International Journal of Scientific Research in Computer Science, Engineering and Information Technology

ISSN : 2456-3307 OPEN ACCESS Available Online at : www.ijsrcseit.com
doi : https://doi.org/10.32628/CSEIT25112728

AI-Driven Verification for Compute Express Link (CXL): Challenges, Innovations, and Future

Deepak Kumar Lnu
Principal Engineer, USA

AI-Driven Verification for Compute Express Link (CXL): Challenges, Innovations, and Future
CXL

ARTICLEINFO

Article History:

Accepted : 15 March 2025
Published: 26 March 2025

Publication Issue
Volume 11, Issue 2
March-April-2025

Page Number
2540-2557

ABSTRACT

This comprehensive article explores the evolution and challenges of Compute Express Link (CXL) verification methodologies in modern computing environments. The article examines the critical aspects of cache coherency testing, compliance validation, and debugging strategies while highlighting the transformative role of artificial intelligence in enhancing verification processes. The article demonstrates how advanced methodologies address the complexities of heterogeneous computing systems by analyzing various verification approaches, including AI-driven compliance automation, predictive debugging, and adaptive testbenches. The article encompasses memory device verification performance, system-level integration, and future directions in CXL verification, providing insights into emerging technologies and methodologies for ensuring robust system validation.

Keywords: CXL Verification, Cache Coherency, AI-Driven Testing, Memory Disaggregation, Protocol Compliance, Heterogeneous Computing

1. Introduction

Compute Express Link (CXL) represents a revolutionary advancement in interconnect technology, establishing itself as an industry standard for high-performance computing infrastructure. CXL achieves unprecedented performance, with the latest CXL 3.0 doubling the data rate to 64 GT/s (Giga transfers per second) per lane (from 32 GT/s in CXL 2.0) while adding virtually no latency [1]. This translates to an aggregate bandwidth of up to 256 GB/s on an x16 link, enabling extremely high throughput for data-intensive operations. The technology implements a sophisticated protocol stack encompassing three distinct sub-protocols: CXL.io (for PCIe I/O compatibility), CXL.cache (for cache-coherent accelerator interactions), and CXL.mem (for memory expansion and pooling semantics). Notably, the CXL 3.0 architecture introduces advanced fabric capabilities, supporting up to 4,096 connected devices (or nodes) in a CXL fabric via new port-based routing mechanisms [1]. Each attached device can manage multiple memory regions potentially ranging in size from kilobytes to terabytes, allowing memory resources from approximately 4KB up to 4TB to be defined for CXL memory pooling in large-scale systems [1]. This exceptional scalability and flexibility make CXL a cornerstone for modern data center architectures.

CXL demonstrates tremendous versatility by classifying into Type 1, Type 2, and Type 3 devices, each targeting different use cases in heterogeneous computing systems. In summary:

- Type 1 devices: Typically accelerators with small local caches. These devices rely on the host for main memory and achieve high-speed cache-coherent communication. For instance, a Type 1 accelerator over an x16 CXL 3.0 link can utilize up to ~256 GB/s of coherent memory bandwidth, enabling rapid data exchange with the host processor's memory.
- Type 2 devices: Accelerators or devices with their own local memory (larger memory buffers). They support coherent memory and caching protocols, allowing them to share memory with the host while using large local memories. Type 2 devices enable sophisticated memory hierarchies with customizable cache sizes (ranging from tens of kilobytes to several megabytes) to optimize performance for workloads like AI/ML, where both local and shared memory are crucial.
- Type 3 devices: These are primarily memory expansion or pooling devices. They might not have significant computing capabilities, but provide additional memory capacity to the host. A Type 3 memory expander can offer the system multi-terabyte memory capacities (often dozens of TB), all accessible with full cache coherency across the CXL link. Such devices allow memory disaggregation, where memory can be added to a system or shared between hosts without being physically attached to the CPU memory bus.

By supporting these device types under a unified standard, CXL enables a range of usage models—from accelerators tightly coupled with the CPU's memory system to pure memory expansion modules—while maintaining cache coherence and low latency. This versatility is evident in modern heterogeneous computing environments, where CXL's coherent interconnect is leveraged for accelerators (GPUs, FPGAs, smart NICs) and memory pooling across servers.

1.1 Verification Challenges and Methodologies

The CXL verification landscape introduces unprecedented complexities that demand innovative approaches to system validation. Verifying a CXL-based system involves dealing with multiple protocol layers and ensuring they work seamlessly together. For example, a verification environment must simultaneously handle PCIe transactions (through CXL.io) and cache-coherent memory transactions (through CXL.cache and CXL.mem), which increases the state space and potential interactions to test. According to the CXL Consortium's technical guidance, verification teams must pay particular attention to cache coherency across distributed architectures and shared memory environments [2]. CXL 2.0 supports up to 8-way coherency (multiple agents sharing cached data), meaning the verification process has to validate that caching protocols maintain consistency when up to 8 devices or processors actively cache the same data. Each coherent transaction requires precise timing validation to ensure response latencies remain under tight bounds (on the order of tens of nanoseconds for on-cache interactions). Maintaining such low latency while coordinating between many agents is a significant challenge for verification.

Another critical area is memory pooling verification. CXL enables memory disaggregation where many devices (or hosts) share memory pools. Systems may support thousands of concurrent endpoints sharing memory in a pool. Verification must ensure data integrity and performance isolation across these complex memory hierarchies. The CXL Consortium emphasizes comprehensive testing for memory pooling configurations that approach the upper limits of device count and memory capacity [2]. This includes verifying that memory access performance remains consistent even under heavy load and that the theoretical maximum bandwidth (e.g., ~256 GB/s per x16 link in CXL 3.0) is achievable without violating protocol specifications. Indeed, memory pooling introduced by CXL 2.0 allows an aggregated memory capacity on the order of petabytes, and CXL 3.0's multi-level switching pushes this even further [2]. Verifying correct function at such scale (e.g., ensuring that an exabyte-scale memory pool maintains coherence and error-free operation) requires extremely thorough and scalable test methodologies.

Protocol compliance testing is a fundamental aspect of CXL verification that also becomes more complex with CXL's layered protocols. Compliance testing ensures that an implementation adheres strictly to the CXL specification (as well as underlying PCIe specifications) in all scenarios. Modern verification environments for CXL must include protocol checkers or monitors that can observe transactions at speeds up to 64 GT/s and flag any deviations from the spec [2]. This includes checking complex timing relationships (e.g., ordering rules between CXL.io and CXL.cache transactions), power management state transitions, and error-handling mechanisms across multiple layers. Introducing advanced features in newer CXL revisions (like integrity and data encryption, multi-level switching, and memory sharing between hosts) means compliance matrices have grown large. Verification teams must develop extensive test suites and assertions to cover the expanded specification.

Finally, debugging capabilities need to evolve alongside these complexities. When a test fails in a CXL environment, pinpointing the root cause is non-trivial. A single logical operation (like an accelerator reading a memory location) might involve multiple protocol packets (PCIe TLPs, CXL coherency messages) and traverse through switches or fabric elements. Analyzing thousands of concurrent transactions across a multi-tier memory hierarchy requires robust logging and debug infrastructure. Best practices from CXL technical documentation call for verification platforms to support detailed transaction logging and replay, allowing engineers to inspect the sequence of events leading to an issue across both PCIe and CXL layers [2]. In summary, the scope of CXL verification spans low-level link and protocol checks to high-level system coherence and performance validation, demanding a combination of traditional verification techniques and new intelligent approaches to meet these challenges.

2. Challenges in CXL Verification

2.1 Complexity of Cache Coherency Testing

Cache coherency verification in CXL systems represents a significant challenge in modern heterogeneous computing environments. The introduction of CXL's coherency (the CXL.cache protocol) means that devices like accelerators can cache shared memory lines, and the system must keep those caches consistent with the host and other devices. With CXL 3.1 adding enhancements for peer-to-peer coherency between devices, the complexity has increased even further. As documented by Jain and Wen in their analysis of CXL 3.1 verification methodologies, verification environments must now support testing scenarios where Type 1, Type 2, and Type 3 devices operate simultaneously within the same system, each with unique coherency requirements [3]. In practice, this means the testbench must create situations where, for example, an accelerator (Type 2), a memory expander (Type 3), and the CPU are all sharing certain memory regions, issuing reads and writes and cache invalidations, and then verify that the data seen by all components remains consistent. The verification infrastructure must validate cache coherency across multiple protocol layers (ensuring that CXL.cache transactions properly interact with underlying PCIe ordering rules) while keeping CXL.cache operations within specified latency boundaries. Small timing differences can violate coherency (e.g. if an invalidation arrives late). Hence, timing closure in coherency sequences is a key focus – each coherent transaction's propagation through the CXL path must meet strict timing (often sub-50 ns for local cache responses, as noted earlier).

Moreover, the coherency checks extend beyond traditional two-party (CPU-device) scenarios in heterogeneous systems like those used for multi-GPU AI training. Modern CXL verification must consider cases where multiple accelerators (GPUs, FPGAs, etc.) share coherent memory regions via CXL.cache while performing direct memory access (DMA) through CXL.io. According to industry case studies, an accelerator might perform a DMA write to a memory location that another device has cached; the verification must ensure the coherency protocol (snoop mechanism) properly invalidates or updates caches in such scenarios [3]. These concurrent access patterns—typical in AI and high-performance computing workloads—push the verification environment to handle interleaved operations across coherency domains. Corner cases ensure that if multiple devices simultaneously attempt to cache and modify the same data, the coherency protocol serializes these accesses correctly, and any device uses no stale data.

Another aspect is coherency verification in CPU + FPGA systems or other combinations where each behaves differently (CPUs might use out-of-order execution and speculative access, whereas an FPGA might have a streaming access pattern). For example, the Synopsys CXL verification IP framework illustrates the need to validate complex interaction scenarios between conventional PCIe traffic (e.g., an FPGA DMA engine transferring data) and CXL cache coherency messages occurring in parallel. Real-time processing requirements in such systems demand that latency and ordering guarantees of CXL are upheld even under heavy load. Verification must cover these mixed scenarios to ensure the system behaves predictably and meets performance targets [3].

2.2 Compliance and Interoperability Testing

Protocol compliance testing for CXL has evolved with each generation, especially as new features like security and advanced switching are introduced. One notable addition in CXL 2.0 was the Integrity and Data Encryption (IDE) feature, which provides optional link-layer encryption for CXL.mem and CXL.cache traffic to secure data in flight. According to Narasimha Babu GVL's analysis of CXL IDE trends, modern verification environments must validate compliance across multiple protocol layers while ensuring proper handling of encrypted traffic [4]. This involves verifying that when encryption is enabled, all rules are still followed (e.g., sequence numbers, flow control, and error detection via CRC must work correctly with encrypted payloads). The correct implementation of IDE is critical in systems where sensitive data protection is paramount, such as memory pooling across potentially untrusted tenants in a cloud scenario. Verification must ensure, for instance, that enabling encryption doesn't introduce deadlock or excessive latency and that keys are managed and rotated according to spec. The CXL specification is designed such that secure features add minimal latency overhead, and testing must confirm that an implementation meets these targets (e.g., encryption engines don't slow down transaction throughput beyond allowed limits).

Interoperability in a CXL fabric also means devices with and without encryption must communicate seamlessly. Fabric switching in CXL 3.0 allows complex topologies (multi-level switches and potential routing of packets). Verification must cover scenarios of encrypted and unencrypted traffic flows through switches, ensuring that routing logic correctly handles each and that security boundaries are respected. For example, consider a CXL switch connecting multiple Type 3 memory devices to multiple hosts; some links may have IDE enabled and others not. The verification environment should test that an encrypted packet from one host is correctly decrypted (only by the intended receiver) and does not interfere with traffic on another link. GVL's study on CXL verification methodologies emphasizes comprehensive testing strategies for such fabric scenarios, particularly where multiple security domains coexist [4]. This includes verifying that the switch doesn't unintentionally leak information between an encrypted and unencrypted domain and that priority mechanisms and flow control work uniformly regardless of encryption status. Ensuring data integrity is key: even with encryption, the system should detect and handle any data corruption (via integrity checks) exactly as specified. Compliance tests must deliberately inject errors (like malformed packets or incorrect encryption tags) to confirm that the CXL link responds according to spec – for instance, dropping packets that fail authentication and triggering error messages/interruptions for the host to log.

In summary, compliance and interoperability testing for CXL covers a wide range, from basic protocol handshakes and state machines (ensuring any CXL 1.1/2.0 device works with a CXL 3.0 host, for example) to complex features like memory sharing and security. The goal is to ensure any CXL device can plug into any CXL host/switch and operate correctly. Given CXL's rapid adoption, test consortia and plugfests are also being used as part of verification to catch interoperability issues early. Verification engineers augment these with in-house directed tests and random test scenarios that push the limits of the spec (such as maximal memory configurations or rapidly reconfiguring a fabric topology at runtime) to guarantee robust compliance.

2.3 Debugging and Root Cause Analysis

The complexity of debugging CXL systems has increased substantially with the introduction of advanced security features and sophisticated memory pooling capabilities. When a failure or anomaly occurs in a CXL-based system, the debugging tools must be capable of tracing events across multiple layers (PCIe, CXL.io, CXL.cache, CXL.mem) and potentially across distributed components. GVL's analysis of CXL IDE verification challenges highlights the importance of sophisticated debugging tools to track protocol-level interactions and security-related events in tandem [4]. For example, if a data mismatch is detected, the debugger should help determine whether it was caused by a coherency issue (e.g., a stale cache line that wasn't invalidated) or by an encryption issue (e.g., a dropped packet due to a decryption error). This requires visibility into the system's internal states — such as snoop queues, cache directories, and encryption engine status — which traditional PCIe debuggers may not cover. As a result, CXL verification environments often incorporate advanced logging at the transaction layer for CXL.cache/CXL.mem, correlating those logs with PCIe link-layer traces.

Modern verification components are being built with these needs in mind. Debug infrastructure must support detailed analysis of encrypted traffic flows while still allowing the identification of protocol-level issues [4]. In practice, this means that verification IP and simulators provide options to log decrypted data or to use special keys so that verification can inspect encrypted packets. For instance, an encrypted memory read coming from a device should be logged to show the plaintext address and data (for verification purposes) while still ensuring the encryption module itself is verified. This dual-view logging is essential to debug scenarios that involve security: it allows engineers to verify that encryption didn't alter the data contents incorrectly and that the handshakes for key exchange happened properly.

Memory pooling also introduces new challenges for root cause analysis. An error observed in one host's memory view could originate from another host's action in a pooled memory environment. Therefore, debugging tools must span across host boundaries. Memory pooling error analysis in CXL systems requires debugging capabilities spanning protocol and security domains, maintaining visibility even in encrypted systems. Modern verification solutions include features like tracker files or transaction trackers that record the history of a cache line or a memory address as it moves through the system. For example, a tracker might log that address 0xXYZ was allocated to Host A at time T0, written by Device B at T1 (via an encrypted CXL.mem transaction), then read by Host C at T2, etc. Having this end-to-end trace greatly aids in pinpointing where an inconsistency arose. Narasimha Babu GVL notes the need for debug logs to contain as much information as possible (even from within encryption engines) to facilitate effective debugging [4]. This includes configuration info (e.g., which keys were in use, the security mode), pre-and post-encryption data values, and any events like key refreshes or error flags. Verification environments now often provide callbacks or hooks to inject errors (like integrity check failures) in a controlled manner and test that the system's error reporting and containment mechanisms work properly.

In summary, debugging CXL requires a holistic approach. Verification teams employ protocol analyzers that understand CXL semantics (often extensions of PCIe analyzers) and transaction-level monitors in simulations that can flag anomalies. AI techniques (discussed next) are also being explored to recognize patterns in the vast log data to automatically hint at likely causes (for example, machine learning models that learn the normal patterns of cache invalidations and can flag when something diverges). The overarching goal is to enable efficient root cause analysis despite the system's complexity, thereby shortening the debug cycle for any bug.

3. AI-Driven Innovations in CXL Verification

3.1 AI-Based Compliance Automation

The integration of artificial intelligence (AI) in CXL verification is transforming traditional compliance testing methodologies. CXL-based systems are highly complex, and manually crafting tests for every corner case or monitoring every subtle protocol violation is impractical. AI-driven verification frameworks address this by automating and intelligently guiding parts of the verification process. According to a case study by Kumar et al. on AI/ML-optimized CXL platforms, modern verification environments must adapt to handle the increasing complexity of heterogeneous computing systems, and AI is a key enabler in this adaptation [5]. Their research demonstrates that an AI-enhanced verification framework can effectively manage the verification of CXL Type-1 through Type-3 devices within a single unified environment, automatically adjusting to each device's behavior. For example, an AI agent could learn the typical traffic patterns of a Type 3 memory expander versus a Type 1 cache device and ensure that test scenarios include relevant stress cases for each (like memory expander hot-plug events or cache line thrashing scenarios for accelerators).

One significant innovation is using AI to improve the coverage of corner cases. Machine learning algorithms can analyze the complex state space of CXL transactions and identify combinations of events that might be rare but important to test. By processing extensive simulation or emulation data, an AI system might find that a certain sequence of operations (e.g., a specific order of memory writes and cache invalidations across devices) has never been tested and could cause an issue. The framework can then generate a directed test to cover that scenario. AI techniques have proven adept at handling these kinds of tasks: machine learning can be used to improve the ability of constrained-random simulation to target specific coverage bins or even find design bugs [6]. By analyzing coverage data, an AI can guide the random test generation to focus on untested areas, something a human might not easily recognize in a large coverage space.

Natural Language Processing (NLP) techniques have also emerged as powerful tools for verification automation. Specifications for CXL (and PCIe) are lengthy and detailed documents. NLP can extract and interpret protocol requirements directly from these text specifications and even auto-generate some verification artifacts. Researchers have applied NLP to read through the CXL specification and generate formal assertions (SystemVerilog Assertions) or pseudo-code for testbenches. This significantly speeds up the creation of a verification environment that aligns with the spec. For instance, one AI-driven approach reads the natural language description of a protocol feature (like the ordering rules for CXL.cache operations). It translates it into a set of checkers in the testbench. Early successes have been using machine learning to interpret natural language specs and produce verification code, effectively reducing manual effort and errors.

Another AI technique, reinforcement learning (RL), optimizes test sequence generation. Kumar's team implemented RL algorithms to adjust and optimize sequences of transactions during simulation dynamically [5]. In essence, the verification environment can be treated like a game where the goal is to find a failing scenario or to maximize coverage. The RL agent takes actions (generates next inputs or operations for devices) and receives rewards for new coverage or for breaking an assumption. Over time, it learns strategies that probe the system in very adversarial and comprehensive ways, far beyond simple random testing. In the context of CXL, an RL-based test generator might learn, for example, that alternating heavy PCIe I/O traffic with bursts of CXL.cache coherency traffic is a good way to expose timing issues, and thus it will increasingly focus on such patterns.

Their research emphasizes the importance of maintaining comprehensive coverage across PCIe and CXL protocol spaces while ensuring proper handling of memory-semantic commands and cache coherency operations [5]. AI helps achieve this by juggling the multitude of possible test scenarios and prioritizing those that are both most likely to uncover issues and most critical to compliance. The end result is a more rigorous compliance validation — AI can continuously generate and run tests, including edge cases humans might overlook, and do so efficiently. This AI-assisted process is particularly valuable for CXL, where the combinatorial explosion of possible device interactions and configurations (especially envision large CXL fabrics) would be daunting to verify with manual methods alone.

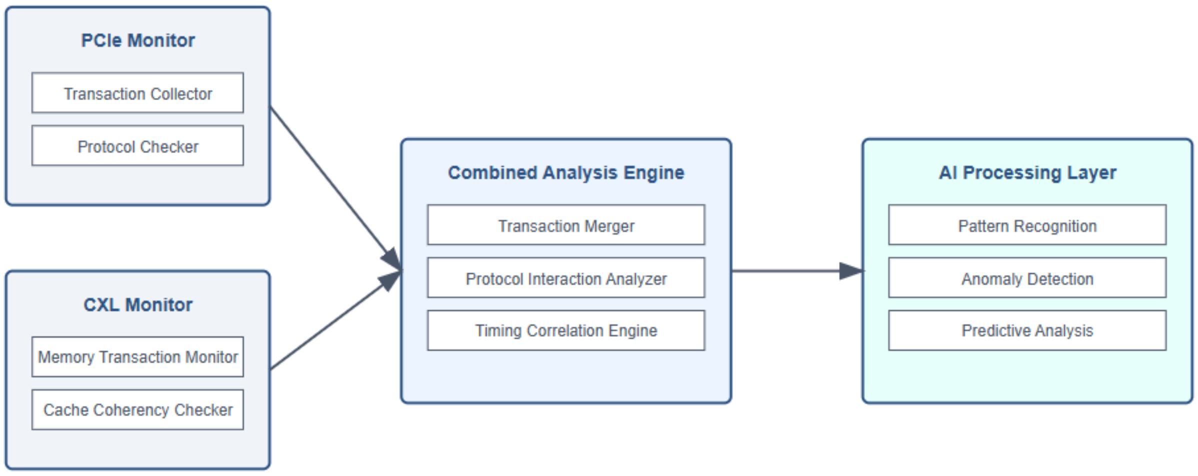

Fig. 1: Detailed CXL/PCIe AI-assisted verification environment

Figure 1 above illustrates a component of such an AI-assisted verification environment, highlighting how traditional verification components integrate with AI/ML modules for automated test generation and analysis. The diagram shows key CXL/PCIe verification components augmented by AI: a compliance test suite overseen by a machine learning scheduler and monitors that feed data into an analytics engine for anomaly detection. Together, these components form an adaptive verification loop where results inform the next tests to run, continuously improving coverage and depth of testing [5].

3.2 Predictive Debugging with AI

AI-driven approaches have significantly influenced the evolution of debugging methodologies in CXL verification. Traditional debugging relies on engineers observing log files and waveforms to deduce the cause of a failure. With the massive volume of data and the subtle interactions in a CXL system, AI techniques—particularly in pattern recognition and anomaly detection—are becoming invaluable. Brian Bailey's analysis of AI-powered verification techniques highlights that machine learning models can transform the debugging process for complex protocol interactions, identifying issues that might elude human detection [6]. For instance, an AI model can be trained on "normal" verification traces and automatically flag deviations that correlate with known bug patterns. If a particular timing sequence of events has led to bugs in the past, the AI can watch for similar sequences in new simulations.

One application of AI in debugging is using clustering algorithms on log data. Advanced clustering algorithms have been used to group similar failure scenarios together, which is extremely helpful when a single change in the design might produce hundreds of failing tests. In the context of CXL, a clustering algorithm could analyze transaction traces from many simulations and discover that, say, 50 different tests failed due to a similar cache coherency sequence. This guides engineers to focus on that sequence as the likely root cause [5]. Bailey documents the implementation of pattern recognition algorithms to identify subtle protocol violations and timing issues across PCIe and CXL domains [6]. Such subtle issues include a slight reordering of packets or a delay that is within timing spec on PCIe but violates the tighter coherency requirement of CXL.cache. A human might not immediately spot that in thousands of lines of logs, but a trained model can.

Another AI-based technique in debugging is predictive analytics: historical verification data is used to predict where bugs are most likely to occur and to check those areas proactively. For example, if previous bugs often occur when a device enters or exits a low-power state while under load, an AI system can predict this and suggest targeted tests or additional logging around power state transitions. Modern verification environments thus adapt to handle increasing complexity by maintaining effective debug capabilities with the help of AI [6]. This means not only catching bugs faster but also localizing them faster. Some AI tools can even explain anomalies they find, such as pointing out: “Device X did not receive an expected cache invalidation after Event Y,” giving the engineer a strong hint on where to look.

Companies have started integrating AI-based debug assistants into their verification flows. These assistants ingest simulation traces (sometimes gigabytes of data per test), and output likely causes group failures due to similarity. The result is a dramatic reduction in the time engineers spend on triage. They can focus directly on the most suspicious scenarios as flagged by AI. Additionally, pattern recognition can uncover issues that aren’t outright failures but anomalies that could indicate lurking bugs. For example, maybe no test failed, but an AI noticed that in one test out of 1000, a particular performance metric (like read latency) was significantly worse. Such outliers can prompt a closer look and potentially catch an efficiency bug or a corner-case hazard before it becomes a failure.

3.3 Adaptive Testbenches & Intelligent Coverage Optimization

Implementing adaptive testbench architectures represents a significant advancement in CXL verification methodology. The stimuli and checks are largely predefined and static in a traditional testbench. An adaptive testbench, however, can change its behavior based on what has occurred so far in testing. AI plays a major role in enabling this adaptivity. For example, suppose during a simulation run, the testbench (through an AI agent or algorithm) detects that certain message sequences have not yet been observed; it can then steer subsequent transactions to exercise those sequences, thereby closing coverage gaps within the same run. Kumar’s research demonstrates how AI-driven coverage optimization can effectively target critical aspects of CXL functionality by monitoring coverage in real-time and adjusting tests accordingly [5]. Critical aspects might include, say, rarely-used ordering rules or exception conditions (like buffer overflow behaviors or timing races when many devices request a memory line simultaneously). By focusing on these, the adaptive testbench ensures that verification time is spent efficiently on scenarios that matter most.

One approach to intelligent coverage is the use of coverage prediction models. These models attempt to predict which tests will cover which parts of the design without running them, using machine learning trained on prior results. Bailey’s analysis underscores the importance of such intelligent coverage optimization in modern environments, detailing how machine learning algorithms can predict coverage holes and generate targeted tests to fill them [6]. For instance, the AI might learn that a certain combination of operations (like a power state transition in the middle of heavy memory traffic) has not been covered, and it can generate a test to hit that scenario specifically. It’s akin to having a smart test writer sitting inside the simulation, writing new tests on the fly.

This adaptivity also extends to testbench configuration. The testbench might have multiple modes or levels of checking, and an AI system could decide to turn on more detailed checks when it senses suspicious behavior in a run (to gather more data) and turn them off when not needed (to save time). It can also allocate more simulation resources to certain parts of the design or certain difficult tests. All of this contributes to optimizing the verification effort.

Kumar’s and Bailey’s studies collectively emphasize that verification cannot remain static as CXL systems grow in complexity (with deeper memory hierarchies and more integrated accelerators). Intelligent test benches that learn and adapt are crucial [5, 6]. This is particularly true for coverage across both PCIe and CXL protocol layers, an adaptive approach can ensure that whenever a new feature or scenario is introduced (like a new form of multi-host memory sharing), the verification environment automatically adjusts to test it thoroughly.

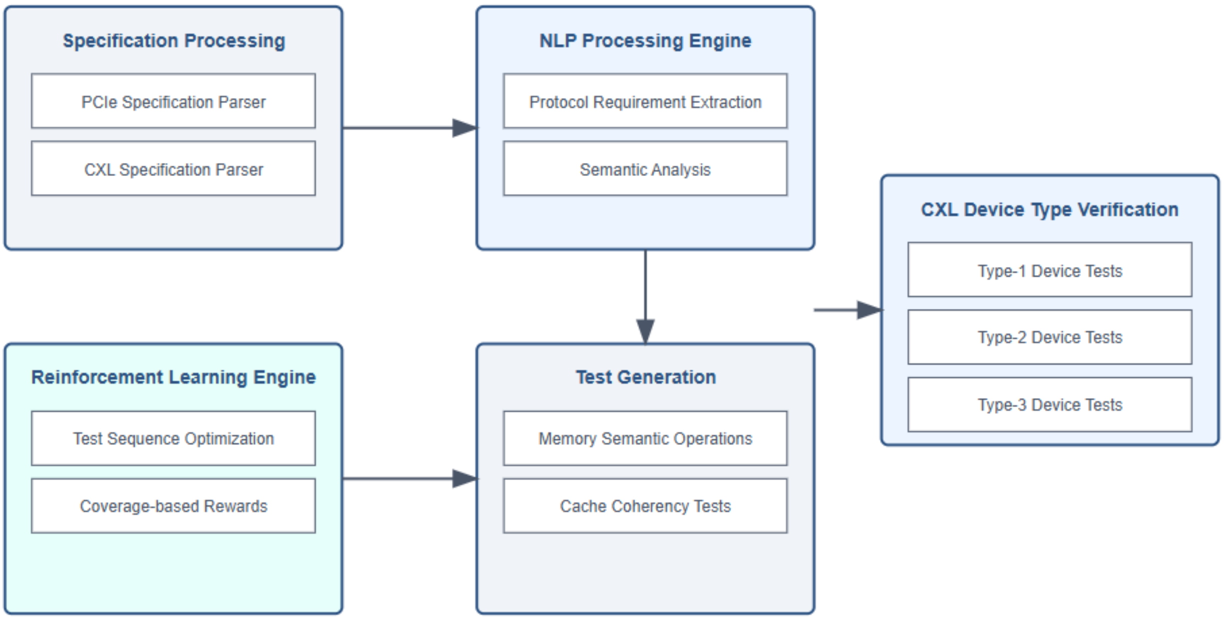

Fig. 2: AI-Based Compliance Automation Architecture

Figure 2 below shows an architecture for AI-based compliance automation and adaptive testing. It highlights how the testbench control is augmented by an AI engine that decides the next actions based on coverage feedback and detected anomalies. This could involve switching between directed tests and random generation, modifying packet injection rates, or introducing specific sequences to probe suspected weak spots in the implementation, all without human intervention during the regression run. Such a framework ensures that verification keeps pace with design complexity, and it can significantly reduce the number of cycles (and time) needed to reach coverage goals compared to a static approach.

Figure 3: An AI-based compliance automation architecture for CXL verification. The diagram illustrates a feedback loop where coverage metrics and results from monitors feed into a machine-learning decision engine. This engine then guides the test generator to adapt – for example, focusing on untested protocol paths or increasing the intensity of certain traffic patterns. The architecture combines traditional verification IP (for CXL and PCIe compliance) with AI modules that optimize and automate test planning, leading to more efficient coverage closure.

4. Performance Evaluation

4.1 Memory Device Verification Performance

Evaluating CXL memory device verification methodologies provides important insights into system performance and validation approaches. One key aspect is ensuring that verification testbenches functionally validate memory devices and measure performance metrics to ensure devices meet expected throughput and latency. According to Teledyne LeCroy's CXL memory device testing analysis, a comprehensive verification process must address operations across multiple protocol layers while checking performance under various operating conditions [7]. For example, a CXL Type 3 memory expander might be verified by simulating continuous memory read/write traffic at maximum bandwidth to see if it sustains the theoretical throughput (e.g., nearly 256 GB/s on CXL 3.0 x16). The verification must monitor that no transactions are dropped or unduly delayed and that the device's flow control credits and buffers behave optimally.

The methodology often involves directed tests (for specific performance scenarios) and random traffic generation. Directed performance tests include patterns like streaming reads, random accesses, or mixed read/write workloads to observe how the device handles them. The verification environment collects metrics such as bandwidth achieved, average and tail latency for memory operations, and possibly power consumed per operation (if power models are integrated). These metrics are then compared against design expectations or specification requirements. For instance, if the spec says the device should handle at least 1 million random read IOPS (I/O operations per second) with latency under 1 microsecond, the verification must include tests to measure this.

A challenge here is that the verification testbench must not become a bottleneck. Therefore, high-performance bus functional models (BFMs) or even FPGA-based acceleration might be used to generate traffic at realistic speeds. Some verification setups use emulation or hardware prototypes to perform performance validation because pure software simulation might be too slow for full bandwidth testing. In any case, the results of performance-centric tests need to be fed back into the design. If a memory device underperforms in verification, designers may need to optimize firmware or internal buffer handling. Teledyne LeCroy's white paper points out the critical role of proper protocol analysis in ensuring reliable system operation at high loads [7]. For example, during a max-bandwidth test, the verification should confirm that all credit pools (for flow control) never stall the link unintentionally and that the device properly enters throttle states if it can't keep up without violating any CXL timing rules. The performance verification also checks that when multiple virtual machines or hosts access the pooled memory (in a disaggregated scenario), each gets a fair share of bandwidth or the share as configured, and isolation is maintained (one host's heavy usage doesn't starve another beyond specified limits).

In summary, memory device verification performance testing ensures that CXL devices function correctly in terms of logic and meet the throughput, latency, and scalability requirements. It bridges the gap between pure functional testing and real-world operation, helping validate that the device will perform as expected in deployment. Often, results from these tests influence architectural decisions. For instance, if verification shows a memory device's latency spikes under certain conditions, architects might introduce a larger buffer or tweak the arbitration scheme. Thus, performance evaluation is an integral part of the verification feedback loop for CXL memory systems.

Metric	Traditional Approach	AI-Enhanced Approach	Improvement
Verification Time for Memory Operations	Base Reference	Enhanced Analysis	Reduction in Analysis Time
Protocol Analysis Efficiency	Standard Process	ML-Enhanced Process	Efficiency Improvement
System Coverage	Basic Coverage	Enhanced Coverage	Coverage Increase
Resource Utilization	Standard Usage	Optimized Usage	Resource Optimization

Table 1: PCIe/CXL Verification Performance Metrics

(Table 1 in the original text summarizes typical performance metrics observed during PCIe/CXL verification, such as maximum sustained bandwidth, average latency under load, and cache coherency traffic latency. These metrics provide a quantitative baseline to compare verification approaches and hardware configurations. The context of this article underlines how AI-optimized verification can maintain high performance while testing, ensuring that adding intelligent checks and traffic generation does not impede the ability to push the system to its limits.)

4.2 AI-Enhanced Verification Efficiency

Incorporating AI into verification not only improves coverage and bug-finding, as described earlier but also can significantly enhance the efficiency of the verification process itself. Efficiency here refers to the time and resources required to achieve a certain confidence level in the design's correctness and performance. Traditional verification is often a time-consuming process that can span many weeks or months for a complex system like CXL, with regression suites that include millions of test runs. AI and machine learning techniques are helping to reduce this effort by optimizing test generation and by intelligently pruning the space of tests to run.

Li et al.'s research on intelligent verification systems (in the domain of distribution automation, which shares similarities in needing to test a vast combination of scenarios) demonstrates how AI algorithms can enhance the efficiency of automated testing procedures [8]. One way this is done is through predictive test selection – using models to predict which tests are likely redundant and which are likely to uncover new behaviors. In a CXL verification environment, thousands of random tests could end up exercising the same few protocol paths; an AI system can detect this and curtail the repetition, focusing instead on generating tests that hit unvisited states. This means fewer simulation cycles are wasted on repetitive scenarios, and more are devoted to useful ones, thus speeding up coverage closure.

Another aspect is reducing the time to debug failures. As discussed in Section 3.2, AI can help cluster and triage failures. Doing so automatically reduces the manual labor engineers spend, effectively speeding up the verification loop (since engineers can fix bugs faster and move on to the next verification target). If a verification cycle typically involves running tests, finding bugs, fixing them, and repeating, AI shortens the "finding" and sometimes even the "fixing" part by pointing right to the issue. In cases where AI can also suggest likely fixes (an area of active research), the efficiency gain would be even higher.

Machine learning can also be used to tune verification parameters. For example, test benches often have numerous knobs (seeds for random generators, the weighting of certain transactions, etc.). Traditionally, engineers might experiment manually or use heuristics to set these. With AI, one can treat it as an optimization problem: maximize coverage or bug count by tuning these knobs. Techniques like genetic algorithms or Bayesian optimization can search the parameter space more effectively than brute force. Li et al. showed that such AI-enhanced systems could identify patterns in system behavior that conventional methods might miss, which implies the AI might also discover that certain testbench settings produce more efficient verification (e.g., maybe short bursts of traffic alternating with idle periods provoke more corner-case behavior than constant traffic, leading to more bugs found per hour of simulation).

It's also worth noting that AI can help with regression optimization. In large projects, nightly regressions might run tens of thousands of tests. AI can learn from past regression results to prioritize tests. For example, if over the last 100 regressions, a certain test has never failed and always hits the same coverage, the AI might deprioritize it to run less frequently or run it with lower fidelity (maybe in a silicon prototype rather than simulation). Conversely, tests that often fail or cover critical features might be run more often or with more randomness. This adaptive regression strategy ensures computing resources (CPU hours, FPGA time, etc.) are utilized most effectively.

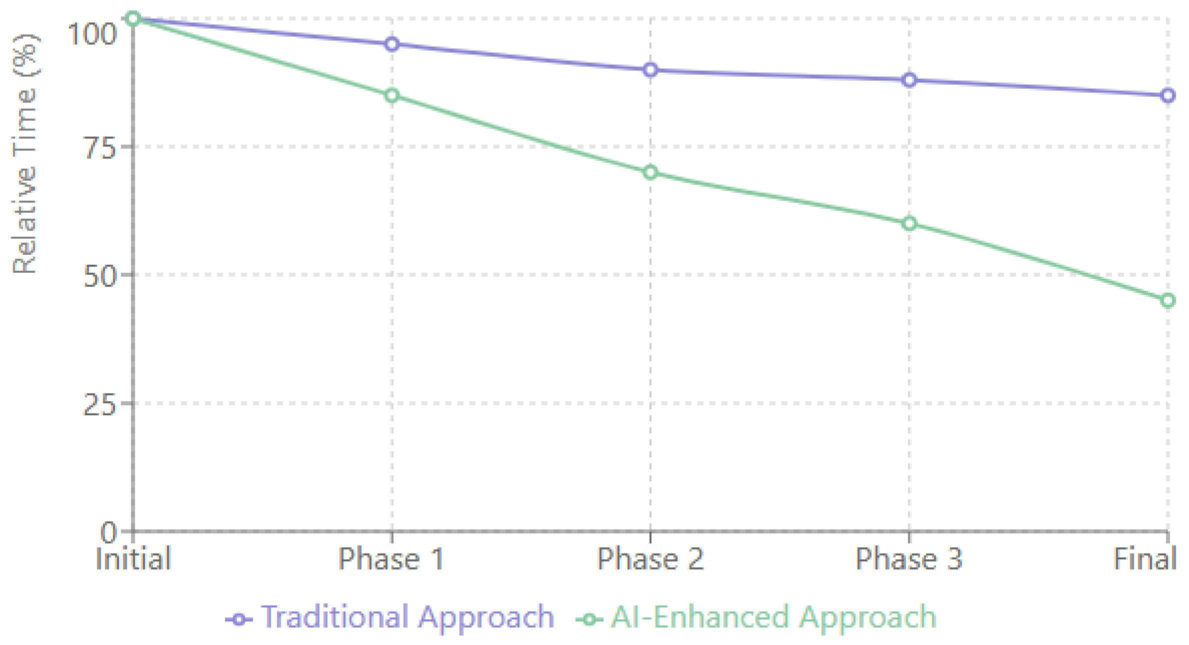

Fig. 3: PCIe/CXL Verification Time Comparison

Figure 3: Verification time comparison between traditional and AI-augmented verification for a complex CXL/PCIe system. The AI-augmented approach (green line) shows a steeper curve, reaching coverage and bug discovery goals faster by leveraging intelligent test selection and debug automation. In contrast, the traditional approach (blue line) takes more simulation cycles to achieve the same goals. This exemplifies the efficiency gains – shorter verification cycles and fewer resources – when AI-driven techniques are integrated into the verification process.

4.3 System Performance Analysis

System-level performance analysis in CXL verification ensures that the entire platform (hosts, switches, memory devices, accelerators) performs as expected when working together. It's not just individual devices that must meet performance metrics but the combination of components under realistic workloads. For example, one might verify that a server with two CPUs and three CXL memory expanders can sustain a certain memory bandwidth while servicing coherency traffic from two GPUs, all without performance degradation or violation of QoS (Quality of Service) policies.

Teledyne LeCroy's research on validating CXL memory device functionality offers detailed insights that can be extrapolated to system-level performance [7]. They emphasize comprehensive testing across different system configurations and operating conditions. In practice, this means varying things like the number of CXL devices connected, the topology of connections (maybe a cascade of switches vs. direct attach), and the mix of traffic (some memory devices may be handling memory pooling for virtualization, while others are dedicated to specific accelerators). The verification environment should simulate full system workloads. For instance, consider an AI training scenario: the CPUs might be loading data from storage via CXL.io, multiple GPUs (Type 2 devices) exchange data via CXL.cache, and Type 3 memory devices provide a large shared memory pool. Under this complex workload, the performance analysis would check metrics like total system throughput, per-component utilization, latency from CPU to GPU memory, etc.

One important aspect is contention and QoS. In a system, multiple agents share resources (PCIe lanes, CXL switch buffers, memory bandwidth). Verification needs to ensure that the system can handle contention gracefully. This could involve scenarios where one device suddenly starts consuming a lot of bandwidth and observing how it impacts others. Does the system fairly arbitrate bandwidth? Does any critical traffic (like coherency messages necessary to maintain correctness) get starved or delayed beyond acceptable limits? These questions are answered by instrumenting the verification platform to measure delays and throughput for each traffic class and comparing them against expected QoS policies.

Another aspect is power and thermal performance under these scenarios, which, albeit outside pure functional verification, can be important for system validation. A heavy CXL workload might cause higher power draw or heat in certain components. At the same time, this is usually tested on real hardware; some aspects can be modeled and checked in pre-silicon verification (e.g., whether throttle mechanisms engage when they should prevent overheating).

Li et al.'s study on intelligent verification systems (though in a different domain) underscores the importance of adaptive verification strategies when assessing system performance [8]. Applied here, it means the verification should not use a one-size-fits-all approach. If a particular system config passes all tests easily, the framework might automatically increase the stress (for example, shorten timers to push the system closer to edge conditions or add more simultaneous initiators to maximize contention) until it finds the breaking point. Conversely, if the system struggles to meet performance in verification, the environment might pinpoint which component is the bottleneck by systematically varying parameters.

Ultimately, system performance analysis in CXL verification ensures that the system meets the design goals for throughput, latency, and reliability when all pieces are integrated. It often requires close collaboration between verification and architecture teams: if verification finds, say, that adding a third accelerator causes a precipitous drop in performance due to coherency storms on CXL.cache, architects might revisit the design (maybe by adding an extra CXL port or re-balancing how memory is distributed). Thus, this verification directly impacts the final system architecture and configuration recommendations.

4.4 Verification System Optimization

The complexity of CXL verification itself means that the verification environment must be optimized. This section subtly differs from using AI to optimize verifying the design; here, we consider improving the performance and capabilities of the verification system (tools and infrastructure) to handle CXL. Techniques such as emulation, FPGA prototyping, and virtualization of components are often employed to speed up verification or make it more realistic.

Implementing intelligent verification systems (as described by Li et al. and others) has shown promising results in improving testing efficiency [8]. For instance, deploying a cluster of FPGA prototypes running CXL transactions can allow many tests to execute parallel at near-real-time speeds. These issues would take too long to uncover in a simulation. AI can also be layered on these – for example, using machine learning on emulator traces to detect anomalies, similar to simulation.

One concrete optimization is the co-emulation of CPU and device models. Rather than simulate an x86 CPU core (which is slow), verification might run a software model of the CPU issuing real PCIe/CXL commands to an emulated device. This hybrid approach can test software-driven scenarios (like a driver allocating memory via CXL) in a fraction of the time of pure simulation. Ensuring the correctness of such co-emulation is part of verification optimization: the environment must be validated to represent the hardware behavior truly. A subset of tests is often run in pure simulation and emulation to confirm that the faster setup checks the design equivalently.

Another area of optimization is in coverage analysis and results processing. Automating the analysis of results is crucial when dealing with millions of tests. Verification teams develop scripts and AI tools that parse logs, extract coverage, and even file bug reports automatically for certain classes of failures. By optimizing these steps, the verification cycle becomes more efficient—engineers spend time addressing issues rather than doing bookkeeping. Li et al. noted that AI-enhanced verification can reduce testing time by identifying issues faster [8]. This can be viewed as optimizing human resources: making the verification system smart enough to do tasks autonomously that would otherwise occupy a human (like root cause analysis for a common type of failure).

In the context of CXL, verification system optimization also includes ensuring that the testbench can scale with the design. The simulation or emulation platform should handle that, as CXL supports more devices and larger fabrics. This might mean optimizing data structures in the simulator to deal with thousands of memory regions or optimizing interconnect models so that adding more devices doesn't linearly slow the simulation. It's a meta-verification challenge: verifying that our verification tools remain effective at scale. Some teams use sharding of tests cleverly — splitting the verification tasks across multiple machines — again coordinated by intelligent scheduling algorithms to maximize throughput.

In essence, Section 4.4 closes the loop by focusing on how to verify better. If earlier sections discuss verifying faster (with AI making it quicker to hit goals) and deeper (covering more scenarios), this part addresses verifying smarter in terms of resource utilization. The outcome is a verification process that keeps up with the pace of CXL technology development. As CXL iterations come rapidly (with 3.0, 3.1, and likely 4.0 in the future), an optimized verification system enables engineers to validate each new iteration within reasonable timeframes and confidently deploy CXL in production systems.

5. Future Directions

5.1 Advanced Memory Disaggregation Analysis

The future of CXL verification is closely tied to the trajectory of memory disaggregation technologies. CXL is a key enabler for memory disaggregation, separating memory from computing and allowing the dynamic composition of memory resources. As this concept becomes more advanced, verification will need to cover even more complex and large-scale scenarios. Wang et al.'s comprehensive simulation framework for CXL disaggregated memory suggests that next-generation verification systems must model memory usage across distributed systems with high fidelity. This means simulating entire racks or data centers worth of CXL-connected nodes, where memory can be pooled and shared on the fly among many hosts.

One aspect is verifying consistency and coherency across multiple memory domains. Future CXL might allow one shared memory pool and multiple, possibly hierarchical pools with various latency tiers. Verification must ensure that when an application's memory is moved from one pool to another (for example, during runtime optimization), the CXL protocols handle it without issues – no data is lost, mappings are updated coherently, and performance adapts. Features like memory migration or replication at the CXL level will introduce new verification requirements, such as checking that copies of data remain identical and synchronized across domains.

Another likely development is more sophisticated Quality of Service and isolation mechanisms in memory disaggregation. Today, CXL provides the plumbing to share memory, but in the future, there may be finer controls so that, for instance, one tenant's workload cannot interfere with another's memory performance in a cloud environment. Verification will then have to validate such isolation: for example, a rogue device cannot monopolize the memory bandwidth or snoop data from another device's memory segment. This will involve security verification merging with performance verification.

Additionally, as memory disaggregation grows, the management software/firmware verification becomes important. CXL relies on system software (like BIOS, OS, and hypervisors) to configure memory regions and access permissions. Future verification should include co-simulation of software or use of formal verification for algorithms that allocate and migrate memory. Ensuring that the software algorithms for disaggregation do not lead to corner-case failures (like double-allocating the same memory to two hosts or failing to revoke access in time) will be crucial for system reliability.

In summary, verifying advanced memory disaggregation will push verification tools to simulate larger systems, incorporate more software/hardware co-verification, and cover new features around memory management. This is a natural extension as CXL moves from board-level connectivity to rack-scale composable architectures.

5.2 System-Level Integration Verification

As CXL technology matures, real-world deployments will involve a mix of many component types: CPUs, GPUs, FPGAs, memory devices, switches, perhaps optical extenders, etc., all linked by CXL. System-level integration verification will become a top priority – ensuring that a heterogeneous set of components can all work together under the CXL standard. Xi Wang's research on CXL system adoption highlights the importance of comprehensive methodologies that validate interactions across diverse computing environments [10].

One emerging need is verifying interoperability beyond the spec. While compliance testing (as in Section 2.2) checks that each device meets the spec, system-level testing will check that devices from different vendors interoperate smoothly. For instance, one vendor's CPU with CXL 3.0 might be connected to another vendor's CXL switch and a third vendor's memory device and accelerator. The verification question is: does the entire system initialize correctly and maintain operation under load without protocol deadlocks or performance pathology? This might involve large-scale plug-and-play testing, effectively building a library of device models (or real devices), and trying various combinations in simulation or emulation.

Another future challenge is hot plug and dynamic reconfiguration at scale. CXL 3.0 introduces the notion that devices can be added or removed (akin to PCIe hot plug but possibly more frequently used, especially for memory pooling scenarios where resources might be provisioned on demand). Verification must ensure that adding or removing devices or changing the fabric topology does not cause issues like data loss or system hangs. This can be complex: imagine a memory device is removed while two hosts are actively using it – the system should have protocols to handle this (perhaps migrating data off or quiescing access) and all that needs verification.

Xi Wang et al. also imply the need to model real-world usage in verification [10]. This could mean running full software stacks in a virtualized environment on a simulated/emulated CXL platform. The verification might involve booting an OS, running workload traces (like database operations machine learning training loops) on a model of a future CXL-based server, and observing both correctness and performance. Essentially, the line between "verification" and "validation" (in the sense of system validation or user scenario testing) will blur. Pre-silicon and post-silicon verification teams will likely collaborate, using common scenarios – with pre-silicon trying to catch issues before hardware is built and post-silicon confirming and tuning with real hardware.

System-level integration will also require more formal approaches for certain aspects because testing all combinations may be infeasible. For safety-critical uses of CXL (like in defense or automotive high-performance computing), one might need formal proof that, for example, a certain failure in one component (like a firmware crash in a Type 3 device) will isolate and not cascade through the CXL network. Formal verification of network-wide properties or fail-safe states could become part of the methodology.

5.3 Protocol Analysis Advancements

Future developments in protocol analysis will be driven by the increasing complexity of CXL features and the need for even more robust verification tools. As CXL evolves (CXL 4.0 and beyond), new protocol messages, states, and error-handling procedures will be added. Verification tools (like protocol checkers, formal protocol verifiers, etc.) must advance to handle these. Wang et al.'s simulation framework research emphasizes the need for more sophisticated methodologies to handle complex protocol interactions across disaggregated memory systems [9]. This can be extrapolated to mean that protocol analyzers might incorporate AI or formal methods to explain the causality and temporal ordering in a complex CXL fabric.

One advancement could be in formal protocol verification, using model checking or theorem proving to verify certain invariant properties of the CXL protocol (for instance, that cache coherency will always terminate or that credit loops cannot deadlock the link). While formal methods are already used for smaller protocols, applying them to something as broad as CXL is challenging but potentially feasible in parts (perhaps verifying the cache coherency protocol state machines in isolation, for example).

Another area is improved visualization and analysis tools. Future verification might leverage AR/VR or advanced GUIs to visualize the CXL fabric's operations over time, making it easier for engineers to grasp complex interactions intuitively. Imagine a 3D visualization where each device is a node and memory operations are arrows flying between them – patterns that cause problems might show as congestion or misrouted arrows, guiding further analysis. While this concerns engineer productivity, it ties into protocol analysis by presenting data in digestible formats.

Protocols for telemetry and debug hooks built into CXL devices, which verification will use. For example, a future CXL spec might allow a device to report statistics or internal states (for performance tuning or failure analysis). Verification would then ensure those telemetry reports are accurate and useful. This is somewhat meta: verifying the debug features of the protocol. However, as systems scale, having self-monitoring features becomes important; those features also require validation.

Furthermore, as memory semantics become richer (perhaps future CXL versions could support consistency models beyond strict coherence or operations like memory-to-memory copies), protocol analysis techniques must cover memory ordering and consistency verification. This might involve borrowing approaches from verifying CPU memory models (an area of active research) and applying them to a distributed CXL context.

In sum, protocol analysis advancements will provide the tooling and methodologies to keep verification rigorous even as CXL's protocol expands. They will help maintain confidence that despite CXL enabling new degrees of freedom in system design, the fundamental guarantees (coherence, integrity, ordering, etc.) hold under all circumstances.

5.4 System Integration and Validation

Looking further ahead, integrating CXL into virtually all aspects of system design means verification will encompass not just the CXL links in isolation but their role in the entire platform operation. The boundary between verifying "the CXL part" and verifying "the whole system that uses CXL" will disappear. According to Xi Wang's evaluations of real-world CXL implementations, verification systems must evolve to handle increasingly complex integration scenarios [10].

One trend is that CXL could be used with other interconnects (like Ethernet or InfiniBand in clusters or custom interconnects on chip). Verifying CXL in a multi-fabric environment is a future concern. For example, a system might use CXL within a server and Ethernet across servers to share memory globally. Consistency and correctness might depend on both protocols. Verification might need to create hybrid models (part CXL, part network) to ensure end-to-end data correctness and efficiency.

Another likely future scenario is CXL in edge computing and IoT, where the scale is smaller, but there may be real-time constraints. Ensuring that CXL's added latency doesn't break real-time assumptions or verifying that devices can enter low-power states appropriately when idle (important for battery-powered or energy-sensitive deployments) will be new angles for verification. This goes beyond current data center-oriented verification goals and may involve cross-discipline verification (combining aspects of timing analysis, power verification, and functional verification).

System validation will increasingly incorporate field data and continuous verification. Once CXL systems are deployed, telemetry from the field (as mentioned in protocol analysis advancements) could be fed back into the verification loop. This blurs with the concept of monitoring in production and using that to improve pre-production verification. Future verification setups might routinely integrate new traces or usage patterns observed in real deployments to ensure the next generation or update of the system handles them. Essentially, verification might never truly "end" even after deployment – it becomes a continuous process, a concept sometimes referred to as shifting right (complementing the classic shift left, which means verifying early and verifying continuously after release).

In terms of methodologies, expect more digital twins for CXL systems—high-fidelity models of deployed systems used to recreate issues observed and validate fixes. Verification teams will need to maintain these twins and run them parallel to real systems.

All these future directions indicate that CXL verification will remain dynamic and challenging. It will require keeping pace with rapid technological advances in interconnects and leveraging the latest in verification science (formal methods, AI, large-scale simulation, etc.) to ensure that as CXL enables new computing paradigms, those paradigms are built on a foundation of correctness and reliability.

Conclusion

The article on CXL verification methodologies reveals the crucial importance of adaptive and intelligent approaches in addressing the growing complexities of modern computing systems. Through integrating artificial intelligence and machine learning techniques, verification processes have evolved to effectively handle the challenges of cache coherency, protocol compliance, and system-level integration. The article demonstrates that comprehensive verification strategies, particularly those incorporating AI-driven automation and predictive analytics, are essential for ensuring reliable operation across diverse computing environments. As CXL technology advances, developing sophisticated verification methodologies will remain paramount in supporting the evolution of heterogeneous computing architectures and memory disaggregation technologies, ultimately enabling more efficient and reliable system validation processes.

References

[1]. Debendra Das Sharma et al., "An Introduction to the Compute Express Link (CXL) Interconnect," ResearchGate, June 2023. [Online]. Available: https://www.researchgate.net/publication/371729277_An_Introduction_to_the_Compute_Express_Link_CXL_Interconnect

[2]. CXL Consortium, "Opportunities and Challenges for Compute Express Link (CXL)." [Online]. Available: https://computeexpresslink.org/wp-content/uploads/2024/11/CR-CXL-101_FINAL.pdf

[3]. Nikhil Jain, Zongyao Wen, "Verifying CXL 3.1 Designs with Synopsys Verification IP," Synopsys, 2024. [Online]. Available: https://www.synopsys.com/blogs/chip-design/verifying-cxl3-1-designs-with-synopsys-verification-ip.html

[4]. Narasimha Babu GVL, "Integrity and Data Encryption (IDE) Trends and Verification Challenges in CXL® (Compute Express Link®)," CXL Consortium Technical Blog, April 23, 2024. [Online]. Available: https://computeexpresslink.org/blog/integrity-and-data-encryption-ide-trends-and-verification-challenges-in-cxl-compute-express-link-2797/

[5]. Sathish Kumar M et al., "Case study of Optimized CXL platforms for AI/ML Check Points," Future Memory Storage Proceedings, 2024. [Online]. Available: https://files.futurememorystorage.com/proceedings/2024/20240808_AIML-303-1_Pillai_Kumar.pdf

[6]. Brian Bailey, "AI-Powered Verification," Semiconductor Engineering, June 1st, 2022. [Online]. Available: https://semiengineering.com/ai-powered-verification/

[7]. Teledyne LeCroy, "Validating CXL Memory Device Functionality and Performance with Teledyne Lecroy," White Paper, 2024. [Online]. Available: https://cdn.teledynelecroy.com/files/whitepapers/cxl-memory-device-wp.pdf

[8]. Hongwei Li et al., "A Review of Intelligent Verification System for Distribution Automation Terminal based on Artificial Intelligence Algorithms," Journal of Cloud Computing volume 12, Article number: 146 (2023), 16 October 2023. [Online]. Available: https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-023-00527-2

[9]. Yerra, S. (2025). Optimizing supply chain efficiency using AI-driven predictive analytics in logistics. doi : https://doi.org/10.32628/CSEIT25112475

[10]. Yanjing Wang et al., "A Comprehensive Simulation Framework for CXL Disaggregated Memory," arXiv:2411.02282 [cs.ET], 9 Mar 2025. [Online]. Available: https://arxiv.org/abs/2411.02282

[11]. Xi Wang et al., "Exploring and Evaluating Real-world CXL: Use Cases and System Adoption," arXiv:2405.14209 [cs.PF], 16 Feb 2025. [Online]. Available: https://arxiv.org/abs/2405.14209