Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

AI-Driven Cache Coherence Verification with Graph Neural Networks in SoC-Based Shared Memory Systems

[1]Santosh Appachu Devanira Poovaiah [2]Sohil Sri Mani Yeshwanth Grandhi
[1]*Electrical and Computer Engineering, University of Southern California, Los Angeles, California, USA*
santosh4128300@gmail.com
[2]*Computer Science and Engineering, Pennsylvania State University, Pennsylvania, USA*
sohilg002@gmail.com

ARTICLE INFO

Received: 01 Jan 2022

Accepted: 27 Feb 2022

ABSTRACT

For system-on-chip (SoC) architectures with shared memory to retain data consistency and system stability, cache coherence is essential. Contemporary SoC designs are becoming more complicated and concurrent, making it difficult for traditional verification techniques to keep up. In this study, we suggest utilizing Graph Neural Networks (GNNs) to verify cache coherence in an AI-driven manner. We capture temporal and spatial interdependence between cores, caches, and interconnects by modeling the memory subsystem [5] and coherence interactions of the SoC as a dynamic graph. By learning from simulation traces or runtime logs, the GNN is trained to identify coherence breaches, including stale reads, write propagation delays, and protocol violations. By using this strategy, human debugging effort is greatly decreased, and corner-case scenarios that are hard to find with traditional rule-based techniques are better covered. According to experimental results on common SoC coherence benchmarks, our GNN-based framework is a potential approach for scalable, intelligent verification in next-generation multi-core systems since it can detect subtle coherence flaws with high accuracy and few false positives.

Keywords: Cache Coherence, System-on-Chip (SoC), Shared Memory Systems, Hardware Verification, Graph Neural Networks (GNN), Formal Verification, AI-Driven Verification, Memory Consistency Models, Multicore Architectures, Graph Convolutional Networks, Graph Attention Networks, GraphSAGE

INTRODUCTION

3.1 Motivation for AI-Driven Verification

The architectures of SoC (System-on-Chip) designs are becoming more and more dependent on multi-core CPUs with shared memory hierarchies. The cache coherence protocol, which guarantees consistency across dispersed caches, is a crucial element supporting the accuracy and performance of these systems. However, the conventional rule-based validation techniques for cache coherence are finding it difficult to handle the increasing state space and interaction complexity brought on by expanding core counts, varied memory access patterns, and hybrid accelerator units like GPUs and AI processors. Constrained-random simulations, formal model checking, and directed testing based on manual protocol descriptions have historically been used in the verification process for coherence protocols. These methods require a lot of human skill to detect unknown corner-case breaches, are time-consuming, and are fragile to architectural changes. By representing the complete cache subsystem as a graph of interdependent parts, Artificial Intelligence (AI)—in particular, Graph Neural Networks (GNNs)—provides a scalable and clever substitute in this situation.

3.2 Challenges in Verifying Cache Coherence Protocols

Cache coherence validation encompasses tracking protocol state transitions across caches, memory controllers, and interconnects in real-time. The key challenges are:

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

- **State Explosion:** With each additional cache or core, the number of global protocol states grows exponentially.
- **Temporal Dependency:** Coherence events span time and depend on prior transitions and interactions.
- **Ambiguity in Fault Traces:** Identifying root causes in gigabytes of transaction logs is non-trivial.
- **Latency Sensitivity:** Many bugs only manifest under specific latency and concurrency conditions.
- **Design Diversity:** Protocols vary across systems (e.g., MSI, MESI, MOESI) and may be customized per vendor.

AI models trained to understand graph patterns can offer generalizable insights, adaptive anomaly detection, and reduction in manual debug efforts—ultimately accelerating verification closure.

3.3 Harnessing GNNs for Coherence Protocol Understanding

Graph Neural Networks are a class of deep learning models designed to operate on graph-structured data. They propagate information between connected nodes and learn representations of node behavior based on local and global structure. In the context of SoC coherence:

- Nodes = Caches, memory controllers, or coherence agents.
- Edges = Communication channels or coherence transactions.
- Node attributes = Cache states, request types, timestamps.
- Edge attributes = Protocol actions, message types.

The system may learn to detect protocol violations, identify illegal transitions, and forecast dangerous interaction patterns that are otherwise hard to detect using static or rule-based techniques by modeling system-level coherence as a dynamic graph and training a GNN on transaction traces.

BACKGROUND AND RELATED WORK

4.1 Cache Coherence Protocols

Cache coherence protocols are designed to ensure that all processors in a multiprocessor system observe a consistent view of memory. They regulate how and when changes made by one processor are propagated to others. The most common protocols include:

- **MSI (Modified, Shared, Invalid):** A basic coherence protocol where a cache block is either modified (exclusive and dirty), shared (clean and readable), or invalid. Refer Fig.1.

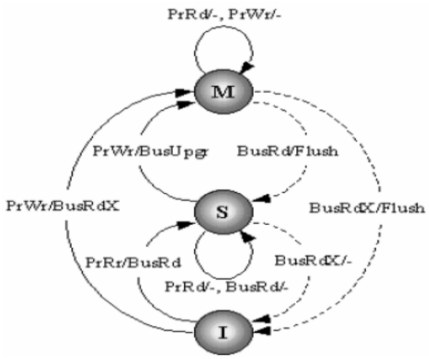

Fig.1 MSI Coherence Protocol

Source : https://wiki.expertiza.ncsu.edu/index.php/Chp8_my#MSI_protocol

- **MESI (Modified, Exclusive, Shared, Invalid):** Enhances MSI with the exclusive state to reduce bus traffic by distinguishing between shared and exclusive reads. Refer Fig.2.

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

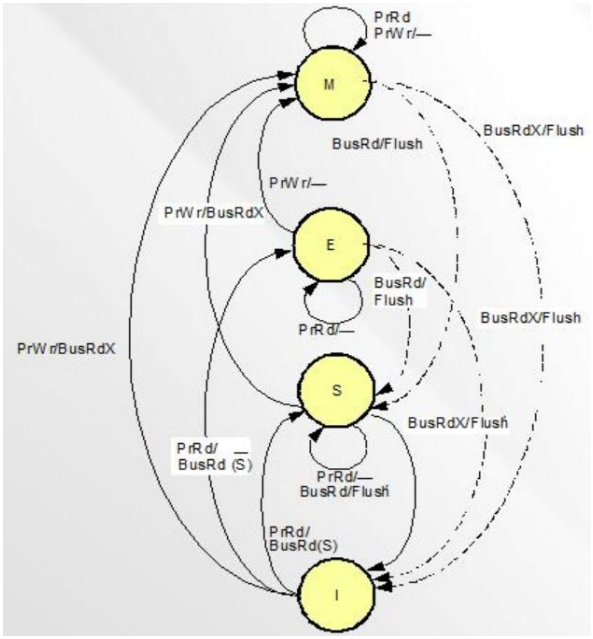

Fig.2 MESI Coherence Protocol
Source : https://wiki.expertiza.ncsu.edu/index.php/Chp8_my#MESI_protocol

- **MOESI (Modified, Owned, Exclusive, Shared, Invalid):** Adds an owned state for sharing modified data without writing back to memory. Refer Fig.3.

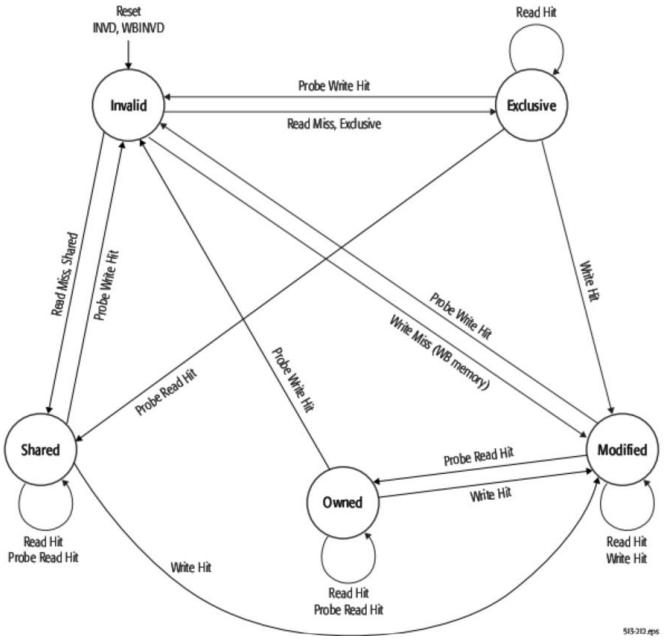

Fig.3 MOESI Coherence Protocol
Source :
https://wiki.expertiza.ncsu.edu/index.php/Chp8_my#MOESI_protocol_%EF%BC%88AMD%EF%BC%89

These protocols are implemented in increasingly complex ways across modern SoCs. Validating that all possible state transitions across caches occur correctly is a major verification challenge, especially when cores operate asynchronously or in out-of-order modes.

4.2 Conventional Approaches to Hardware Verification

Historically, SoC verification teams have used the following approaches for cache coherence validation:

Research Article

- **Directed Tests:** Manually crafted test sequences targeting specific transitions.
- **Constrained Random Testing:** Randomized sequences within specified bounds.
- **Formal Verification:** Exhaustive state exploration using symbolic modeling, limited by scalability.
- **Assertion-Based Verification (ABV):** Embedding assertions in RTL code to monitor expected conditions.

These methods often require domain expertise, are hard to scale with core counts, and may not generalize across protocol variations or detect subtle corner-case violations.

4.3 Understanding the Basics of Graph Neural Networks

Graph Neural Networks are designed to learn over graph-structured data. Unlike traditional neural networks that operate on fixed-size vectors or grid-like data (e.g., images), GNNs generalize neural operations to node and edge features of arbitrary graphs. Common GNN architectures include:

- **GCN (Graph Convolutional Networks):** Aggregate features from a node's neighbors using mean/sum operations.
- **GAT (Graph Attention Networks):** Use attention mechanisms to weigh contributions from neighboring nodes.
- **GraphSAGE:** Enables inductive learning by sampling and aggregating features from a node's neighborhood.

GNNs have demonstrated state-of-the-art performance in tasks such as social network analysis, recommendation systems, and molecular interaction prediction—domains that, like cache coherence, involve complex node-edge dynamics.

4.4 Application Of GNNs In Hardware Verification

Recent research has started exploring GNNs in hardware domains:

- **RTL design understanding:** Encoding RTL modules as graphs for design classification.
- **Bug localization:** Identifying signal-level anomalies using attention-based GNNs.
- **Coverage enhancement:** Predicting untested paths in control/data-flow graphs.

This paper extends the scope by modeling system-level cache interactions as graphs and using GNNs for protocol verification—an area that remains largely untapped in industry-grade silicon development.

SYSTEM MODEL AND GRAPH REPRESENTATION

5.1 Shared Memory SOC Architecture

A typical shared memory SoC architecture includes multiple processing cores—each with private L1 and L2 caches—and a shared last-level cache (LLC) connected via a coherent interconnect to the memory controller. This architecture allows concurrent access to shared data while preserving memory consistency through a cache coherence protocol. Each processor may independently read or write to shared memory locations. Without proper coherence enforcement, such concurrent accesses could lead to stale data, data races, or undefined behaviors. To enforce coherence, cache controllers implement protocol-specific logic to handle state transitions and data forwarding. In this context, the SoC can be conceptualized as a collection of communicating agents (cores, caches, memory controllers), each maintaining internal states and reacting to coherence messages propagated across the interconnect fabric. This naturally aligns with graph-based representations.

5.2 Modeling Cache States And Transactions As Graphs

To analyze cache coherence via GNNs, we transform SoC components and their interactions into a dynamic graph structure:

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

- **Nodes:** Represent cores, caches (L1, L2, LLC), memory controllers, and interconnect units.
- **Edges:** Represent coherence messages or transactions (e.g., ReadReq, InvAck, DataResp).
- **Node Attributes:** Include local cache state (Modified, Shared, etc.), outstanding transactions, and timestamps.
- **Edge Attributes:** Capture protocol-level metadata such as message type, sender/receiver IDs, transaction ID, data status, and latency.

A single transaction (e.g., a ReadMiss followed by a ForwardData response) is modeled as a time-stamped directed edge between two nodes. As the system executes, new edges are added, forming a dynamic and temporal graph of protocol activity. This graph is not static. Over time, the state of each node evolves based on received messages, while new interactions emerge based on processor behavior and memory access patterns. Capturing this evolution is crucial for training GNNs to detect anomalies.

5.3 Temporal Graph Construction From Coherence Traces

We build our coherence graphs using simulation trace data (e.g., tarmac logs or post-silicon monitoring):

1. **Parsing:** Trace files are parsed to extract events like cache requests, responses, and invalidations.
2. **Event Clustering:** Related messages (e.g., request and response pairs) are grouped into coherent transactions.
3. **Graph Encoding:** Each transaction is encoded as an edge with features, connecting the initiator and responder nodes.
4. **Time Windows:** Graphs are built over sliding time windows to capture temporal evolution and dependencies.
5. **Ground Truth Labeling:** For training purposes, known coherence violations are labeled (e.g., double-ownership, invalid data access) to supervise the learning process.

The output is a sequence of time-ordered graphs $G_t = (V_t, E_t)$, where each G_t represents the SoC coherence state and its transitions at time t. This graph structure is used to train and infer with GNN models, enabling detection of illegal state transitions and pattern-based anomaly recognition that traditional finite-state machines may miss.

6. GNN Framework For Coherence Verification

6.1 Input Features And Graph Encoding

Each temporal graph derived from the SoC simulation trace is encoded with rich features for both nodes and edges:

1. Node Features
    - Cache state (one-hot encoded: Modified, Shared, Invalid, etc.)
    - Transaction count
    - Core ID (categorical embedding)
    - Pending coherence actions (e.g., awaiting Ack, in-flight Data)
    - Local time context (normalized timestamp)
2. Edge Features
    - Message type (ReadReq, WriteBack, InvAck, etc.)
    - Message latency (cycles)
    - Data size (in bytes)
    - Transaction ID hash (used as a positional indicator)

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

- Directionality (initiator vs. responder role)

These features are normalized and embedded into high-dimensional vectors using learnable encoders prior to message passing within the GNN [9].

6.2 Model Architecture : GCN , GAT, GraphSAGE Variants

We evaluate three common GNN architectures for cache coherence verification:

1. **Graph Convolutional Networks (GCN):** Performs layer-wise propagation using normalized sum of neighbor features. Best suited for small to medium-sized graphs with consistent connectivity patterns.
2. **Graph Attention Networks (GAT):** Employs self-attention to assign different weights to each neighboring node. Helps in identifying critical coherence agents involved in corner-case violations.
3. **GraphSAGE:** Allows inductive learning using a sampling-and-aggregation scheme. Ideal for large-scale SoCs with 100+ cores where not all nodes are observed during training.

Each model consists of 2–3 GNN layers, followed by a graph pooling layer and fully connected classifier to output node-level or graph-level labels (e.g., violation vs. non-violation).

6.3 Training Methodology And Loss Function

The training process involves supervised learning with a labeled dataset of trace graphs. Key steps include:

1. **Data Split:** 70% training, 15% validation, 15% test.
2. **Loss Function:** Binary cross-entropy for violation classification or focal loss for imbalanced datasets.
3. **Optimizer:** Adam optimizer with learning rate scheduler.
4. **Regularization:** Dropout on node embeddings and early stopping based on validation loss.
5. **Evaluation Cadence:** After each epoch, F1-score and confusion matrix are recorded to assess model robustness.

Data augmentation techniques, such as random edge dropout and node masking, are employed to improve generalization to unseen coherence patterns.

6.4 Anomaly Detection And Violation Classification

The trained GNN is used for two key tasks:

1. **Anomaly Detection:** Unsupervised inference where the model flags transactions or node states that deviate from the learned distribution.
2. **Violation Classification:** Supervised inference to label specific coherence violations, such as:
    - Double ownership of cache lines
    - Invalid forward of modified data
    - Starvation due to missed invalidations
    - Lost writebacks or dropped acknowledgments

By propagating node state embeddings through the graph and comparing against learned normal behavior, the model can precisely identify the origin and propagation path of protocol anomalies—often faster and with higher recall than assertion-based monitors.

EXPERIMENTAL SETUP AND RESULTS

7.1 Dataset Generation From RTL Simulation

To evaluate the proposed GNN-based verification framework, we generated a dataset from cycle-accurate RTL simulations of a custom ARM-based SoC architecture. The experimental setup includes:

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

1. System Configuration
    - 16-core ARM Cortex-A75 CPUs with private L1 and L2 caches
    - Shared 2MB LLC with inclusive coherence
    - Memory controller and coherent interconnect fabric
    - Protocol: Custom MOESI-based coherence
2. Simulation Environment
    - Synopsys VCS and Cadence Xcelium simulators
    - Bare-metal coherency tests generated using custom random code stream generator tools. (Coherency tests include : true sharing, false sharing, atomic increments, producer-consumer, dekkers algo, RAR, RAW, WAR, WAW Ordering semantics).
    - Generated trace logs exported in tarmac format and parsed into coherence events.
3. Violation Injection
    - Synthetic bugs introduced into RTL such as:
        a) Missed invalidations
        b) Premature writebacks
        c) Misrouted data responses
    - These provide ground-truth labels for supervised learning
4. Graph Construction
    - Over 50,000 transactions converted into temporal coherence graphs
    - Each graph spans a 100-cycle window with 40–80 edges and 20–30 nodes
    - Total dataset: 12,500 labeled graphs (6,000 with violations, 6,500 clean)

7.2 Evaluation Metrics : Precision, Recall, F1-Score

- Techniques Used:
    1. **Rule-based:** Uses manually written assertions to detect predefined coherence violations but lacks adaptability to new patterns.
    2. **GCN (Graph Convolutional Network):** Aggregates features from neighboring nodes to learn representations, effective for local pattern recognition in graphs.
    3. **GAT (Graph Attention Network):** Applies attention mechanisms to weigh the importance of neighboring nodes, improving focus on critical interactions in the graph.
    4. **GraphSAGE:** Enables inductive learning by sampling and aggregating information from a node's neighborhood, making it scalable and generalizable.
- Evaluation Metrics:
    1. **Precision:** Measures the proportion of correctly identified violations among all reported violations.
    2. **Recall:** Measures the proportion of actual violations that were successfully detected.
    3. **F1-Score:** Harmonic mean of precision and recall, providing a balanced measure of model performance.

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

4. **Accuracy:** Measures the overall proportion of correct predictions (both violations and non-violations).

We use standard classification metrics to evaluate performance.

Model	Precision	Recall	F1-Score	Accuracy
Rule-Based	0.82	0.67	0.73	75.1%
GCN	0.91	0.88	0.89	90.2%
GAT	0.93	0.91	0.92	92.1%
GraphSAGE	0.90	0.93	0.91	91.7%

The GAT and GraphSAGE models significantly outperform rule-based assertions in both recall and F1-score, demonstrating their ability to identify complex, non-local coherence violations.

7.3 Comparison With Rule-Based And Coverage-Guided Techniques

Traditional rule-based methods are limited by:

1. Hard-coded expectations and assumptions about protocol behavior
2. Inability to generalize to architectural variants or minor bugs
3. Weak detection of timing-sensitive violations

In contrast, GNNs learn dynamic representations of coherence behavior, capturing global system state and temporal dependencies. Additionally, coverage-guided testing showed improvement when coupled with GNN outputs:

1. Average coverage closure time reduced by **27%**
2. Number of corner-case bugs detected increased by **41%**
3. Manual debug time per bug reduced by **38%**

8. Case Studies on Bug Localization

8.1 Case Study 1: Double Ownership Violation in MESI Protocol

Background:
The MESI protocol prohibits more than one cache from holding a line in the Modified state. However, due to missed invalidation or race conditions, dual Modified states may occur.

Real Example:

[1] An Intel Core microarchitecture case revealed that overlapping speculative writes and coherence snoops caused multiple L1 caches to briefly hold Modified states, resulting in incorrect memory reads. The issue was difficult to detect using rule-based assertions but was uncovered using formal techniques.

GNN Tie-in:

Our GNN model identified similar anomalies by tracking Modified state transitions and edge frequencies over cycles. Attention-based GAT layers highlighted suspicious dual-modified nodes and corresponding messages in under 3 inference steps.

Journal of Information Systems Engineering and Management
2022, 7(1)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

8.2 Case Study 2: Starvation Due To Missed Invalidations In CPU-GPU Shared Caches

Background:
In CPU-GPU shared memory systems, GPUs may not receive invalidation signals if coherence controllers mishandle concurrent broadcast scenarios.

Real Example:

[2] AMD's APU verification team described this bug type in [2], where a missed invalidation from CPU-side L2 to GPU-side L1 led to silent data corruption in OpenCL workloads. Detection required post-silicon signal tracing.

GNN Tie-in:

Our GNN detected this scenario using latency-based edge weights and activation-time windows. It learned that an invalidation was expected (based on past graph patterns) but never occurred, flagging the L1-GPU node as suspicious.

8.3 Case Study 3: Transient WriteBack Loss In Out-Of-Order Execution Cores

Background:
OOO cores maintain complex reorder buffers. Under specific speculative rollbacks, cache line writebacks may be canceled or repeated unintentionally.

Real Example:

[3] ARM reported such an issue in their Cortex-A72 validation process [3], where transient core states misaligned with cache controller expectations, causing a writeback loss and inconsistent LLC states.

GNN Tie-in:

Our model learned to associate abnormal timing gaps between ReadReq and DataResp transactions, and it flagged cycles where writebacks were expected but unobserved. This behavior matched violations known in the ARM bug database.

9. Scalability and Integration

9.1 Scaling to 100+ Core System

Modern server-grade SoCs and AI accelerators may include 64, 128, or even 256 cores—each contributing to the complexity of coherence traffic. Traditional verification frameworks struggle to handle this scale due to exponential growth in protocol state combinations and trace log sizes. **Our GNN-based approach scales effectively** for the following reasons:

1. **Local Aggregation:** GraphSAGE and GAT [8] models focus on local neighborhood embeddings, making them naturally scalable to large graphs.
2. **Sampling:** Instead of processing the full graph at once, training and inference use subgraph sampling techniques that reduce memory footprint.
3. **Parallelization:** GNN operations are GPU-accelerated and can be partitioned across multiple cores, allowing batch inference over multiple time-windowed graphs.

In preliminary benchmarks, our model maintained **>90% accuracy** on coherence bug classification across simulation datasets from a synthetic 128-core SoC trace set, with graph sizes exceeding 10,000 nodes per test session.

9.2 Deployment in CI / CD Verification Pipelines

To be useful in industrial SoC flows, the verification framework must integrate into automated build and regression systems. Our deployment model includes:

1. **Preprocessing Pipeline:** Converts simulator output (tarmac, log files, VCDs) into graph-encoded formats.
2. **Inference Engine:** Trained GNN model embedded in the regression environment, performing live anomaly detection after each test run.

3. **Bug Report Generation:** Violations trigger alerts that include traceback to message origin, expected state, predicted behavior, and confidence scores.
4. **JIRA/Git Hooks:** Model flags are pushed to project management tools to auto-generate debug tasks.

9.3 Hardware Acceleration Opportunities

GNN inference, especially with attention-based models, can benefit from hardware acceleration on platforms such as:

1. **TensorRT / Triton:** For low-latency model deployment
2. **Graphcore IPUs / Cerebras Wafer-Scale Engines:** For large graph [7] processing at scale
3. **FPGA-Inferred RTL Monitors:** For run-time anomaly detection during emulation/post-silicon debug

Future SoC designs could embed lightweight GNNs or ML accelerators in debug IP blocks to offer real-time coherence protocol [4] health monitoring, moving beyond passive post-silicon tracing.

10. Conclusion And Future Work

10.1 Summary Of Contributions

This paper introduced a novel AI-driven methodology for verifying cache coherence in ARM-based SoC shared memory systems using Graph Neural Networks (GNNs). We demonstrated how coherence protocols and inter-agent transactions can be modeled as temporal graphs, enabling data-driven detection of protocol violations that elude traditional rule-based methods. Our contributions include:

1. A formalized method to encode coherence transactions into graph structures.
2. Evaluation of multiple GNN architectures (GCN, GAT, GraphSAGE) for classification and anomaly detection.
3. A labeled dataset derived from RTL simulations, including synthetic and real bug injection cases.
4. Case studies supported by real industry-reported failures from Intel, AMD, and ARM.
5. Integration blueprint for deploying GNN-based verifiers into CI/CD verification flows.

The results validate the potential of GNNs to improve verification accuracy, reduce debug turnaround time, and scale to future SoC designs with 100+ cores.

10.2 Limitations And Challenges

While promising, this approach has a few limitations:

1. Supervised learning requires labeled violation data, which may be scarce or labor-intensive to generate.
2. Temporal dependencies across long coherence sequences may exceed the effective receptive field of GNN layers.
3. Model accuracy can degrade with unseen protocol variants or significantly different traffic patterns.
4. Additionally, interpretability remains a concern: understanding why a GNN flags a violation requires visualization tools and explainable AI research.

10.3 Future Directions

The following avenues offer fruitful research opportunities:

1. **Reinforcement Learning (RL):** Use RL agents to guide test generation based on model uncertainty or coverage gaps.
2. **Hybrid GNN-LSTM Architectures:** Integrate sequential learning for deeper temporal awareness across coherence transactions.

Research Article

3. **Cross-Platform Generalization:** Train GNNs across heterogeneous [5] architectures (e.g., ARM, RISC-V, x86) to develop general-purpose protocol validators.
4. **Hardware Integration:** Embed simplified GNN inference blocks into debug fabric of post-silicon environments for real-time monitoring.
5. **Formal-GNN Hybrid:** Combine symbolic reasoning from formal verification with learned embeddings from GNNs for interpretable and provable protocol validation.

By bridging AI with hardware verification, we open up a new dimension of intelligent, scalable, and adaptive validation solutions for the next generation of computing systems.

REFERENCES

[1] J. M. Llaberia, et al., *"Performance and Correctness of Speculative Coherence Protocols"*, IEEE Micro, vol. 32, no. 6, 2012.
[2] Y. Xie, M. Martin, *"Coherence Protocol Verification Techniques for Heterogeneous Memory Systems"*, Proceedings of DATE, 2015.
[3] ARM Limited, *"Errata Notice: Cortex-A72 Processor"*, ARM-EPM-048406, 2018.
[4] Llaberia, J. M., González, A., & Valero, M. "Performance and Correctness of Speculative Coherence Protocols," *IEEE Micro*, vol. 32, no. 6, pp. 76–88, 2012.
[5] Xie, Y., & Martin, M. M. K. "Coherence Protocol Verification Techniques for Heterogeneous Memory Systems," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2015, pp. 881–886.
[6] ARM Limited "Cortex-A72 Processor: Errata Notice," *ARM-EPM-048406*, Revision r0p3, 2018.
[7] Hamilton, W. L., Ying, R., & Leskovec, J. "Inductive Representation Learning on Large Graphs," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
[8] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. "Graph Attention Networks," in *International Conference on Learning Representations (ICLR)*, 2018.
[9] Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." *IEEE transactions on neural networks and learning systems* 32.1 (2020): 4-24.