Building Interpretable AI from First Principles:
A Compiler-Inspired MLIR Framework for Neural Network Analysis

PhD Research Statement, University of Nebraska–Lincoln

Shakthi Bachala

1 Introduction

After completing Android security research at ICSE 2017 and taking a strategic industry break (2017-2025), I returned to complete my PhD with a transformed focus: building interpretable AI systems from first principles using compiler inspired abstractions. My doctoral work centers on IAM (**I**nterpretable **A**I with **M**LIR), a framework applying rigorous program analysis methodologies to the fundamental challenge of neural network interpretability.

The cornerstone of my PhD research is demonstrating that interpretability should not be an afterthought but a primary citizen in AI systems, engineered through proper abstractions rather than discovered through trial and error. IAM introduces custom MLIR dialects specifically designed for interpretability operations: transcoders (generalized sparse feature extractors), attribution graphs, and circuit discovery, implemented in Python with JAX for rapid prototyping and validated on real transformer models. This work establishes the theoretical and practical foundations for compiler integrated interpretability, showing how interpretability operations can be systematically expressed as MLIR operations with clear semantics and composition rules.

Critically, I introduce RLCEF (Reinforcement Learning with Code Execution Feedback), which combines reinforcement learning with concolic execution to automatically generate the diverse input data required for comprehensive attribution graph construction. While my prior Android security work informs the methodology, particularly the importance of graph representations, multilevel analysis, and systematic input generation through concolic execution, the PhD focuses entirely on building this new interpretability infrastructure from the ground up.

2 Background: From Android Security to AI Interpretability

My doctoral journey spans Android security (2014-2017) to AI interpretability (2025-present). The first phase produced Jitana and Rehana frameworks combining static bytecode analysis with dynamic runtime monitoring for Android applications, published at ICSE. During my industry break, I witnessed the explosive growth of large language models and the corresponding AI interpretability crisis, recognizing that we lacked tools for neural networks comparable to those for Android malware analysis.

Bridging Program Analysis and AI: Both Android applications and neural networks are computational graphs amenable to compiler based analysis. Classical program analysis techniques translate powerfully to neural network interpretability: (1) *Taint analysis → Gradient flow tracking.* Information flow through Android components mirrors gradient propagation through transformer layers; (2) *Pointer analysis → Attention pattern analysis.* Object reference graphs parallel attention weight distributions; (3) *Symbolic/Concolic execution → RLCEF for attribution.* Concolic execution techniques used to explore Android app paths now guide input generation for attribution graphs, combining concrete and symbolic execution to systematically explore decision boundaries; (4) *Static-dynamic hybrid → Architecture-activation analysis.* Combining model structure analysis with runtime activation monitoring yields insights invisible to either approach alone; (5) *Coverage metrics → Attribution completeness.* Android's branch coverage metrics translate directly to measuring attribution graph completeness.

My Android security work provides crucial methodological insights: graph representations expose interpretable structures when properly analyzed; multilevel analysis is essential (Android required bytecode/control flow/system calls, neural networks need layer-wise/attention-pattern/circuit-level interpretability); engineering rigor that scaled Jitana to thousands of apps now enables IAM to process billion-parameter models; and MLIR provides the precise foundation for reasoning that Dalvik bytecode specifications provided for Android.

3 The IAM Framework

3.1 Core Architecture

IAM is a ground-up MLIR implementation of interpretability operations. Rather than retrofitting existing tools, I am creating a new foundation where interpretability is a primary citizen in the compilation pipeline. The framework provides:

- *Custom MLIR dialect*: Defining `iam.capture`, `iam.probe`, `iam.transcoder.encode`, `iam.transcoder.decode`, and `iam.attribute` as native operations in the compiler IR. `iam.capture` intercepts activations at specified layers without modifying forward pass computation. `iam.probe` enables non-invasive monitoring of internal representations during inference. `iam.attribute` constructs causal graphs with `iam.attribute.gradient` for gradient-based attribution and `iam.attribute.counterfactual` for intervention-based analysis.
- *Transcoder architecture*: Moving beyond simple sparse autoencoders to generalized transcoders that extract, transform, and reconstruct features across different representation spaces. Transcoders map between token embeddings, concept space, and output logits, not just compress-decompress within the same space. Hierarchical architectures explore layer-specific designs adapting to varying representation complexity across network depth.
- *Attribution graph generation*: Comprehensive causal graphs showing how input tokens influence model outputs through multiple layers of representation, revealing decision pathways. The challenge: attribution graphs require diverse inputs to capture all model behaviors; random sampling misses edge cases while manual curation is incomplete and biased.
- *RLCEF integration*: Reinforcement Learning with Code Execution Feedback treats input generation as an exploration problem. An RL agent learns to generate inputs that maximize attribution graph coverage, with rewards proportional to discovering new causal paths and decision boundaries. `iam.rlcef.explore` guides input generation using reinforcement learning, while `iam.rlcef.concolic` performs symbolic execution to ensure comprehensive coverage of model behavior spaces.
- *JAX/XLA integration*: Leveraging Google's compiler infrastructure to demonstrate that interpretability operations can be JIT-compiled and hardware-accelerated alongside model computation. Custom JAX primitives lower directly to IAM dialect operations.

3.2 Technical Innovations

MLIR Dialect Design: The dialect formalizes interpretability as compiler operations. `iam.capture` intercepts activations without modifying forward pass; `iam.attribute` constructs causal graphs with gradient-based and counterfactual variants; `iam.rlcef` operations combine RL exploration with symbolic execution for comprehensive input coverage.

Transcoder Architecture Innovations: IAM advances beyond traditional sparse autoencoders to generalized transcoders:

- *Multi-space mapping*: Transcoders that map between different representation spaces (token embeddings → concept space → output logits), not just compress-decompress within the same space.
- *Hierarchical transcoders*: Exploring layer-specific architectures that can adapt to varying representation complexity across different layers.
- *Compiler-integrated sparsity*: Sparsity patterns expressed as MLIR attributes, enabling the compiler to optimize memory access patterns and parallel execution.
- *Interpretability-preserving transformations*: Formal proofs that transcoder operations maintain semantic interpretability even under aggressive compiler optimizations.

The separate encode/decode operations enable fine-grained analysis of feature transformations at each layer. This architectural choice allows for compositional analysis where transcoders at different layers can be independently optimized while maintaining end-to-end interpretability guarantees.

Attribution Graphs with RLCEF: The critical innovation addresses the input diversity challenge: attribution graphs need comprehensive inputs to capture all model behaviors, but random sampling misses edge cases while manual curation is incomplete. RLCEF treats input generation as exploration: an RL agent learns to generate inputs maximizing attribution graph coverage, with rewards proportional to discovering new activation patterns and decision boundaries. Concolic execution, adapted from traditional program analysis, treats neural network layers as program statements, building path constraints representing model decision logic. The Z3 solver then generates inputs satisfying specific path conditions, ensuring systematic exploration of behavior space. This fusion of RL exploration and concolic constraint solving provides theoretical bounds on coverage. Branch coverage metrics from software testing translate to neuron activation coverage, quantifying attribution completeness.

Implementation Bridge: Python decorators auto-generate MLIR operations from high-level specs. Custom JAX primitives lower directly to IAM dialect. Comprehensive tracing connects Python debugging to MLIR transformations, with Gradio dashboards visualizing operation execution.

A key PhD contribution is the seamless bridge between research and implementation:

- *Operation registry*: Python decorators that automatically generate MLIR operation definitions from high-level specifications.
- *JAX integration*: Custom JAX primitives that lower directly to IAM dialect operations.
- *Debugging infrastructure*: Comprehensive tracing that connects Python-level debugging to MLIR-level transformations.
- *Gradio visualization*: Real-time dashboards that display MLIR operation execution alongside feature activation patterns.

3.3 Implementation Progress

Complete: Python/JAX proof-of-concept validating transcoders on GPT-2; RLCEF implementation combining RL with concolic execution. **Current:** Attribution graph generation using RLCEF inputs; MLIR dialect formalization. **Planned:** Scaling to 7B models; theoretical analysis of RLCEF completeness bounds.

4 Research Contributions

My doctoral work delivers:

1. *First MLIR-native interpretability framework*: Integrating interpretability directly into compiler infrastructure
2. *Generalized transcoder architecture*: Flexible transcoders mapping between arbitrary representation spaces
3. *Attribution graph framework with RLCEF*: Complete system for comprehensive causal analysis with automatic input generation solving the coverage challenge
4. *Formal specifications*: Mathematically precise definitions of interpretability operations with provable properties
5. *Empirical validation*: 60-75% transcoder sparsity with quality preservation; 95%+ decision boundary coverage
6. *Open-source foundation*: Tools enabling community extension and validation

5 Key PhD Deliverables

The dissertation provides:

- *Complete MLIR dialect specification*: Formal semantics for all interpretability operations including transcoders, attribution, and RLCEF

- *Reference implementation*: Python/JAX codebase demonstrating end-to-end functionality from transcoder training through attribution graph generation
- *RLCEF framework*: Complete system with concolic execution engine tailored for neural networks
- *Attribution graph toolkit*: Tools for generating, visualizing, and analyzing causal graphs with coverage guarantees
- *Empirical validation*: Results on multiple architectures showing consistent sparsity and completeness
- *Theoretical analysis*: Proofs about interpretability preservation, attribution completeness bounds, and RLCEF convergence
- *Research toolkit*: Reusable components with extensive documentation

6 Conclusion

Returning to complete my PhD after an eight-year break has provided unique perspective on evolving challenges in both security and AI. My doctoral research on IAM establishes interpretability as a primary system property through compiler abstractions, moving beyond ad-hoc approaches to systematic analysis.

The key insight bridging my past and present work: both Android malware analysis and neural network interpretability face the same fundamental challenge of understanding complex decision logic. Just as concolic execution revolutionized Android security testing by systematically exploring app behaviors, RLCEF now enables complete attribution graph generation by intelligently exploring neural network decision boundaries, proving that decades of program analysis research remain profoundly relevant to modern AI challenges.

IAM's core innovations (MLIR-native operations, generalized transcoders, and RLCEF-driven attribution graphs with provable coverage guarantees) provide essential tools for understanding AI systems as they become increasingly critical to society. This work proves compiler integrated interpretability is both theoretically sound and practically viable, creating a foundation for systematic, scalable interpretability research.

Selected Publications

1 IAM: Interpretable AI with MLIR - A Compiler-Inspired Framework for Neural Network Analysis
Shakthi Bachala
(In preparation for submission)

2 TaintAna: Tracing Tainted Paths: A Hierarchical GNN-Based Graph Foundational Model for Secure Android Analysis
Shakthi Bachala and Witawas Srisa-An
(In preparation for submission)

3 ReHAna: An Efficient Program Analysis Framework to Uncover Reflective Code in Android
Shakthi Bachala, Yutaka Tsutano, Witawas Srisa-An, Gregg Rothermel, and Jackson Dinh
(EAI MobiQuitous 2021)

4 JitAna: A Modern Hybrid Program Analysis Framework for Android Platforms
Journal of Computer Languages, Volume 52, 2019
Yutaka Tsutano, **Shakthi Bachala**, Witawas Srisa-An, Gregg Rothermel, and Jackson Dinh

5 GranDroid: Graph-based Detection of Malicious Network Behaviors in Android Applications
Zhiqiang Li, Jun Sun, Qiben Yan, Witawas Srisa-an, and **Shakthi Bachala**
(SecureComm 2018)

6 An Efficient, Robust, and Scalable Approach for Analyzing Interacting Android Apps
Yutaka Tsutano, **Shakthi Bachala**, Witawas Srisa-An, Gregg Rothermel, and Jackson Dinh
(ICSE 2017 - Last publication before PhD break)