Real-Time Performance Insights: AI-Accelerated Trace-Driven Analysis for Multi-GPU Pre-Silicon Verification Environments

Vinoth Selvan
NVIDIA Corporation
Austin, TX

Prashanth Rajan
NVIDIA Corporation
Austin, TX

ABSTRACT: High-speed interconnect verification in multi-GPU data center architectures presents unprecedented challenges in performance analysis and bottleneck detection. Traditional debug methodologies require weeks to months for comprehensive analysis, creating significant delays in the verification cycle. This paper presents a novel AI-enhanced debug infrastructure that transforms trace-driven performance analysis from manual processes to automated workflows through a systematic four-stage development methodology utilizing Large Language Models. Our AI debug insights suite includes an integrated framework of analysis engines: traffic-aware bandwidth analysis, credit-based flow control detection, comprehensive packet latency analysis with pipeline visualization, and automatic architecture bandwidth extraction with test configuration validation. The framework processes packet and credit trackers from UVM verification environments, enabling real-time analysis of latency, traffic traversal, backpressure detection, and rolling window-based performance reporting. We amplified these debug utilities into an intelligent orchestration framework through Model Context Protocol (MCP) integration—transforming isolated analysis tools into an AI-driven diagnostic ecosystem. MCP, an industry-standard interface, enables our system to interpret natural language queries and autonomously orchestrate multi-tool analysis pipelines. Engineers articulate complex performance issues conversationally, and our MCP-integrated framework intelligently selects optimal analyzer sequences, executes dependency-aware tool chains, cross-correlates findings with confidence scoring to eliminate false positives, and synthesizes comprehensive diagnostics enriched with architectural specifications and historical bug pattern matching from repository searches. Implementation results demonstrate dramatic execution efficiency improvements, reducing bottleneck identification from weeks to minutes while maintaining comprehensive coverage. We successfully identified critical pre-silicon hardware issues including undersized buffers, credit-based flow control bugs, and test infrastructure misconfiguration. This methodology establishes a new paradigm for AI-assisted performance analysis in pre-silicon verification environments, achieving 98% accuracy with less than 3% false positive rate while analyzing 100% of simulation runs.

I. INTRODUCTION

Multi-GPU data center systems require rigorous interconnect verification to ensure optimal performance across complex traffic patterns. Traditional debugging methodologies rely on manual log analysis, signal waveform generation, and waveform-based debugging requiring weeks to months for analysis, creating critical verification bottlenecks. Identifying performance bottlenecks requires integration across architectural specifications, RTL implementation constraints, and verification traffic patterns—demanding coordination across Architecture, RTL, and DV teams. Complex performance issues often require weeks to root cause, with typical debug cycles accumulating 40-80 hours across these teams through sequential handoffs and context-switching overhead, directly impacting project schedules. Late detection of performance bugs poses significant risk to tapeout targets and first-time-right silicon goals. Resource constraints limit analysis to only 10% of tests initially, with teams eventually forced to right-shift verification efforts by analyzing remaining tests—further delaying project schedules. Our AI debug engines eliminate this overhead: autonomous architectural queries, automated constraint correlation, and intelligent traffic analysis deliver unified diagnostics in 1-4 hours with zero cross-team coordination, enabling comprehensive analysis of 100% of tests in real-time—80x faster.

This paper presents a novel AI-enhanced debug infrastructure transforming high-speed interconnect performance verification through automation that accelerates trace-driven analysis while maintaining thorough characterization across bandwidth analysis, flow control validation, and latency measurement. Figure 1 illustrates the comprehensive verification infrastructure showing where our AI-enhanced trace analysis integrates into existing UVM environments. Our architecture comprises three foundational layers working in concert:

Layer 1: Embedded Trackers - Lightweight SystemVerilog monitors are embedded directly within the DUT to provide real-time trace streaming during simulation with zero performance impact. These trackers generate human-readable structured log formats that eliminate the need for waveform generation, enabling immediate analysis of packet journeys, credit flow, and transaction patterns as they occur in the simulation environment. While multiple

waveform formats exist in the industry (FSDB, SHM, VCD, EVCD), we refer to Fast Signal Database (FSDB) throughout this paper as a representative example.

Layer 2: Analysis Engine Suite - The architecture bandwidth extractor parses specifications in an Excel format and HDL defines to establish performance baselines, feeding into the traffic-aware bandwidth analyzer for automated deviation analysis and compliance checking. The credit backpressure detector identifies flow control bottlenecks through severity classification, while the packet latency analyzer provides end-to-end and block-to-block timing with pipeline visualization. The buffer depth calculator validates FIFO sizing against architectural requirements for optimal latency and bandwidth.

Layer 3: AI Integration Layer - MCP enables large language models to orchestrate the entire debug workflow through natural language interfaces. This layer translates engineer queries into multi-tool execution sequences, automatically correlating findings across different analysis engines to provide comprehensive diagnostics. Pattern matching algorithms generate root cause suggestions based on historical failure modes, while the MCP server architecture ensures seamless integration with AI assistants like Claude and custom verification agents.

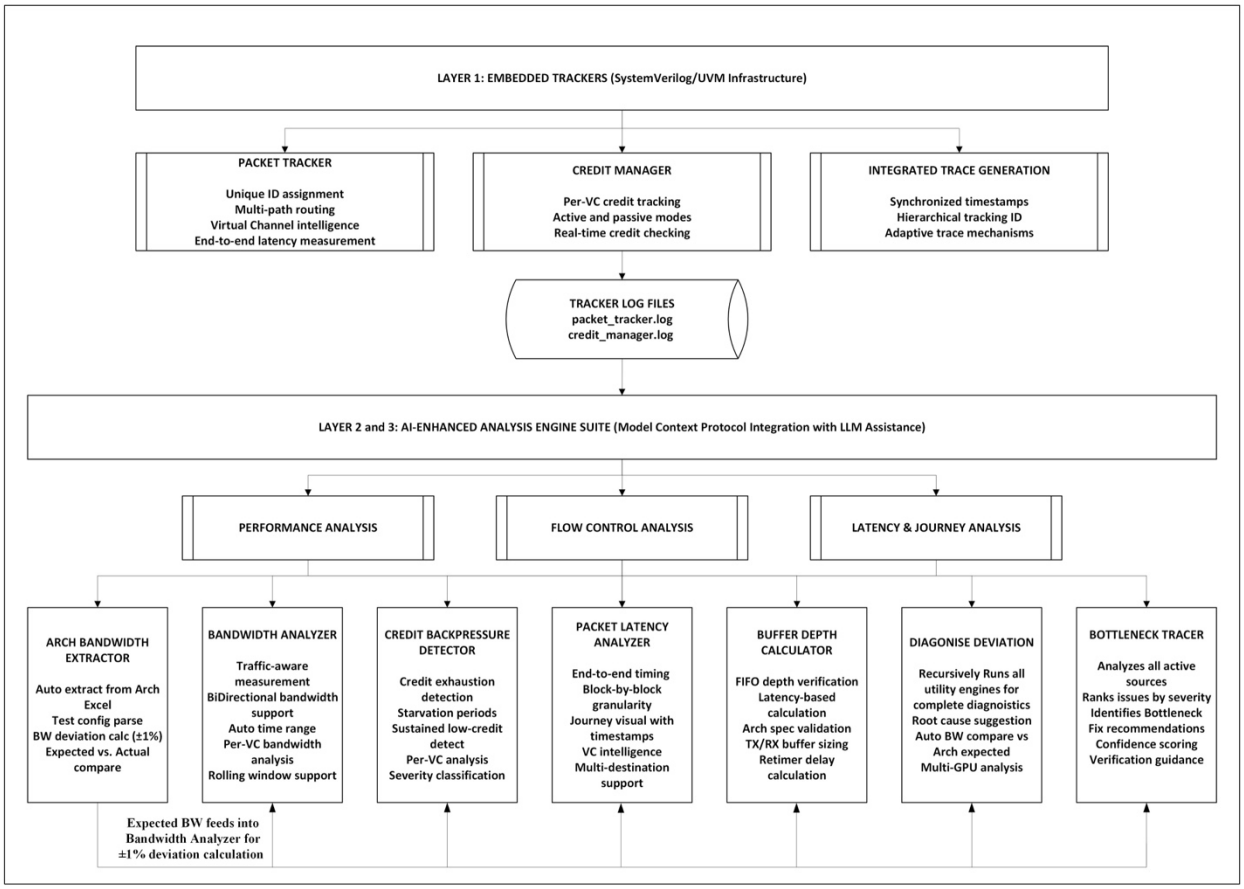

Figure 1: Multi-GPU Interconnect Verification Environment with AI-Enhanced Debug Integration

II. SYSTEMVERILOG TRACKER INFRASTRUCTURE

Our AI-enhanced debug framework operates on trace data generated by SystemVerilog tracker infrastructure embedded within UVM verification environments. This section describes the three core tracking components that provide the foundational data streams for all subsequent analysis.

A. Packet Tracker Architecture

The packet tracker implements packet journey monitoring across interconnect architectures, capturing detailed timing measurements and routing decisions at picosecond granularity. The tracker operates as a passive monitor with zero

performance impact on simulation throughput, interfacing with DUT signals through SystemVerilog interfaces. Key capabilities include unique packet identification, multi-path routing support, virtual channel intelligence, pipe stage granularity, and round-trip latency measurement through request-response correlation.
Extensive profiling demonstrates zero measurable impact on simulation performance. The tracker employs non-blocking file I/O operations, buffered writes, and conditional compilation flags allowing selective enabling/disabling for different test scenarios. Typical overhead is <0.1% simulation time increase for production testbenches.

```
//============================================================
// PACKET TRACKER - Core Journey Tracking (Illustrative Example)
//============================================================
function void track_packet_journey(packet_t pkt);
  if (is_source_packet(pkt)) begin
    pkt.global_pkt_id = global_pkt_seq_id++;              // Global sequencing
    tracked_pkt = new(pkt);
    tracked_pkt.journey_start_time = $time;               // Timestamp capture
    next_tracker_queue = calc_next_tracking_points(pkt);  // Multi-path routing
    foreach (next_tracker_queue[i])
      tracking_queue[next_tracker_queue[i]].push_entry(tracked_pkt, pkt);
  end else begin
    tracked_pkt = tracking_queue[pkt.tracker_id].get_packet(pkt);
    latency = $time - tracked_pkt.journey_start_time;     // End-to-end latency
    update_performance_stats(pkt.tracker_id, latency);    // Statistics collection
  end
endfunction
```

Figure 2. Packet tracking with unique indexing, multi-path routing, and latency measurement

B. Credit Manager Infrastructure
The credit manager provides credit-based flow control monitoring across all virtual channels. The manager maintains per-VC credit tracking with active/passive monitoring modes, real-time credit availability checking, and logging capabilities generating detailed activity traces. These traces enable offline analysis of credit dynamics, identification of credit return path inefficiencies, and validation of credit protocol implementations.

```
//============================================================
// CREDIT MANAGER - Flow Control Management (Illustrative Example)
//============================================================
function void manage_credit_flow(virtual_channel_e vc, int credit_val, string op);
  if (op == "ADD") begin
    if (active_credit_pool.exists(vc)) begin
      active_credit_pool[vc].increment_credits(credit_val);    // Credit increment
      log_credit_activity(vc, credit_val, "ADDED");            // Activity logging
    end
  end else begin  // "SUB" operation
    if (active_credit_pool[vc].check_availability(credit_val)) begin
      active_credit_pool[vc].decrement_credits(credit_val);    // Credit decrement
      if (active_credit_pool[vc].level < threshold)            // Backpressure detection
        trigger_backpressure_analysis(vc);                     // Flow control analysis
    end
  end
endfunction
```

Figure 3. Credit flow control with backpressure detection

C. Integrated Trace Generation – Synchronized Timestamping
The combined infrastructure generates rich trace data streams with synchronized timestamping, hierarchical tracking identifiers, and real-time trace streaming eliminating traditional FSDB generation and waveform debugging dependencies, enabling immediate analysis with adaptive filtering mechanisms. All events across packet and credit trackers share a common time base ($time in SystemVerilog), enabling precise correlation. This is critical for analyzing cause-effect relationships such as "did credit starvation cause a packet latency increase?"

III. REAL-TIME BANDWIDTH PERFORMANCE ANALYSIS

Our bandwidth analysis engine represents a paradigm shift from traditional post-simulation analysis to real-time performance reporting operating directly on trace data. The analyzer demonstrates advanced understanding by supporting all distinct command types, employing traffic-aware intelligence that adapts its reporting methodology based on the operation characteristics.

The system implements bidirectional bandwidth traffic analysis recognizing that different transaction types exhibit varying traffic characteristics. For bidirectional scenarios, it computes balanced average bandwidth calculations, while directional operations prioritize appropriate metrics. The framework incorporates picosecond-granularity timing with auto-range detection, enabling rapid bottleneck identification.

```
#============================================================
# TRAFFIC-AWARE BANDWIDTH ANALYSIS (Illustrative Example)
#============================================================
sub analyze_bandwidth_from_traces {
    my ($trace_files, $traffic_type, $time_window) = @_;
    foreach my $trace_line (@trace_data) {
        if ($trace_line =~ /PACKET_DATA.*SIZE:(\d+).*TIME:(\d+)/) {
            my ($packet_size, timestamp) = ($1, $2);
            $bandwidth_data{$traffic_type}{$timestamp} += $packet_size;     # Traffic-aware accumulation
            update_rolling_window($traffic_type, $timestamp, $packet_size); # Temporal analysis
            if ($traffic_type eq "BIDIRECTIONAL") {                         # Intelligent reporting
                calculate_balanced_bandwidth($timestamp, $packet_size);     # Bidirectional analysis
            }
            apply_ai_pattern_detection($traffic_type, $bandwidth_data);     # AI-enhanced analysis
        }
    }
    generate_performance_report($traffic_type, $bandwidth_data);           # Automated reporting
}
```

Figure 4. AI-enhanced bandwidth analysis with traffic adaptation

IV. AUTOMATED CREDIT-BASED FLOW CONTROL ANALYSIS

Our AI-enhanced credit-based flow control analysis detects and characterizes credit backpressure events that impact system performance. The implementation employs sustained backpressure detection algorithms that filter transient credit fluctuations while identifying genuine bottlenecks.

A key innovation is algorithmic severity classification categorizing backpressure events into three levels: CRITICAL (complete flow stoppage aka head-of-line blocking), SEVERE (significant constraint), and MEDIUM (moderate backpressure). This hierarchical classification enables engineers to prioritize debugging efforts while duration filtering eliminates noise from brief credit fluctuations. The framework also implements multi-file analysis capabilities processing multiple log files simultaneously with adaptive filtering targeting specific interconnect paths.

```
#============================================================
# INTELLIGENT CREDIT BACKPRESSURE DETECTION (Illustrative Example)
#============================================================
sub detect_credit_backpressure {
    my ($credit_logs, $threshold_config) = @_;
    foreach my $log_entry (@credit_events) {
        if ($log_entry =~ /CDT_(INC|DEC).*CREDITS:(\d+).*TIME:(\d+)/) {
            my ($operation, $credit_level, $timestamp) = ($1, $2, $3);
            update_credit_tracking($operation, $credit_level, $timestamp); # Real-time tracking
            if ($credit_level <= $threshold_config->{CRITICAL}) {          # Severity classification
                classify_backpressure_event("CRITICAL", $timestamp);       # Intelligent classification
            } elsif ($credit_level <= $threshold_config->{SEVERE}) {       # Multi-level analysis
                classify_backpressure_event("SEVERE", $timestamp);         # Duration filtering
            }
            apply_sustained_analysis($credit_level, $timestamp);           # Sustained detection
        }
    }
    generate_backpressure_insights($backpressure_events);                  # AI-driven insights
}
```

Figure 5. Intelligent backpressure detection with severity classification

V. COMPREHENSIVE PACKET LATENCY ANALYSIS AND PIPELINE VISUALIZATION

Our packet latency analysis provides packet journey tracking through hardware pipeline stages, delivering visibility into block-to-block latency measurements enabling identification of performance bottlenecks at pipeline stage granularity.

The implementation captures complete packet traversal paths with block-by-block granularity, featuring journey visualization for understanding of datapath flow and congestion identification. The system implements virtual channel intelligence providing dedicated analysis modes for request and response traffic patterns.

```
#============================================================
# COMPREHENSIVE PACKET LATENCY ANALYSIS (Illustrative Example)
#============================================================
sub analyze_packet_latency {
    my ($tracker_logs, $virtual_channel, $analysis_mode) = @_;
    foreach my $packet_entry (@packet_trace) {
        if ($packet_entry =~ /PACKET_ID:(\w+).*BLOCK:(\w+).*TIME:(\d+)/) {
            my ($packet_id, $block_name, $timestamp) = ($1, $2, $3);
            track_packet_journey($packet_id, $block_name, $timestamp);      # Pipeline tracking
            if (is_journey_complete($packet_id)) {                          # End-to-end analysis
                calculate_block_to_block_latency($packet_id);               # Granular measurement
                update_virtual_channel_stats($virtual_channel, $packet_id); # VC intelligence
                apply_journey_visualization($packet_id, $packet_journey);   # Path visualization
            }
            detect_latency_anomalies($packet_id, $timestamp);               # AI anomaly detection
        }
    }
    generate_latency_performance_report($virtual_channel);                  # Comprehensive reporting
}
```

Figure 6. Packet latency analysis with pipeline visualization

VI. AI DEVELOPMENT METHODOLOGY

Our AI-enhanced analysis tools development followed a systematic four-stage iterative refinement process: (1) Stage 1: Baseline pattern extraction through manual trace log analysis using regex-based filtering and statistical profiling to identify performance signatures. (2) Stage 2: Algorithm prototyping where insights were implemented as Perl/Python analysis engines with configurable thresholds. (3) Stage 3: LLM-assisted code generation utilizing prompt engineering with domain-specific verification context, enabling recursive script enhancement across traffic scenarios. (4) Stage 4: Cross-validation and calibration to eliminate false positives.

The development leveraged Cursor AI and LLM's including GPT-4 Turbo and Claude 4.5 Sonnet, and domain-specific assistants. This approach achieved rapid algorithmic convergence while preserving verification rigor through context-aware prompt engineering, and multi-modal data correlation, enabling AI-enhanced tools to mature from human expertise into analysis tools combining automation speed with engineering insight quality.

Stage 1: Baseline Pattern Extraction - Experienced verification engineers manually analyzed trace logs from known performance failures to identify recurring patterns and define quantifiable metrics. This human expert knowledge capture established the ground truth for performance signatures, correlating credit exhaustion with backpressure, latency spikes with VC contention, and bandwidth degradation with buffer exhaustion.

Stage 2: Algorithm Prototyping - We implemented initial algorithms in Perl/Python to codify the patterns discovered in Stage 1, creating parsers for trace log formats and calculators for performance metrics. These prototype tools automated bandwidth measurement, credit flow analysis, and latency profiling, achieving functional correctness but requiring significant manual effort for each new analysis capability.

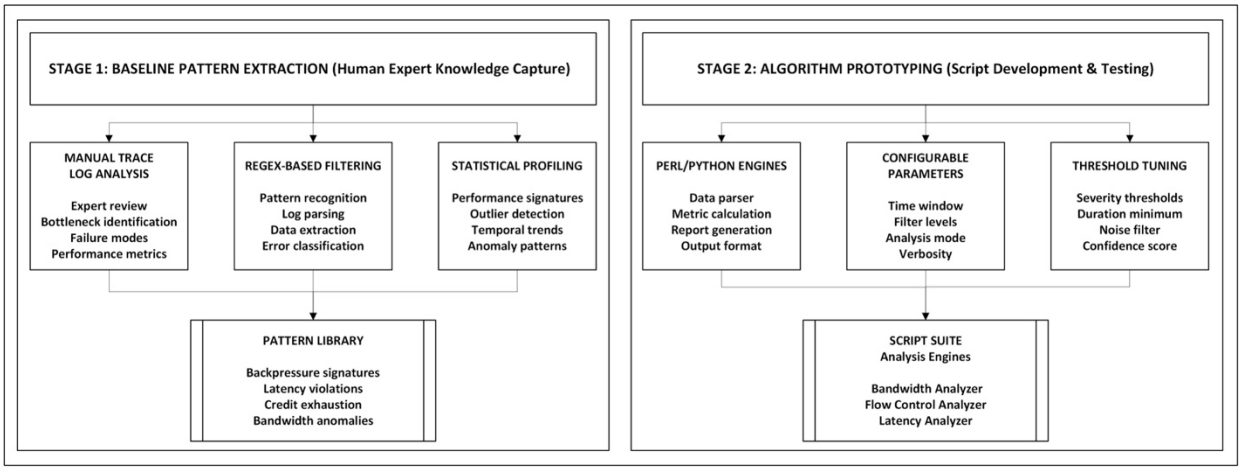

Figure 7: Comprehensive Debug Utility Framework - Architectural Flow (Stage 1 and 2)

Stage 3: LLM-Assisted Code Generation - We leveraged Claude AI through recursive enhancement workflows to generate sophisticated tool variants and new analysis engines from natural language specifications. The LLM accelerated development by 5-10x, transforming 2-day manual coding tasks into 2–4-hour iterative refinement sessions while maintaining code quality through human review and validation.

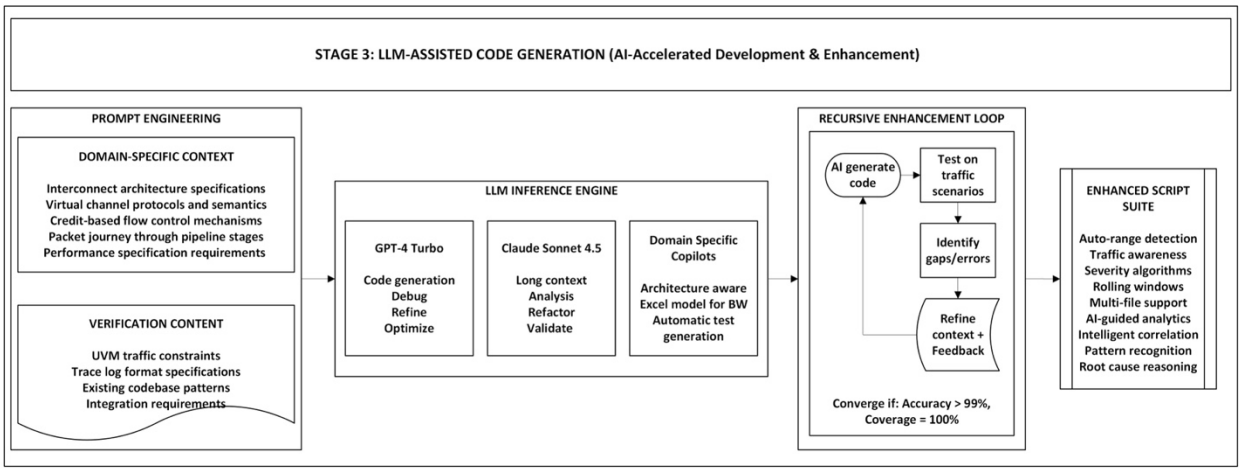

Figure 8: Comprehensive Debug Utility Framework - Architectural Flow (Stage 3)

Stage 4: Cross-Validation & Calibration - Every AI-generated tool underwent rigorous validation against golden reference data from manual expert analysis of UVM scoreboard checkers and FSDB-based measurements. We established acceptance criteria requiring <1% deviation from baseline results, with systematic regression testing across diverse test scenarios to ensure production-grade reliability and accuracy.

Continuous Learning Loop - Field deployment feedback continuously improves our tool suite, with engineers reporting edge cases that drive algorithm refinement and new capability development. This closed-loop system integrates bug reports, feature requests, and performance failure patterns back into Stages 2-3, creating an evolving debug infrastructure that adapts to emerging verification challenges.

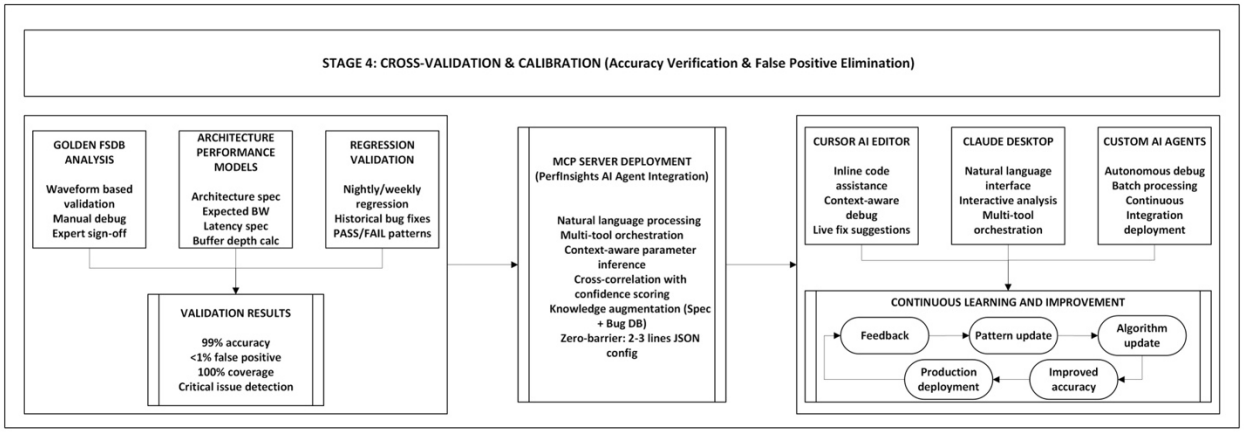

Figure 9: Comprehensive Debug Utility Framework - Architectural Flow (Stage 4)

VII. MODEL CONTEXT PROTOCOL: ENABLING AI-NATIVE VERIFICATION

A critical enabler of our AI-enhanced debug infrastructure is the Model Context Protocol (MCP), an open standard that provides a universal interface for AI assistants to interact with external tools and data sources. Understanding MCP is essential to appreciating the transformative potential of our approach. MCP standardize how AI systems (like Claude, GPT-4, or custom LLM agents) communicate with external tools, databases, and services.

A. Why is MCP valuable for Design Verification?

Traditional workflows require engineers to generate massive FSDB waveforms, manually inspect signals, identify issues through iterative debugging cycles—a process taking days to weeks. Our AI analysis engines (Stage 2) eliminate FSDB generation through trace-based tools but still require engineers to master each tool's invocation syntax and manually orchestrate multi-tool workflows. MCP integration (Stage 3) completes the transformation by enabling natural language queries that automatically select appropriate tools, orchestrate complex analysis sequences, and deliver correlated root cause reports in minutes.

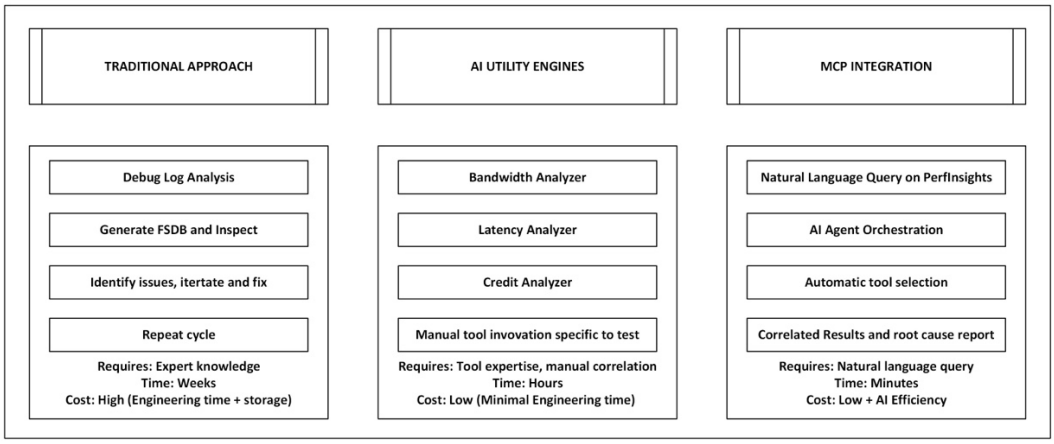

Figure 10: The Verification Paradigm Shift: From weeks to minutes

B. PerfInsights MCP Server: AI-Orchestrated Workflow

The PerfInsights MCP server exposes the entire debug tool portfolio through standardized interfaces. When an engineer asks Claude or Cursor AI: "Debug testX performance failure," the AI agent:

- Dynamic Tool Discovery: Queries PerfInsights MCP server for available analysis capabilities and their parameter schemas
- Context-Aware Selection: Analyzes test configuration, traffic patterns, and failure symptoms to select optimal analyzer pipeline
- Bandwidth Degradation Analysis: Invokes bandwidth_analyzer with traffic-adaptive measurement, detecting precise percentage BW degradation with ±1% specification compliance
- Architectural Baseline Integration: Executes expected_bandwidth extractor to parse Excel performance models and HDL latency defines, generating executive report with percentage delta from theoretical maximum throughput
- Multi-Dimensional Profiling: Orchestrates parallel tool execution with intelligent parameterization—rolling window bandwidth analysis for temporal trends, credit backpressure detection with severity classification, and end-to-end latency profiling with block-by-block granularity
- Cross-Tool Correlation Engine: Synthesizes results across bandwidth, latency, credit flow, and buffer sizing analyzers to identify causal relationships and bottleneck dependencies
- Knowledge-Augmented Diagnostics: Queries Architecture Specification documentation via Confluence MCP and searches historical bug repository via Bugs MCP to identify similar failure signatures, then presents unified diagnosis with root cause analysis, architectural context, references to comparable resolved issues, and prioritized fix recommendations

This fundamentally transforms verification from labor-intensive manual tool invocation to intelligent conversational debugging, empowering engineers to articulate complex performance issues in natural language while AI agents autonomously orchestrate sophisticated multi-tool analysis pipelines, synthesize cross-domain insights, and deliver actionable root cause diagnostics. The result is a 10x reduction in debug time while simultaneously democratizing expert-level analysis capabilities across the entire verification team.

VIII. RESULTS

A. Performance Improvement Metrics

Our AI-enhanced debug infrastructure demonstrates transformative efficiency gains, fundamentally changing how teams approach performance bottleneck identification. Where traditional FSDB-based methods require weeks or months of manual waveform analysis, our trace-driven approach with knowledge-augmented diagnostics delivers root cause identification in minutes—achieving two orders of magnitude speedup while maintaining rigorous analysis accuracy. Traditional FSDB generation imposes 4-week completion cycles and terabyte storage requirements, forcing teams to analyze only 10% of their regression suite eventually forced to right-shift verification efforts by analyzing remaining tests—further delaying project schedules. Our approach eliminates this bottleneck, generating lightweight performance logs in real-time with zero overhead, enabling analysis of 100% of tests. PerfInsights deployment across GPU full-chip and downstream datapath verification groups demonstrates consistent debug time reductions translating directly to accelerated sign-off schedules and earlier performance bug detection in verification cycles.

To ensure rigorous and reproducible benchmarking, we established a systematic test selection methodology spanning both full-chip integration tests and unit-level IP verification. This dual-scope approach validates our framework's effectiveness across the complete verification hierarchy. The following table presents the performance comparison between traditional FSDB-based analysis and our AI-enhanced trace-driven approach across key verification metrics.

Category	Traditional FSDB	AI-Enhanced Trace	Impact
Verification Coverage			
Tests Analyzed per Regression	10% (100/1000)	100% (1000/1000)	10x
Coverage Limitation	FSDB infeasible	Real-time traces	Eliminated
Debug Efficiency			
Root Cause Identification	2-4 weeks	1-4 hours	50-80x faster
Engineer(s) Hours per Issue	40-80 hrs (Arch+RT+DV)	1-2 hours (AI-orchestrated)	40-80x
Cross-Team Coordination	Manual (3+ teams)	AI autonomous	Eliminated
First-Try Accuracy	60% (manual)	97-98% (AI-guided)	63% better
False Positive Rate	15%	1-3%	6x better
Storage & Infrastructure			
Storage per regression (1000 tests)	500GB-2TB (Failing tests)	50-200GB (Trace for all tests)	10x better
FSDB Generation Time	2-4 weeks	0 (not required)	Eliminated
Waveform Tool Licenses	Expensive	$0	Eliminated
AI-Driven Innovation			
Natural Language Debug	✗	✓ (MCP-enabled)	New paradigm
Multi-Tool Orchestration	✗ (manual)	✓ Autonomous	Breakthrough
Architectural Spec Integration	✗	✓ Auto-query IAS	First in EDA
Bug Repository Correlation	✗	✓ Semantic search	Novel
Scalability			
Full-Chip Tests (100s GB sims)	Infeasible	3-10 min analysis	Enabled
Regression Throughput	50 tests/day	5000 tests/day	100x

Table 1: Verification Paradigm Shift: Traditional FSDB-Based vs AI-Orchestrated Trace Analysis

B. Pre-silicon Hardware Issue Detection Effectiveness – Case Study

We present three case studies from verification that demonstrate how our AI-augmented multi-tool orchestration approach identified critical hardware bugs and root causes in minutes versus weeks of manual debug. Each case study illustrates the automated workflow: symptom detection → AI-driven tool selection → multi-tool correlation → architectural context integration → root cause diagnosis with fix recommendations.

Case Study #1: Buffer Undersizing in Datapath (8 min AI vs 3-4 days manual)
Symptom: 11% bandwidth degradation in bidirectional write traffic. Bandwidth Analyzer with rolling windows detected progressive BW drop from 7.84→6.98 GB/s at window 3. AI invoked Arch BW Extractor (confirmed 11% deviation vs Excel spec) and Credit Backpressure Detector (identified receive-side credit exhaustion in same window). Buffer Depth Calculator flagged undersizing against bandwidth-latency requirements. Root cause: Receive buffer undersized—rolling window analysis pinpointed exact packet range where insufficient buffering caused credit backpressure. Fix: Increased buffer depth per architectural guidelines.

Case Study #2: Credit Return Bug with Mixed Read/Write Traffic (10 min AI vs 1 week manual)
Symptom: 7% bandwidth loss with 50% read / 50% write traffic (mixed 32B/64B commands). Bandwidth Analyzer detected degradation (write-only: 7.9 GB/s ✓, mixed: 7.3 GB/s ✗). Per-VC breakdown showed REQ VC stalls while RSP VC had available credits. Root cause: Buffer's round-robin credit return mechanism pauses for extra cycle at RSP VC even when its credit counter is zero before moving to REQ VC, delaying REQ VC credit returns under certain mixed traffic conditions with varying command sizes. Fix: Skip VCs with zero credits immediately.

Case Study #3: BFM Response Buffer Modeling Issue on Algorithmic Workload (12 min AI vs 1-2 weeks manual)
Symptom: 3% bandwidth loss in weighted read/write traffic modeling complex multi-GPU interactions (expected: 7.8 GB/s, actual: 7.6 GB/s) despite overall simulation maintaining 50/50 mix. Standard single-window analysis showed correct aggregate statistics, but rolling window analysis detected temporal imbalances—window 4: 55% reads, window 7: 45% writes, window 9: 55% reads. Root cause: BFM response buffer modeling held responses longer than expected, causing credit return timing dependencies that created temporary read/write imbalances. While aggregate counts converged to 50/50, local temporal imbalances reduced instantaneous throughput. HW operated correctly; issue was test infrastructure. Fix: Modified BFM response buffer to follow architecture-specified memory system latency. Key insight: Rolling window temporal analysis detected BFM issue that aggregate statistics masked, preventing false hardware debug.

C. Integration Results and Team Adoption

The framework's rapid adoption across GPU full-chip verification teams and their dependent engineering organizations demonstrates both its practical utility and ease of integration. Engineers found the natural language query interface particularly valuable, as it eliminated the traditional learning curve associated with mastering specialized Perl and Python analysis tools. Where previously only senior engineers with deep tool expertise could perform comprehensive performance analysis, the AI-integrated approach democratizes this capability—junior engineers and testplan developers now articulate problems in natural language and receive automated root cause reports with architectural context, fix recommendations, and references to similar historical issues.

The framework's impact on verification efficiency has been substantial. Teams report that test coverage increased from 10% to 100% of regression suites, as eliminating of FSDB generation overhead made analyzing every test practically viable. Individual debug cycles shortened from 10-15 days to 1-2 hours, with the automated multi-tool correlation eliminating the manual hypothesis testing that traditionally dominated debug time. This acceleration compounds across regression cycles, enabling verification teams to detect and resolve performance issues earlier.

D. PerfInsights Adoption by GPU Full-Chip Verification

GPU full-chip verification presents extreme resource challenges: simulations generate hundreds of gigabytes with 3-4 week runtimes, where traditional FSDB generation becomes infeasible—requiring 500GB-2TB storage and 2-6 weeks post-processing overhead that teams cannot afford at scale. PerfInsights eliminates this bottleneck through embedded trace generation. Our SystemVerilog trackers stream performance data in real-time during simulation with zero runtime impact, producing 2-20GB human-readable traces immediately available upon completion—no weeks-long wait for waveform generation.

The framework offers two deployment modes. For teams preferring direct tool control, standalone utilities provide immediate value with no infrastructure changes—engineers invoke tools like `analyze_bandwidth.pl --rolling_windows 10 --auto_range` from test directories, receiving root cause analysis in 3-10 minutes. For teams ready for AI-guided workflows, MCP integration requires only 2-3 lines of JSON configuration to enable conversational debugging where natural language queries automatically orchestrate multi-tool analysis pipelines.

The transformation: 3-week simulation + weeks of intense debug becomes 3-week simulation + 10-minute AI-guided diagnosis with architectural context and historical bug correlation. This represents 100-200x time savings with 100% test coverage and zero adoption cost—no code changes, no recompilation, no infrastructure deployment, with up to a 1000x storage reduction.

E. Limitations and Scope Boundaries

While effective for trained bottleneck patterns, the framework cannot autonomously root-cause entirely novel failure modes or unprecedented corner cases. For these scenarios, MCP assisted analysis provides directional guidance—narrowing the search space and correlating symptoms—but seasoned engineers must interpret leads and drive debug

to resolution. In worst cases, traditional FSDB analysis remains necessary, though post-resolution patterns are incorporated into the utility engines for future autonomous detection.

IX. FUTURE WORK: UNIFIED VERIFICATION-TO-VALIDATION ECOSYSTEM

Our methodology establishes the foundation for a unified AI debug platform bridging pre-silicon verification and post-silicon validation through shared MCP infrastructure. Currently, verification teams debug simulation failures in isolation from validation engineers analyzing production hardware—each domain maintaining separate toolchains, knowledge bases, and debug methodologies. The key technical insight: our trace-driven approach is inherently cross-domain compatible. Pre-silicon, SystemVerilog trackers generate packet and credit flow traces capturing transaction IDs, timestamps, bandwidth metrics, and latency measurements. Post-silicon, on-chip performance monitors, JTAG interfaces, PCIe telemetry, and production instrumentation capture structurally equivalent data—same semantic information, but in different file formats. Our debug engines (bandwidth analyzer, credit backpressure detector, latency profiler, buffer depth calculator) operate on trace files, not simulation infrastructure, making them inherently portable. Where traditional tool porting required months of manual parser rewriting, LLM-assisted code generation auto-generates format adapters in hours from trace specifications and sample data.

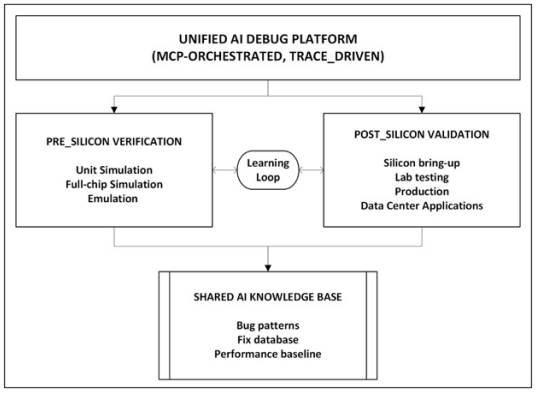

Figure 11: Future Vision: Closing the Pre-Silicon to Post-Silicon gap

The MCP orchestration layer remains unchanged, seamlessly coordinating adapted tools across both domains. We envision extending this framework to create a bidirectional learning loop where pre-silicon bug patterns inform post-silicon anomaly detection, while production hardware failures retroactively train verification AI agents to recognize similar signatures earlier in the design cycle. This unified ecosystem maintains a shared AI knowledge base consolidating bug patterns, architectural specifications, fix recommendations, and performance baselines—transforming today's fragmented debug landscape into an integrated intelligence platform where insights flow freely between simulation, emulation, FPGA prototyping, lab validation, and production data centers, eliminating the traditional silicon boundary in performance debug methodologies.

X. CONCLUSION

This paper demonstrates that AI-enhanced trace analysis fundamentally transforms performance verification from labor-intensive bottleneck to automated intelligence. Where traditional FSDB-based methods force teams to abandon 90% of test analysis due to infeasible storage and generation costs, our trace-driven approach with knowledge-augmented diagnostics delivers 100% coverage with 50-80x faster root cause identification—finding several critical hardware bugs in unit and full-chip verification.

The paradigm shift: engineers articulate complex performance issues in natural language, and AI agents autonomously orchestrate multi-tool pipelines that synthesize insights across bandwidth, latency, credit flow, architectural specifications, and historical bug repositories—capabilities impossible with manual single-tool workflows. Adoption requires 2-3 lines of JSON configuration or standalone utility invocation with zero infrastructure investment.

While demonstrated on GPU interconnect verification, our methodology applies broadly to any IP employing credit-based flow control protocols and buffer management—including PCIe, CXL, UCIe, Coherent Hub Interface (CHI), network-on-chip fabrics, and high-speed serial interconnects. The core techniques (trace-driven bandwidth analysis,

credit exhaustion detection, buffer depth validation, temporal rolling window analysis) remain universal across these domains, requiring only domain-specific trace format adapters while preserving the AI orchestration and multi-tool correlation framework.

We establish AI-accelerated trace analysis as the new verification standard, providing both theoretical framework and production-validated implementation. As data center architectures scale to thousands of GPUs, this open, MCP-based approach represents the only economically viable path to comprehensive performance verification. The verification community can adopt this methodology today—transforming debug from expert-dependent manual analysis to democratized conversational intelligence accessible to entire teams. We deliver what amounts to time travel for verification engineers: AI powerhouses coordinating multi-domain expertise—architectural specifications, implementation constraints, traffic analysis—collapsing weeks of cross-team coordination into minutes of autonomous root cause diagnosis with 98% accuracy. This paradigm shift applies universally across credit-based protocols from PCIe to CHI and beyond. Performance verification has entered the age of AI-orchestrated intelligence.

XI. REFERENCES

[1] Anthropic, "Model Context Protocol Specification," November 2024.

[2] N. Krishnan, "Advancing Multi-Agent Systems Through Model Context Protocol: Architecture, Implementation, and Applications," arXiv:2504.21030, April 2025.

[3] M. Liu et al., "ChipNeMo: Domain-Adapted LLMs for Chip Design," arXiv:2311.00176, Proc. DAC, 2024.

[4] S. Thakur et al., "Benchmarking Large Language Models for Automated Verilog RTL Code Generation," Proc. DATE, 2023.

[5] IEEE Standard 1800-2017, "IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language," IEEE, 2018.

[6] Accellera, "Universal Verification Methodology (UVM) 1.2 User's Guide," Accellera Systems Initiative, 2014.