropic and other research labs, to Physics-Informed Neural Networks (PINNs). The fundamental question we seek to answer is: ***What computational mechanisms do neural networks develop when learning to solve differential equations, and how do these mechanisms relate to classical numerical methods?***

PINNs have achieved remarkable success in solving partial differential equations across physics, engineering, and applied mathematics. However, despite thousands of papers demonstrating their empirical effectiveness, we have almost no understanding of the internal algorithms these networks develop. Do they learn finite difference approximations? Spectral methods? Or perhaps entirely novel computational strategies that could advance numerical analysis itself?

This project will systematically dissect PINN architectures using activation patching, probing classifiers, and circuit analysis to recover interpretable computational graphs. The expected outputs include: (1) identification of learned numerical schemes within network weights, (2) mechanistic explanations for known PINN failure modes such as spectral bias, (3) principled architectural recommendations based on discovered computational bottlenecks, and (4) potential discovery of novel numerical algorithms implemented by trained networks.

The project duration is 8 weeks, with a focused implementation track that can deliver meaningful results in as few as 4-6 weeks for a single dedicated researcher. The work directly addresses Anthropic's research agenda on interpretability while showcasing the unique value of combining physics expertise with AI safety research.

2. Problem Definition and Motivation

2.1 The Interpretability Gap in Scientific Machine Learning

Physics-Informed Neural Networks represent one of the most successful applications of deep learning to scientific computing. Introduced by Raissi, Perdikaris, and Karniadakis in 2019, PINNs solve differential equations by incorporating physical laws directly into the neural network's loss function. Rather than requiring labeled input-output pairs, PINNs learn to satisfy the governing equations at collocation points throughout the domain, essentially turning the PDE into a regularization term.

Despite their success, a fundamental question remains unanswered: How do PINNs actually compute their solutions? When a PINN is trained to solve the heat equation, what algorithm does it implement? Traditional numerical methods for PDEs are well-understood: finite differences approximate derivatives using local polynomial interpolation, finite elements project the solution onto basis functions, and spectral methods expand solutions in global eigenfunctions. Each method has known convergence properties, error bounds, and computational characteristics.

Neural networks, in contrast, remain black boxes. We can measure their accuracy on test points, but we cannot explain why a particular network succeeds or fails. This opacity has several practical consequences:

1. Trust and Deployment: Engineers cannot confidently deploy PINNs in safety-critical applications (nuclear reactors, aircraft design) without understanding their failure modes.
2. Architecture Design: Without understanding what computations networks perform, architecture choices remain largely heuristic.
3. Debugging Failures: When PINNs fail to converge or produce inaccurate solutions, practitioners lack tools to diagnose the computational bottleneck.
4. Scientific Discovery: If networks develop novel algorithms, we have no way to extract and formalize this knowledge.

2.2 The Promise of Mechanistic Interpretability

Mechanistic interpretability is a research paradigm that aims to reverse-engineer the algorithms implemented by neural networks by analyzing their weights and activations. Unlike post-hoc explanation methods (SHAP, LIME, attention visualization) that describe what inputs influenced an output, mechanistic interpretability seeks to understand how the network computes its outputs.

Pioneering work at Anthropic and elsewhere has demonstrated that language models develop interpretable circuits for specific tasks. Elhage et al. (2021) identified 'induction heads' that implement in-context learning through attention patterns. Nanda et al. (2023) showed that GPT-2 develops interpretable algorithms for modular addition. These discoveries were enabled by systematic techniques: activation patching to identify causal components, probing classifiers to detect internal representations, and circuit analysis to map computational graphs.

Critically, this methodology has never been applied to scientific machine learning. PINNs present an ideal testbed for mechanistic interpretability for a unique reason: we know the ground-truth algorithm space. Centuries of numerical analysis have produced a rich taxonomy of methods for solving differential equations. If a PINN learns to approximate derivatives, we can precisely characterize whether it implements forward differences, backward differences, central differences, or higher-order schemes. This verifiability is absent in language model interpretability, where the 'correct' algorithm for language generation is undefined.

2.3 Why This Matters Now

Several converging trends make this research timely:

- Scaling of AI for Science: Projects like AlphaFold, GraphCast, and FourCastNet demonstrate that AI is becoming central to scientific research. Understanding these systems is essential for scientific integrity.
- Maturation of Interpretability Tools: The interpretability community has developed increasingly sophisticated techniques (automated circuit discovery, sparse autoencoders) ready for application to new domains.
- Known PINN Failure Modes: The PINN literature has documented several systematic failures (spectral bias, causality violations, gradient imbalance) that remain unexplained. Mechanistic analysis could provide the first causal explanations.
- Regulatory Pressure: As AI systems are deployed in safety-critical applications, regulators increasingly demand explanations. Mechanistic understanding provides a principled foundation for such explanations.

3. Theoretical Background

3.1 Physics-Informed Neural Networks: Formulation

Consider a general partial differential equation of the form:

N[u](x,t) = f(x,t), for (x,t) in domain D

where N is a differential operator (potentially nonlinear), u(x,t) is the unknown solution, and f(x,t) is a source term. The solution must also satisfy boundary conditions B[u] = g on the boundary of D, and initial conditions u(x,0) = h(x).

A PINN approximates u(x,t) with a neural network u_theta(x,t) and trains by minimizing:

*L = w_pde * L_pde + w_bc * L_bc + w_ic * L_ic*

where L_pde measures the PDE residual at interior collocation points, L_bc measures boundary condition violations, L_ic measures initial condition errors, and w_* are weighting coefficients. The derivatives appearing in N[u] are computed via automatic differentiation through the neural network.

3.2 Classical Numerical Methods as Reference Algorithms

To interpret what algorithms PINNs learn, we need a taxonomy of reference methods. The primary classes are:

Finite Difference Methods

These approximate derivatives using values at discrete grid points. The first derivative can be approximated by forward differences (u'(x) approximately equal to [u(x+h) - u(x)]/h), backward differences ([u(x) - u(x-h)]/h), or central differences ([u(x+h) - u(x-h)]/(2h)). Higher-order schemes use more points for improved accuracy. A trained PINN that computes derivatives similar to central differences would exhibit specific patterns: weights connecting to symmetric neighborhoods, activation patterns that approximate linear combinations.

Spectral Methods

These expand the solution in global basis functions (typically Fourier modes or Chebyshev polynomials). Derivatives become multiplication in the spectral domain. Interestingly, Fourier Networks (a PINN variant) explicitly embed Fourier features, potentially learning spectral-like algorithms. Our analysis will examine whether standard MLPs also develop spectral representations.

Finite Element Methods

These decompose the domain into elements and approximate the solution as a weighted sum of local basis functions. The Galerkin formulation converts the PDE into a system of algebraic equations. A PINN learning finite-element-like computations would show localized receptive fields and hierarchical aggregation.

3.3 Mechanistic Interpretability Techniques

The following techniques, developed primarily for language model analysis, will be adapted for PINNs:

Activation Patching

This technique identifies causal components by replacing activations from one input with activations from another and measuring the effect on the output. For PINNs, we will patch activations between: (a) different spatial locations to identify receptive fields, (b) different time steps to understand temporal integration, (c) different PDE regimes (smooth vs. discontinuous) to find specialized circuits.

Probing Classifiers

Linear probes trained on intermediate activations reveal what information the network has computed at each layer. We will train probes to predict: (a) numerical derivatives (du/dx, du/dt, d2u/dx2) from hidden layer activations, (b) local Taylor coefficients, (c) spectral coefficients (if the network uses Fourier-like representations), (d) physical quantities (energy, momentum) that might be implicitly conserved.

Circuit Analysis

Following Elhage et al., we decompose the network into interpretable subgraphs (circuits) that perform specific computations. For PINNs, candidate circuits include: derivative computation circuits, boundary condition enforcement circuits, and nonlinearity handling circuits for nonlinear PDEs.

Automated Circuit Discovery (ACDC)

Conmy et al. (2023) developed ACDC, which automatically identifies minimal circuits responsible for specific behaviors using iterative edge pruning. We will adapt this for PINNs by defining behavior metrics specific to PDE solving: derivative accuracy, boundary condition satisfaction, and conservation law preservation.

4. Research Objectives and Hypotheses

4.1 Primary Research Questions

This project addresses the following fundamental questions:

5. Algorithm Identification: Do PINNs learn recognizable numerical algorithms, and if so, which ones? Specifically, do they implement finite differences, spectral methods, or hybrid approaches?
6. Derivative Computation: How do PINNs compute the derivatives required for the PDE residual? Are derivatives computed explicitly in intermediate layers, or do they emerge only in the final output?
7. Failure Mode Explanation: Can mechanistic analysis explain known PINN pathologies, particularly spectral bias (difficulty learning high-frequency components)?
8. Novel Algorithms: Do PINNs develop computational strategies that differ from classical numerical methods? If so, can these be extracted and formalized?

4.2 Working Hypotheses

Based on prior knowledge of both neural networks and numerical methods, we propose the following testable hypotheses:

Hypothesis 1: Local Derivative Circuits

Early layers of PINNs develop circuits that approximate local derivatives using weighted combinations of nearby input coordinates, functionally equivalent to finite difference stencils. Evidence supporting this would include: weight patterns showing symmetric or antisymmetric structure (for even/odd derivatives), activation patching showing locality (distant patches have minimal effect), and probes successfully extracting derivatives from early layers.

Hypothesis 2: Spectral Bias Originates in Initialization

The well-documented spectral bias of PINNs (preference for low-frequency solutions) results from the smoothness prior induced by random initialization combined with gradient-based learning dynamics. Mechanistically, we hypothesize that: (a) early training develops low-frequency components first, (b) high-frequency circuits require larger weight magnitudes that training struggles to reach, and (c) Fourier feature networks avoid this by injecting high-frequency information at the input.

Hypothesis 3: Boundary Enforcement via Specialized Subnetworks

PINNs develop distinct computational pathways for interior PDE residual minimization versus boundary condition enforcement. These pathways may interfere, explaining observed training instabilities. We expect to find neurons or circuits that activate selectively near boundaries.

4.3 Success Criteria

The project will be considered successful if we achieve at least two of the following:

- Identify at least one interpretable circuit corresponding to a known numerical operation (e.g., central difference derivative)
- Provide a mechanistic explanation for spectral bias that predicts intervention outcomes (e.g., specific weight modifications that alleviate bias)
- Train probing classifiers that extract derivatives from intermediate layers with R-squared greater than 0.9
- Identify architectural modifications that improve PINN performance, motivated by mechanistic findings

5. Methodology and Technical Approach

5.1 Experimental Design Overview

The project follows a systematic progression from simple to complex PDEs, using a consistent interpretability toolkit at each stage:

Stage	PDE System	Phenomena	Interpretability Focus
1	Poisson Equation	Elliptic, steady-state	Derivative computation, boundary handling
2	Heat Equation	Parabolic, diffusion	Temporal integration, smoothing
3	Burgers Equation	Nonlinear, shock formation	Nonlinearity handling, multi-scale
4	Helmholtz Equation	Wave propagation, oscillatory	Spectral bias, frequency representation

5.2 PINN Architectures Under Study

We will analyze three architecture families to understand how architectural choices affect learned algorithms:

5.2.1 Standard Multi-Layer Perceptron (MLP)

Configuration: 4-8 hidden layers, 50-200 neurons per layer, tanh activation. This is the canonical PINN architecture from the original paper. Analysis will focus on weight structure (do early layers show finite-difference-like patterns?) and activation flow (where do derivatives emerge?).

5.2.2 Modified Fourier Network (MFN)

Configuration: Fourier feature embedding followed by MLP. The input (x,t) is mapped to sin and cos features at various frequencies before processing. This architecture was introduced to combat spectral bias. Analysis will examine how the Fourier embedding changes internal representations and whether the network learns to compose these features spectrally.

5.2.3 Attention-Enhanced PINN

Configuration: Self-attention layers interspersed with MLPs. Attention mechanisms can aggregate information non-locally, potentially enabling finite-element-like computations. Analysis will use attention pattern visualization alongside standard interpretability tools.

5.3 Interpretability Toolkit Implementation

5.3.1 Activation Extraction and Storage

Using PyTorch forward hooks, we will extract activations at every layer for analysis. Storage will use HDF5 for efficient access to large activation datasets. For each trained PINN, we will store activations evaluated on a dense grid covering the PDE domain.

5.3.2 Probing Classifier Suite

Linear probes (single-layer neural networks) will be trained to predict target quantities from intermediate activations. Targets include: first and second spatial derivatives, temporal derivatives, boundary distance, spectral coefficients, and physical quantities (when applicable). Probe accuracy (R-squared, MSE) indicates whether the information is linearly accessible at that layer.

5.3.3 Activation Patching Framework

We will implement activation patching with spatial awareness. Given two input points x_source and x_target, patching replaces activations at x_target with those from x_source and measures the output change. By sweeping over source-target pairs, we can map causal influence and identify receptive fields.

5.3.4 Circuit Discovery Pipeline

Adapting ACDC for PINNs, we will iteratively prune edges in the computational graph while preserving specific behaviors (derivative accuracy, boundary satisfaction). The minimal circuit retaining behavior reveals the essential computation.

5.4 Tools and Frameworks

Primary implementation will use PyTorch for neural network training and hook-based activation extraction. We will adapt TransformerLens concepts for MLP analysis. Additional tools include: NumPy and SciPy for numerical analysis, Matplotlib and Plotly for visualization, Weights and Biases for experiment tracking, and pytest for code validation.

6. Week-by-Week Implementation Plan

Week 1: Infrastructure and Baseline PINNs

Objectives

- Establish development environment with version control
- Implement PINN training pipeline for Poisson equation
- Validate training achieves expected accuracy

Technical Tasks

Days 1-2: Set up PyTorch environment, configure logging with Weights and Biases, create GitHub repository with proper structure (src/, tests/, notebooks/, configs/). Implement modular PINN architecture with configurable depth, width, and activation functions.

Days 3-4: Implement Poisson equation (Laplacian(u) = f) with Dirichlet boundary conditions. Use automatic differentiation for derivative computation. Train on manufactured solution with known analytical form for validation.

Days 5-7: Implement activation extraction using PyTorch hooks. Store activations for 10,000+ evaluation points. Create visualization utilities for activation patterns. Document code and commit to GitHub.

Deliverables

- GitHub repository with clean project structure
- Working PINN achieving less than 1% relative L2 error on Poisson
- Activation extraction pipeline saving to HDF5

Week 2: Probing Classifier Development

Objectives

- Implement probing classifier framework
- Train probes to detect derivative information at each layer
- Establish baseline: where do derivatives become linearly accessible?

Technical Tasks

Days 1-3: Implement linear probing framework. For each layer's activations h_l, train a linear model to predict target quantities. Compute ground-truth derivatives using high-precision finite differences on the analytical solution. Evaluate probe accuracy using R-squared and MSE.

Days 4-5: Train probes for first derivatives (du/dx, du/dy), second derivatives (d2u/dx2, d2u/dy2), and Laplacian. Generate layer-by-layer accuracy plots showing information emergence.

Days 6-7: Analyze probe weights. For first-layer probes, visualize weight patterns to identify potential finite-difference-like structure. Document findings in notebook with reproducible analysis.

Deliverables

- Probing classifier codebase with unit tests
- Layer-wise derivative accessibility analysis
- Initial hypothesis about derivative computation mechanism

Week 3: Activation Patching Implementation

Objectives

- Implement spatially-aware activation patching
- Map causal influence between spatial locations
- Identify effective receptive fields at each layer

Technical Tasks

Days 1-3: Implement activation patching for MLPs. For input point x_target, replace hidden activations with those from x_source and measure output change. Create efficient batched implementation to sweep over many source-target pairs.

Days 4-5: Generate causal influence maps. For each target point, visualize which source points most affect the output. Compute effective receptive field radius at each layer. Compare with theoretical receptive fields of finite difference stencils.

Days 6-7: Analyze boundary effects. Does activation patching reveal special treatment of boundary regions? Document observations regarding boundary-interior interaction.

Deliverables

- Activation patching framework with documentation
- Receptive field analysis visualizations
- Evidence regarding locality of learned algorithms

Week 4: Circuit Discovery for Poisson PINN

Objectives

- Identify minimal circuits for derivative computation
- Map circuit structure to known numerical algorithms
- Validate circuits via ablation experiments

Technical Tasks

Days 1-3: Implement ACDC-style circuit discovery adapted for PINNs. Define behavior metrics: Laplacian accuracy, boundary condition satisfaction. Iteratively prune edges while preserving behavior.

Days 4-5: Analyze discovered circuits. Visualize circuit structure as computational graph. Identify candidate 'derivative neurons' whose removal degrades derivative probing accuracy. Examine weight patterns in identified circuits.

Days 6-7: Compare circuit computations to finite difference stencils. If circuit weights form patterns like [-1, 2, -1] (second derivative), document this correspondence. Prepare first technical report summarizing Poisson findings.

Deliverables

- Circuit discovery implementation
- Identified circuits with interpretable structure
- Technical report on Poisson PINN mechanisms

Week 5: Extension to Time-Dependent PDEs

Objectives

- Train and analyze PINN for Heat equation
- Investigate temporal integration mechanisms
- Compare spatial vs. temporal derivative circuits

Technical Tasks

Days 1-2: Implement Heat equation (du/dt = alpha * Laplacian(u)) PINN. Train on diffusion of initial Gaussian. Validate against analytical solution.

Days 3-4: Apply probing analysis. Compare accuracy of spatial derivative probes vs. temporal derivative probes. Does the network treat space and time symmetrically or differently?

Days 5-7: Activation patching across time. How does information from initial condition propagate? Identify circuits responsible for 'memory' of initial state. Document temporal evolution of internal representations.

Deliverables

- Heat equation PINN with validated accuracy
- Comparative analysis of spatial vs. temporal mechanisms
- Initial condition propagation analysis

Week 6: Nonlinear PDEs and Spectral Bias Investigation

Objectives

- Analyze Burgers equation PINN for nonlinearity handling
- Investigate Helmholtz equation for spectral bias mechanisms
- Propose mechanistic explanation for spectral bias

Technical Tasks

Days 1-3: Train Burgers equation (du/dt + u*du/dx = nu*d2u/dx2) PINN. Choose parameters that produce shock formation. Analyze how network handles the nonlinear convective term u*du/dx. Does it compute u and du/dx separately then multiply, or develop a different strategy?

Days 4-5: Train Helmholtz equation (Laplacian(u) + k^2*u = f) with varying wavenumber k. Document spectral bias: at what k does accuracy degrade? Use probing to track Fourier coefficient representations at each layer.

Days 6-7: Comparative analysis with Modified Fourier Network. Does Fourier embedding change internal representations? Propose mechanistic hypothesis for spectral bias.

Deliverables

- Nonlinearity handling mechanism analysis
- Spectral bias mechanistic hypothesis with supporting evidence
- MFN comparison analysis

Week 7: Synthesis and Intervention Experiments

Objectives

- Synthesize findings across all PDEs into unified framework
- Design and execute intervention experiments to validate hypotheses
- Derive architectural recommendations from mechanistic insights

Technical Tasks

Days 1-3: Design intervention experiments. If we hypothesize that specific neurons compute derivatives, verify by: (a) ablating those neurons and measuring derivative accuracy drop, (b) modifying their weights and predicting output changes. Execute interventions and document results.

Days 4-5: If spectral bias hypothesis involves insufficient high-frequency representation, intervene by: manually adding high-frequency features, modifying initialization to boost high-frequency weights, or adding explicit derivative computation layers. Measure effectiveness.

Days 6-7: Compile architectural recommendations based on findings. For example: 'Add explicit derivative computation layers because MLPs struggle to learn central differences' or 'Use skip connections to preserve spatial locality.'

Deliverables

- Intervention experiment results validating/refuting hypotheses
- Principled architectural recommendations
- Unified mechanistic framework document

Week 8: Documentation and Publication Preparation

Objectives

- Prepare publication-quality documentation
- Create open-source release with tutorials
- Draft paper for submission to interpretability venue

Technical Tasks

Days 1-2: Code cleanup and documentation. Ensure all functions have docstrings, create README with installation and usage instructions, add example notebooks demonstrating key analyses.

Days 3-4: Create publication-quality figures. Generate clear visualizations of: derivative probing accuracy by layer, activation patching receptive fields, identified circuit structures, spectral bias mechanism.

Days 5-7: Draft paper. Target venues: NeurIPS (Interpretability track), ICML (AI for Science), or dedicated interpretability workshops. Structure: Introduction, Related Work, Methods, Experiments, Results, Discussion.

Deliverables

- Open-source GitHub repository with full documentation
- Publication-ready figures and tables
- Paper draft ready for submission

7. Expected Deliverables

7.1 Code Repository

A well-documented, open-source GitHub repository containing:

- Modular PINN training framework supporting multiple PDEs
- Interpretability toolkit: probing classifiers, activation patching, circuit discovery
- Pre-trained models and saved activations for reproducibility
- Tutorial notebooks demonstrating key analyses
- Comprehensive test suite with greater than 80% code coverage

7.2 Research Outputs

- Technical report documenting mechanistic findings for each PDE
- Unified framework relating PINN computations to classical numerical methods
- Mechanistic explanation for spectral bias (if successful)
- Architectural recommendations with empirical validation

7.3 Publication

Draft paper suitable for submission to top-tier venues. Target: 8-10 pages plus appendix, following NeurIPS or ICML format.

8. Risk Assessment and Mitigation

8.1 Technical Risks

Risk: Networks may learn uninterpretable computations

Probability: Medium. Mitigation: Negative results (demonstrating that PINNs do NOT learn recognizable algorithms) are still publishable and valuable. Focus on characterizing what IS learned, even if unfamiliar. Use sparse autoencoders if standard probing fails.

Risk: Probing may not reveal linear features

Probability: Medium. Mitigation: Use nonlinear probes (small MLPs) if linear probes fail. Apply sparse autoencoders to find interpretable directions in activation space.

Risk: Circuit discovery may be computationally expensive

Probability: Low. Mitigation: Focus on smaller networks (4 layers, 50 neurons) for detailed analysis. Scale up only for validation.

8.2 Timeline Risks

Risk: Unexpected debugging delays

Probability: Medium. Mitigation: Build in buffer time. Prioritize core analyses (Poisson, probing) over extensions (attention PINNs). Can produce meaningful results with just first 4 weeks.

8.3 Contingency Plan

If the full 8-week plan proves infeasible, prioritize the following minimum viable project:

9. Weeks 1-2: Poisson PINN with probing analysis
10. Weeks 3-4: Activation patching and circuit discovery

This 4-week core still produces a publishable contribution demonstrating novel application of interpretability methods to scientific ML.

9. References

Raissi, M., Perdikaris, P., and Karniadakis, G.E. (2019). 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.

Elhage, N., et al. (2021). A Mathematical Framework for Transformer Circuits. Anthropic.

Nanda, N., et al. (2023). Progress measures for grokking via mechanistic interpretability. ICLR.

Conmy, A., et al. (2023). Towards Automated Circuit Discovery for Mechanistic Interpretability. NeurIPS.

Wang, S., Teng, Y., and Perdikaris, P. (2021). Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM Journal on Scientific Computing, 43(5), A3055-A3081.

Rahaman, N., et al. (2019). On the spectral bias of neural networks. ICML.

Tancik, M., et al. (2020). Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. NeurIPS.

Krishnapriyan, A., et al. (2021). Characterizing possible failure modes in physics-informed neural networks. NeurIPS.

Elhage, N., et al. (2022). Toy Models of Superposition. Anthropic.

Alain, G. and Bengio, Y. (2016). Understanding intermediate layers using linear classifier probes. ICLR Workshop.