A Systematic Survey of CXL Type-3 Devices: Evolution, Architectures, and Future Directions for Memory-Intensive Computing

Dawen Liang *College of Engineering and Computer Science*
Syracuse University
Syracuse, NY, USA
dliang02@hotmail.com

***Abstract*—The rapid growth of memory-intensive workloads, such as trillion-parameter language models and in-memory databases that process petabytes of data, has revealed fundamental limitations in traditional memory architectures. Compute Express Link (CXL), a cache-coherent interconnect built on PCIe, introduces Type-3 memory expanders. These expanders create a new layer between local DRAM and storage, providing byte-addressable, load/store-accessible capacity with a latency of 200 to 400 nanoseconds. This survey examines CXL Type-3 devices in three areas: (1) hardware architecture and performance, (2) systems software including kernel-level tiering, allocation, and virtualization support, and (3) emerging applications in in-memory databases, high-performance computing, and AI/LLM workloads. We review over 40 papers from respected venues (OSDI, ASPLOS, MICRO, SIGMOD, SOSP) published from 2022 to 2025 and highlight key challenges such as managing tail latency, ensuring security in shared memory pools, and exploring hardware-software co-design opportunities. By connecting the systems and machine learning communities, we assert that CXL Type-3 devices are crucial for the next generation of memory-intensive computing systems.**

***Index Terms*—CXL, Compute Express Link, Type-3 devices, memory disaggregation, memory-intensive computing, memory tiering**

I. Introduction

Modern datacenter workloads often face memory bottlenecks rather than computation delays. Barroso et al. identified a critical latency range, called "killer microseconds," where operations taking 1 to 100 microseconds—including remote memory accesses and inter-socket communication—become the primary performance limitation in large systems [1]. This challenge has escalated significantly: large language models now need hundreds of gigabytes to terabytes of parameter storage, in-memory databases require increasingly larger buffer pools, and graph analytics workflows frequently surpass the memory capacity of a single server.

Compute Express Link (CXL) has emerged as a transformative technology to tackle this memory capacity issue. Built on the PCIe physical layer, CXL enables cache-coherent, byte-addressable access to external memory devices through its CXL.mem protocol [2], [3]. Unlike previous disaggregation methods using RDMA or network-attached storage, CXL Type-3 memory expanders allow CPUs to access remote DRAM using native load/store instructions with latencies of 200 to 400 nanoseconds—about 2 to 3 times that of local DRAM, compared to the 10 to 100 times penalty of network-based solutions. The concept of disaggregated memory was first systematically examined by Lim et al., who showed that separating memory from compute blades could achieve 50% better performance per dollar for datacenter workloads [4].

The systems and application communities have quickly converged on CXL as the enabling technology for memory-intensive computing. Database systems now use CXL for unified buffer pools across multiple compute nodes [5], [6]; operating systems have added CXL-aware tiering into the Linux kernel [7]; and AI researchers are designing memory-augmented architectures that naturally align with CXL's features [8]. A recent competing survey by Chen et al. [9] looks at CXL from a broad protocol viewpoint; our work is distinct as it specifically focuses on Type-3 devices and their applications in memory-intensive computing, especially at the emerging intersection of CXL and AI.

This survey addresses four research questions: **RQ1:** What are the performance characteristics of real CXL Type-3 hardware? **RQ2:** How has systems software adapted to leverage CXL memory tiering? **RQ3:** Which application areas benefit the most from CXL Type-3 devices? **RQ4:** What are the key challenges for CXL adoption?

The rest of this paper is organized as follows. Section II provides background on CXL protocols and the evolution of memory disaggregation. Section III explores CXL hardware architecture and performance. Section IV reviews systems software support. Section V analyzes applications in databases, high-performance computing, and AI workloads. Section VI discusses open challenges, and Section VII concludes.

II. Background

A. CXL Protocol Overview

CXL defines three sub-protocols operating over a shared PCIe physical link: *CXL.io* provides PCIe-compatible I/O semantics for device discovery and configuration; *CXL.cache* enables devices to cache host memory with hardware-managed coherence; and *CXL.mem* allows hosts to access device-attached memory using load/store instructions with full cache coherence [2]. Devices fall into three types based on protocol usage: Type-1 devices (accelerators using CXL.io and CXL.cache), Type-2 devices (accelerators with their own

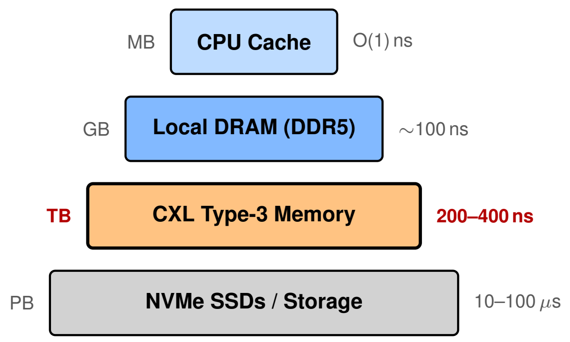

Fig. 1. The expanded memory hierarchy with CXL Type-3 devices. CXL introduces a critical intermediate tier that bridges the "killer microsecond" capacity-latency gap between direct-attached DRAM and block storage. Latency and capacity values are representative ranges from real hardware measurements [10], [11].

memory using all three protocols), and Type-3 devices (memory expanders using CXL.io and CXL.mem). This survey focuses only on Type-3 devices, which are fundamental for CXL-based memory disaggregation.

CXL's specification has rapidly evolved. CXL 1.0 and 1.1 established single-device attachment over PCIe 5.0. CXL 2.0 introduced memory pooling using CXL switches, allowing multiple hosts to share a common memory pool. CXL 3.0 further enhanced the fabric with multi-level switching and back-invalidation snooping for improved coherence scalability [2]. The latest CXL 4.0 specification doubles the data rate to 128 GT/s on a PCIe 7.0 physical layer, achieving up to 768 GB/s unidirectional bandwidth per x16 link, and introduces native support for multi-rack topologies along with enhanced memory RAS features [3].

B. Type-3 Device Positioning in the Memory Hierarchy

CXL Type-3 devices create a new tier in the memory hierarchy, as illustrated in Fig. 1. Sun et al. measured genuine CXL memory systems and found that CXL DRAM access latencies range from 1.35 to 3 times higher than local DDR, depending on how mature the implementation is [10]. At the system level, CXL memory acts as a CPU-less NUMA node: it is byte-addressable and can be accessed using standard load/store instructions, though it has different latency and bandwidth characteristics than local DRAM.

This placement—between local DRAM (about 100 ns) and NVMe storage (about 10 to 100 μs)—makes CXL Type-3 memory especially suitable for workloads needing large capacity but less sensitivity to latency. The host processor's hardware prefetcher can utilize CXL memory, and existing NUMA-aware allocation policies can be applied for page placement, allowing applications to migrate without requiring code changes.

C. Pre-CXL Disaggregation: The Evolutionary Path

The idea of memory disaggregation has deep roots that predate CXL. Lim et al. first proposed this concept at ISCA 2009 [12], imagining "memory blades" connected through a high-speed interconnect. They later developed a software prototype on the Xen hypervisor, showing that disaggregated memory with microsecond-level latency (4 μs over PCIe 2.0) could achieve performance equivalent to all-local memory setups while offering 50% better performance per dollar [4]. This research established crucial design principles, including the need for simple page replacement policies at microsecond latencies and the beneficial interaction between disaggregated memory and content-based page sharing, both of which remain relevant for CXL system design today.

Klimovic et al. extended disaggregation to flash storage, demonstrating that remote flash accessed via 10 GbE with iSCSI could reduce application-level throughput by only 20%, with a 260 μs latency acceptable for workloads with millisecond-scale SLAs [13]. This work highlighted the key tension between the benefits of disaggregation (like improved utilization and independent scaling) and the network overhead that later drove the need for the CXL interconnect.

Gao et al. took disaggregated memory further with Clio, a hardware-software co-designed system featuring custom CBoards with ASIC-accelerated pathways and ARM-based control systems [14]. Clio achieved bounded lookup latency by using overflow-free hash-based page tables and showed that disaggregated memory nodes benefit from separating fast-path data operations (handled by hardware) from slow-path metadata management (managed by software). CXL Type-3 devices incorporate many of these design lessons while removing the need for custom network protocols by utilizing the PCIe physical layer.

D. Workload-Driven Demand for CXL

Three application areas have driven the demand for CXL Type-3 memory. In-memory databases like SAP HANA have shown that CXL memory expansion allows processing of datasets larger than available DRAM with acceptable performance loss [15]. Billion-scale nearest neighbor search services, such as CXL-ANNS, require over 40 TB of index memory across multiple devices, making CXL pooling crucial for cost-effective deployment [16]. Recently, memory-augmented LLM architectures like Engram have introduced enormous parameter tables (with over 100 billion entries) featuring deterministic, prefetchable access patterns that naturally fit CXL's byte-addressable memory design [8]. These converging demands from the database, systems, and AI communities encourage a unified examination of CXL Type-3 devices.

III. CXL Hardware Architecture and Performance

A. HDM Models, MLD, and Multi-Headed Devices

The CXL specification defines Host-managed Device Memory (HDM) as the main abstraction for Type-3 device memory. HDM supports three decoder models: HDM-H (host-only decoding), HDM-D (device-managed decoding), and HDM-DB (device-managed with back-invalidation) [2]. Starting with CXL 2.0, Multi-Logical Devices (MLDs) allow a single physical device to present up to 16 logical devices, each assignable to different hosts. This feature, along with CXL switches that support fabric-level pooling, supports rack-scale memory sharing. Multiple hosts can allocate and release memory from a shared pool. Fig. 2 illustrates how CXL topologies have changed from direct attachment to multi-rack fabrics.

Pond, the first CXL-based memory pooling system designed for production cloud environments, showed that small-scale pooling across 8–16 sockets can recover up to 25% of stranded DRAM while introducing only 70–90 ns of added latency [17]. Pond introduced a Zero-CPU NUMA Node (zNUMA) abstraction that presents pooled CXL memory as a NUMA node without linked CPU cores. This allows the OS to use its existing NUMA allocation policies. DRack expanded pooling to the rack scale, combining CXL memory pooling with NIC pooling to tackle inter-rack communication issues [18]. DRack's design demonstrated that the blend of memory-semantic access (CXL.mem) and device pooling (CXL switches) could resolve both memory stranding and network underutilization issues.

B. Performance Characteristics

1) Quantitative Benchmarking with Real Hardware: Sun et al. conducted the first thorough evaluation of real CXL memory systems using Intel Sapphire Rapids CPUs with three commercial CXL devices [10]. A key finding was that real CXL memory shows fundamentally different performance than emulated CXL (remote NUMA node). Genuine CXL devices exhibited up to 26% lower latency and 3–66% higher bandwidth efficiency compared to NUMA emulation. This is because CXL memory lacks CPU cores and caches, which allows the CPU to perform faster coherence checks. Weisgut et al. extended this evaluation to database workloads, measuring sequential read throughput at 40 GB/s, random read latency at 520 ns, and TPC-H performance at 71–73% of local DRAM [11].

2) Tail Latency Analysis and Stability: Liu et al. conducted the most extensive CXL characterization to date, testing 265 workloads on real hardware [19]. They found that tail latency on CXL memory is a significant issue. Their analysis showed consistent tail latency inflation under load. They proposed the Spa (Stall-based CXL performance analysis) framework for workload-aware CXL placement. A crucial finding was that CXL+NUMA setups could lead to unexpectedly high slowdowns (up to $2.9\times$ for tail-latency-sensitive workloads). However, interleaving multiple devices across two CXL expanders can reach an aggregate bandwidth of approximately 104 GB/s, close to local DRAM levels.

3) NUMA Integration and Interleaving: Several studies confirm that CXL memory is best understood as a high-latency NUMA node rather than a separate device class. DirectCXL showed that zero-copy, load/store access to CXL memory via FPGA prototypes achieves three times lower latency than RDMA-based disaggregation [20]. The Sub-NUMA Clustering (SNC) effect identified by Sun et al. demonstrated that CPU cores using CXL memory can benefit from 2–4 times larger effective LLC capacity, partially offsetting the higher memory latency [10].

C. Device Pooling Beyond Memory

The pooling concept extends beyond memory devices. Zhong et al. introduced Oasis, which pools network interfaces (NICs) and NVMe SSDs over CXL fabric, achieving twice the device utilization and 38 ms failover times [21]. This work illustrates that CXL's device pooling abilities create a flexible disaggregation fabric useful for various I/O devices, not just memory expanders.

IV. Systems Software for CXL Memory

A. Linux Kernel CXL Subsystem

Linux kernel support for CXL has advanced quickly since its introduction in version 5.12. The `drivers/cxl/` subsystem provides abstractions for CXL ports, memory regions, and endpoints. DAX (Direct Access) integration allows user-space applications to map CXL devices directly into memory. Key developments include CXL memory hotplug support (5.19), NUMA-aware QoS policies (6.8), and weighted interleave across different memory types (6.9). These kernel features form the basis for higher-level tiering and allocation strategies, as shown in the systems software chart in Fig. 3.

B. Evolution of Memory Tiering and Placement

1) TPP: First CXL-Aware Kernel Tiering: Transparent Page Placement (TPP) by Maruf et al. was the first CXL-aware memory management feature added to the Linux kernel (version 5.18) [7]. When analyzing Meta production workloads, the authors discovered that 55–80% of allocated memory is cold (not accessed for over 2 minutes) and that anonymous pages are much hotter than file-backed pages. TPP introduces four main mechanisms: (1) lightweight demotion that moves rather than swaps cold pages to CXL memory, (2) separate allocation and demotion thresholds to keep room for hot allocations, (3) page-fault-driven promotion that moves hot CXL pages back to local DRAM, and (4) first-touch allocation bias that directs new anonymous pages to local DRAM, since they are typically short-lived and frequently accessed.

2) TMO: Production-Scale Memory Offloading: Meta's Transparent Memory Offloading (TMO) takes a different approach by adding Pressure Stall Information (PSI)—a kernel feature that measures lost work due to resource shortages—as feedback for memory tiering choices [22]. Unlike TPP's page-access tracking, TMO's Senpai agent applies light memory pressure and uses PSI data to decide how much memory to offload without needing prior application knowledge. TMO works with various backends like zswap compressed memory and NVMe SSDs, with CXL as a suitable future tier. Deployed across Meta's entire fleet, TMO leads to significant memory savings while keeping application-level service level objectives (SLOs).

3) Colloid: Loaded-Latency Aware Tiering: Colloid identifies a major issue in current tiering systems: they assume the default memory tier has lower access latency than other tiers, which is not true under realistic conditions [23]. The authors show that the loaded latency of the default tier can increase by up to five times, making it 2.5 times slower than CXL memory during heavy access. Colloid introduces the idea of *balancing loaded latencies* across tiers using hardware CHA counters and Little's Law to assess each tier's queue occupancy with microsecond precision. When integrated with

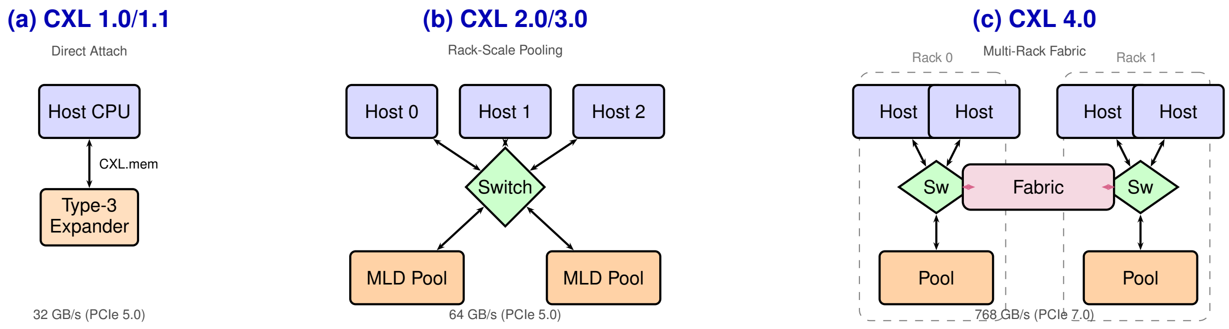

Fig. 2. Architectural evolution of CXL topologies. (a) CXL 1.0/1.1 provides direct-attach memory expansion for a single host. (b) CXL 2.0/3.0 introduces CXL switches and Multi-Logical Devices (MLDs) to enable rack-scale memory pooling across multiple hosts. (c) CXL 4.0 adds native multi-rack fabric support with PCIe 7.0, enabling datacenter-scale memory disaggregation.

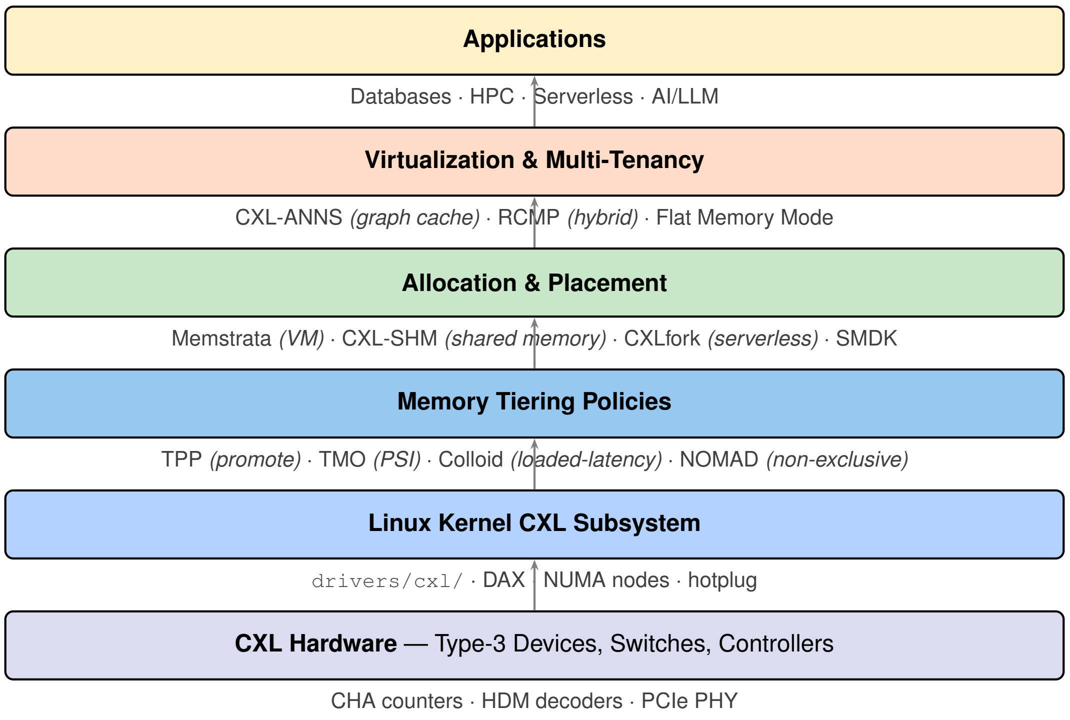
Applications
Databases · HPC · Serverless · AI/LLM
Virtualization & Multi-Tenancy
CXL-ANNS (graph cache) · RCMP (hybrid) · Flat Memory Mode
Allocation & Placement
Memstrata (VM) · CXL-SHM (shared memory) · CXLfork (serverless) · SMDK
Memory Tiering Policies
TPP (promote) · TMO (PSI) · Colloid (loaded-latency) · NOMAD (non-exclusive)
Linux Kernel CXL Subsystem
drivers/cxl/ · DAX · NUMA nodes · hotplug
CXL Hardware — Type-3 Devices, Switches, Controllers
CHA counters · HDM decoders · PCIe PHY

Fig. 3. Taxonomy of systems software for CXL tiered memory management. Each layer builds upon the one below, from hardware telemetry through kernel abstractions to application-level policies. Representative systems are mapped to their primary layer. Works in Section IV span the middle four layers.

existing tiering systems like TPP, Colloid achieves a 1.2 to 2.3 times improvement over unchanged baselines.

4) Nomad: Non-Exclusive Tiering: NOMAD challenges the standard exclusive placement model where pages exist in either fast or slow memory [24]. Under memory pressure, exclusive tiering can cause excessive swapping of hot pages between tiers. NOMAD keeps shadow copies of promoted pages in CXL memory, allowing efficient demotion by simply changing page table entries for clean pages. Its Transactional Page Migration (TPM) feature copies page content without unmapping the original page, verifying cleanliness upon completion. This means migration does not interfere with the application's critical path and decreases perceived bandwidth loss from as much as 95% to virtually zero.

C. CXL-Aware Allocation Strategies

1) Fine-Grained Placement with Memstrata: Zhong et al. address CXL memory management in cloud virtualization settings with Memstrata [25]. The system uses Intel's Flat Memory Mode—a hardware-managed tiering method working at cache-line detail within the memory controller—combined with page coloring to minimize inter-VM contention. A simple online random forest classifier estimates per-VM slowdown from CXL placement, enabling real-time allocation of dedicated local DRAM to performance-sensitive tenants. Across 115 production workloads from Microsoft Azure, Memstrata limits the average VM slowdown to under 5%.

2) Handling Partial Failures: Zhang et al. handle automatic memory management in CXL shared memory during partial failures [26]. Their CXL-SHM system implements era-based non-blocking reference counting, allowing recovery from client crashes without blocking other participants. CXL-SHM achieves tens of millions of allocations per second (43.2 million operations per second) with an asynchronous recovery rate of 23.5 million objects per second.

3) Dynamic Forking with CXLfork: CXLfork showcases a new use for CXL shared memory in serverless computing, enabling zero-copy remote forking across different physical hosts linked through a CXL switch [27]. By keeping process memory in CXL-attached DRAM, CXLfork eliminates the need to transfer memory state during fork operations. This leads to 2.26× faster cold starts compared to CRIU-based checkpointing, with an 87% reduction in memory usage. Under memory pressure, CXLfork's companion system CXL-porter cuts P99 latency by up to 16×.

D. Virtualization and Multi-Tenancy

CXL memory pooling presents distinct virtualization challenges. CXL-ANNS showed that billion-scale vector searches over a shared memory pool need relationship-aware graph caching, keeping nodes within 2–3 hops of the entry point in local DRAM while the rest remains in CXL memory, servicing 59.4% of traversals locally [16]. Samsung's SMDK offers a user-space toolkit for explicit CXL memory allocation, including mmap extensions and NUMA-aware placement APIs [**?**]. RCMP bridges the transition from RDMA to CXL by combining CXL's low-latency intra-rack access with RDMA's cross-rack scalability in a hybrid setup [28].

V. Applications of CXL Type-3 Devices

A. In-Memory Databases and Transaction Processing

1) Tigon: CXL Pod-Based Distributed Databases: Tigon introduces the idea of a "CXL Pod," which is a closely connected group of 8 to 16 hosts sharing CXL memory through a multi-headed device (MHD). This serves as a new scaling option for distributed databases [6]. Within a Pod, Tigon replaces the traditional two-phase commit (2PC) with atomic operations on shared CXL memory. It keeps the Cross-host Active Tuples (CAT, ∼7 MB)—which are the set of tuples accessed concurrently by transactions from different hosts—in CXL memory to avoid network round-trips. The system supports both hardware coherence (HWcc) and software coherence (SWcc) modes. In TPC-C benchmarks, Tigon achieves 2.5 times the throughput compared to shared-nothing baselines and 15.9 to 18.5 times the throughput compared to RDMA-based systems.

2) CtXnL: Hybrid Coherence for Transactional Workloads: Wang et al. propose CtXnL, which uses different coherence policies based on data semantics: strict hardware coherence for metadata (locks, indexes) and relaxed software coherence for data records [29]. This hybrid method achieves 2.08 times the throughput of standard CXL coherence while reducing coherence traffic by 95%.

3) PolarCXLMem: Production-Scale Unified Buffer Pool: PolarCXLMem is the first production deployment of CXL 2.0 switches for cloud-native databases at Alibaba [5]. The system stores the entire database buffer pool (data pages and metadata) in CXL memory, eliminating local buffer management and reducing memory overhead by 30 to 50%. In pooling scenarios, PolarCXLMem achieves 3.6 M QPS compared to 1.1 M QPS for RDMA baselines (3.27× improvement). Crash recovery is 4.1 to 4.9 times faster because CXL memory's independent power domain preserves buffer state during compute node failures.

B. HPC, Graph, and Serverless Computing

1) PMem Migration Pathways in HPC: With Intel Optane persistent memory discontinued, Fridman et al. assess CXL memory as a replacement for PMem-based HPC workloads [30]. STREAM benchmark results show that CXL memory offers comparable bandwidth to Optane PMem while providing better latency predictability. This positions CXL as a natural successor for memory capacity-sensitive HPC applications.

2) CXLfork for Serverless Cold Start Acceleration: In addition to its systems software contribution (Section IV-C), CXLfork shows a strong application-level benefit for serverless computing [27]. Function-as-a-Service platforms face multi-second cold start latencies mostly due to memory initialization. CXLfork's zero-copy remote fork cuts cold start times by 2.26× compared to CRIU while reducing the memory used per function by 87% through memory sharing across CXL-connected hosts.

C. AI and LLM-Specific CXL Architectures

1) SmartQuant: Bit-Plane Storage for Dynamic Quantization: SmartQuant uses CXL Type-3 memory as a model store with runtime quantization capability [31]. It organizes model weights in bit-plane format across CXL memory and employs logical space bloating to fit quantized access patterns. This allows for dynamic precision selection at inference time without needing to reload the model.

2) CXL-PNM and CLAY: Near-Memory Processing for AI: CLAY introduces a CXL-based Near-Data Processing architecture for accelerating embedding layers in recommender systems and graph neural networks [32]. Unlike DIMM-based NDP approaches that are limited by memory channel constraints, CLAY uses CXL's packetized protocol to connect DRAM clusters through board-level interconnects, each containing vector processing units alongside memory. This setup allows for detailed load balancing across embedding dimensions while complying with the CXL protocol. CXL-PNM extends this idea to LLM attention and activation computation, offloading memory-heavy tasks to processing elements within CXL expanders [33].

3) Engram as a Co-Design Case Study: Engram offers compelling evidence that CXL and memory-augmented AI architectures work well together [8]. Engram implements "conditional memory," which adds a sparsity aspect that complements MoE's conditional computation. It retrieves static knowledge patterns through hash-based lookups into large embedding tables (100B+ parameters). Three aspects make this workload ideal for CXL, as shown in Fig. 4:

TABLE I
COMPARISON OF INTERCONNECT TECHNOLOGIES FOR SPARSE PARAMETER ACCESS

Dimension	**CXL.mem**	**RDMA**	**NVMe**
Latency	200–400 ns	2–5 μs	10–100 μs
Addressing	Byte, load/store	Page, verbs	Block, I/O
Coherence	HW cache coherent	None	None
Prefetch	CPU prefetcher	No	No

Deterministic addressing: Unlike attention-based memory access, Engram's N-gram hash lookups generate known addresses before the data is needed, allowing the CPU hardware prefetcher to overlap CXL latency with computation.

Zipfian locality: Engram's access pattern follows a power-law distribution where about 10 to 20% of entries account for over 80% of accesses. This active set fits in local DRAM or CXL DRAM, while the longer tail is in lower-cost storage.

Coherence-friendly reads: The embedding tables are mostly read during inference. This reduces cache invalidation overhead and makes CXL's hardware coherence protocol essentially free compared to RDMA's software-managed consistency.

Engram shows that offloading a 100B-parameter table to host memory results in less than 3% loss in inference throughput [8]. Since CXL offers a faster and larger memory tier than NVMe, this overhead would decrease further while allowing for cross-node parameter sharing via CXL pooling. Engram's U-shaped scaling law indicates that mixed allocation of around 75 to 80% MoE computation and 20 to 25% Engram memory performs better than pure MoE. This suggests that expanding memory capacity with CXL could fundamentally change the compute-memory trade-off in LLM scaling.

VI. OPEN CHALLENGES AND FUTURE DIRECTIONS

A. Protocol Bottlenecks and Tail Latency

Although CXL reduces the gap between local and remote memory, tail latency remains a major issue. Liu et al. found that CXL tail latency increases significantly under contention, with bandwidth-bound workloads experiencing slowdowns of 1.5 to 5.8 times, while CXL+NUMA configurations can lead to up to 2.9× degradation, even with low bandwidth demands [19]. The 256-byte flit overhead and protocol processing add fixed latency that becomes increasingly significant for small random accesses. CXL 4.0's bandwidth increase to 128 GT/s should help with throughput bottlenecks [3], but managing tail latency needs workload-aware placement policies and possibly hardware-level QoS mechanisms that are not yet in the specification.

B. Security and Isolation

Shared CXL memory pools create new vulnerabilities. Two recent studies tackle this challenge from different perspectives.

1) CXL-Address-Based Security: Abdullah et al. propose Salus, a security model for systems with dynamic page migration between GPU memory and CXL-expanded pools [34]. Salus separates security metadata from physical data locations, which minimizes costly security recalculations during page migration. The evaluation shows a geometric mean improvement of 29.94% in GPU throughput compared to traditional security models, with security-related traffic reduced to 47.79% of baseline levels.

2) Scalable Freshness Guarantees with Toleo: Dong et al. introduce Toleo, which uses a small CXL-attached smart memory device (168 GB) with Processing-in-Memory (PIM) capability to safeguard a larger 28 TB CXL pool [35]. Toleo achieves a 240× metadata compression ratio through trip-based version tracking, keeping freshness guarantees with only 2% performance overhead. The system uses CXL IDE (Integrity and Data Encryption) for in-transit protection.

C. AI-Systems Co-Design: The Engram Case

Combining memory-augmented AI designs with CXL infrastructure offers a unique co-design chance. Engram's conditional memory concept [8] shows that AI model designers can shape memory access patterns to fit CXL features. Future co-design could cover three areas: *hardware* (CXL controllers with application-aware prefetch logic), *operating systems* (workload-specific tiering policies for AI memory access patterns), and *model architecture* (designing memory layers that consider CXL latency and bandwidth limits).

D. Future Directions

Several emerging trends will influence CXL Type-3 adoption. Near-memory computing, as shown by CLAY's NDP approach [32], shifts computation to CXL expanders to lower data movement. The merger of CXL 4.0 with UCIe (Universal Chiplet Interconnect Express) promises closer integration of memory and compute chiplets. Multi-rack CXL fabrics, now supported in CXL 4.0 [3], will enable memory pooling at the datacenter level. Lastly, the CXL software ecosystem is still developing: standardized benchmarking suites, production-quality drivers for compatibility with different vendors, and application-level profiling tools are essential gaps that need to be filled for broader adoption.

VII. CONCLUSION

This survey has looked into CXL Type-3 devices regarding hardware, systems software, and applications, addressing four research questions.

RQ1 (Performance): Real CXL hardware adds 200 to 400 ns latency compared to local DRAM, with bandwidth reaching 40 GB/s for sequential reads. Performance varies widely among implementations, and NUMA emulation tends to overstate CXL overhead.

RQ2 (Systems Software): Memory tiering has progressed from basic page-fault-driven promotion (TPP) to large-scale offloading (TMO), then to contention-aware placement (Colloid) and non-exclusive caching (NOMAD). Each method addresses increasingly detailed aspects of CXL memory management.

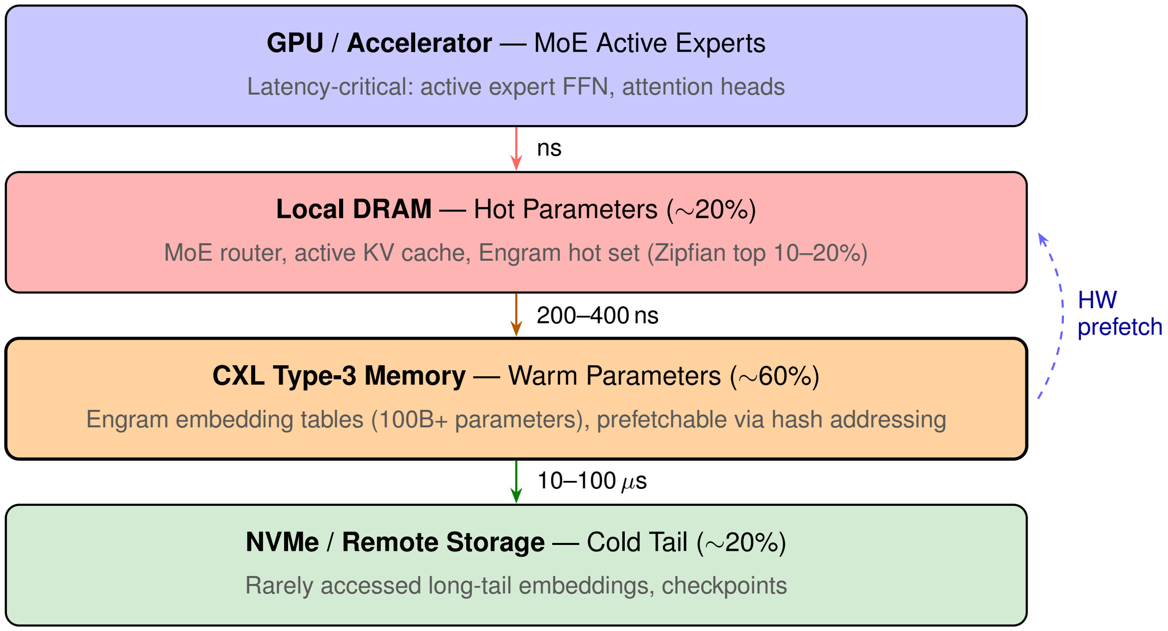
GPU / Accelerator — MoE Active Experts
Latency-critical: active expert FFN, attention heads
ns
Local DRAM — Hot Parameters (∼20%)
MoE router, active KV cache, Engram hot set (Zipfian top 10–20%)
200–400 ns
HW prefetch
CXL Type-3 Memory — Warm Parameters (∼60%)
Engram embedding tables (100B+ parameters), prefetchable via hash addressing
10–100 μs
NVMe / Remote Storage — Cold Tail (∼20%)
Rarely accessed long-tail embeddings, checkpoints

Fig. 4. Memory mapping strategy for LLMs on CXL tiered memory. Latency-critical compute parameters (active MoE experts) reside in GPU/accelerator memory, while Engram's massive embedding tables—whose Zipfian access patterns and deterministic hash addressing enable effective hardware prefetching—are offloaded to CXL Type-3 memory. The long tail migrates to NVMe storage. (Adapted from insights in [8].)

RQ3 (Applications): Databases are the primary beneficiaries, with production implementations at Alibaba (PolarCXLMem) showing 3.27× throughput gains. HPC and serverless workloads benefit from CXL's larger capacity and zero-copy forking. The AI sector, though still in early stages, shows great long-term promise because memory-augmented architectures have access patterns that fit well with CXL features.

RQ4 (Challenges): Managing tail latency, ensuring security in shared pools, and maturing the software ecosystem remain key hurdles. The most promising opportunity lies in co-designing hardware, software, and models, where CXL-aware AI architectures could greatly change the compute-memory trade-off.

CXL Type-3 devices are more than just faster memory expanders; they represent a fundamental shift in how datacenter systems manage and share memory resources. As CXL 4.0 enables multi-rack fabrics and near-memory computing, the systems and AI communities must work together to fully realize this potential.

References

[1] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Communications of the ACM*, vol. 60, pp. 48–54, 2017.

[2] C. Consortium, "Compute express link (CXL) 3.1 specification," CXL Consortium, Tech. Rep., 2023, specification version 3.1. [Online]. Available: https://computeexpresslink.org/cxl-specification/

[3] ——, "Compute express link (CXL) 4.0 specification," CXL Consortium, Tech. Rep., 2025, specification version 4.0. [Online]. Available: https://computeexpresslink.org/cxl-specification/

[4] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *IEEE International Symposium on High-Performance Comp Architecture*, New Orleans, LA, USA, 2012, pp. 1–12.

[5] X. Yang, Y. Zhang, H. Chen, F. Li, G. Fan, Y. Kong, B. Wang, J. Fang, Y. Wang, T. Huang, W. Hu, J. Kao, and J. Jiang, "Unlocking the potential of CXL for disaggregated memory in cloud-native databases," in *Companion of the 2025 International Conference on Management of Data*, ser. SIGMOD/PODS '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 689–702. [Online]. Available: https://doi.org/10.1145/3722212.3724460

[6] Y. Huang, H. Chen, N. Ni, Y. Sun, V. Chidambaram, D. Tang, and E. Witchel, "Tigon: A distributed database for a CXL pod," in *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, 2025, pp. 109–128.

[7] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan, "TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, Vancouver BC Canada, 2023, pp. 742–755.

[8] X. Cheng, W. Zeng, D. Dai, Q. Chen, B. Wang, Z. Xie, K. Huang, X. Yu, Z. Hao, Y. Li, H. Zhang, H. Zhang, D. Zhao, and W. Liang, "Conditional Memory via Scalable Lookup: A New Axis of Sparsity for Large Language Models," 2026.

[9] C. Chen, X. Zhao, G. Cheng, Y. Xu, S. Deng, and J. Yin, "Next-gen computing systems with compute express link: a comprehensive survey," 2025. [Online]. Available: https://arxiv.org/abs/2412.20249

[10] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, C. Song, J. Huang, H. Ji, S. Agarwal, J. Lou, I. Jeong, R. Wang, J. H. Ahn, T. Xu, and N. S. Kim, "Demystifying CXL memory with genuine CXL-ready systems and devices," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 105–121. [Online]. Available: https://doi.org/10.1145/3613424.3614256

[11] M. Weisgut, D. Ritter, P. Tözün, L. Benson, and T. Rabl, "Cxl memory performance for in-memory data processing," *Proceedings of the VLDB Endowment*, vol. 18, no. 9, pp. 3119–3133, 2025.

[12] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 267–278. [Online]. Available: https://doi.org/10.1145/1555754.1555789

[13] A. Klimovic, C. Kozyrakis, E. Thereska, B. John, and S. Kumar, "Flash storage disaggregation," in *Proceedings of the Eleventh European Conference on Computer Systems*, London United Kingdom, 2016, pp. 1–15.

[14] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: a hardware-software co-designed disaggregated memory system," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 417–433. [Online]. Available: https://doi.org/10.1145/3503222.3507762

[15] M. Ahn, T. Willhalm, N. May, D. Lee, S. M. Desai, D. Booss, J. Kim, N. Singh, D. Ritter, and O. Rebholz, "An examination of CXL memory use cases for in-memory database management systems using SAP HANA," *Proceedings of the VLDB Endowment*, vol. 17, no. 12, pp. 3827–3840, 2024.

[16] J. Jang, H. Choi, H. Bae, S. Lee, M. Kwon, and M. Jung, "CXL-ANNS: Software-hardware collaborative memory disaggregation and computation for billion-scale approximate nearest neighbor search," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 585–600.

[17] H. Li, D. S. Berger, S. Novakovic, L. Hsu, D. Ernst, P. Zardoshti, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, "Pond: CXL-Based Memory Pooling Systems for Cloud Platforms," 2022.

[18] X. Zhang, K. Liu, Y. Hui, X. Zheng, Y. Chang, Y. Shan, G. Zhang, K. Zhang, Y. Bao, M. Chen *et al.*, "DRack: A CXL-disaggregated rack architecture to boost inter-rack communication," in *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, 2025, pp. 1261–1279.

[19] J. Liu, H. Hadian, Y. Wang, D. S. Berger, M. Nguyen, X. Jian, S. H. Noh, and H. Li, *Systematic CXL Memory Characterization and Performance Analysis at Scale*. New York, NY, USA: Association for Computing Machinery, 2025, pp. 1203–1217. [Online]. Available: https://doi.org/10.1145/3676641.3715987

[20] D. Gouk, S. Lee, M. Kwon, and M. Jung, "Direct access, High-Performance memory disaggregation with DirectCXL," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 287–294. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/gouk

[21] Y. Zhong, D. S. Berger, P. Zardoshti, E. Saurez, J. Nelson, D. R. K. Ports, A. Psistakis, J. Fried, and A. Cidon, "Oasis: Pooling PCIe devices over CXL to boost utilization," in *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, ser. SOSP '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 101–119. [Online]. Available: https://doi.org/10.1145/3731569.3764812

[22] J. Weiner, N. Agarwal, D. Schatzberg, L. Yang, H. Wang, B. Sanouillet, B. Sharma, T. Heo, M. Jain, C. Tang, and D. Skarlatos, "Tmo: transparent memory offloading in datacenters," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 609–621. [Online]. Available: https://doi.org/10.1145/3503222.3507731

[23] M. Vuppalapati and R. Agarwal, "Tiered memory management: Access latency is the key!" in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, ser. SOSP '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 79–94. [Online]. Available: https://doi.org/10.1145/3694715.3695968

[24] L. Xiang, Z. Lin, W. Deng, H. Lu, J. Rao, Y. Yuan, and R. Wang, "Nomad: Non-Exclusive memory tiering via transactional page migration," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 19–35. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/xiang

[25] Y. Zhong, D. S. Berger, C. Waldspurger, R. Wee, I. Agarwal, R. Agarwal, F. Hady, K. Kumar, M. D. Hill, M. Chowdhury, and A. Cidon, "Managing memory tiers with CXL in virtualized environments," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 37–56. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/zhong-yuhong

[26] M. Zhang, T. Ma, J. Hua, Z. Liu, K. Chen, N. Ding, F. Du, J. Jiang, T. Ma, and Y. Wu, "Partial failure resilient memory management system for (CXL-based) distributed shared memory," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 658–674. [Online]. Available: https://doi.org/10.1145/3600006.3613135

[27] C. Alverti, S. Psomadakis, B. Ocalan, S. Jaiswal, T. Xu, and J. Torrellas, *CXLfork: Fast Remote Fork over CXL Fabrics*. New York, NY, USA: Association for Computing Machinery, 2025, pp. 210–226. [Online]. Available: https://doi.org/10.1145/3676641.3715988

[28] Z. Wang, Y. Guo, K. Lu, J. Wan, D. Wang, T. Yao, and H. Wu, "RCMP: Reconstructing RDMA-based memory disaggregation via CXL," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 1, Jan. 2024. [Online]. Available: https://doi.org/10.1145/3634916

[29] Z. Wang, Y. Chen, C. Li, Y. Guan, D. Niu, T. Guan, Z. Du, X. Wei, and G. Sun, "CTXNL: A software-hardware co-designed solution for efficient CXL-based transaction processing," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 192–209. [Online]. Available: https://doi.org/10.1145/3676641.3716244

[30] Y. Fridman, S. Mutalik Desai, N. Singh, T. Willhalm, and G. Oren, "CXL memory as persistent memory for disaggregated HPC: A practical approach," in *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 983–994. [Online]. Available: https://doi.org/10.1145/3624062.3624175

[31] R. Xie, A. U. Haq, L. Ma, K. Sun, S. Sen, S. Venkataramani, L. Liu, and T. Zhang, "SmartQuant: CXL-based AI model store in support of runtime configurable weight quantization," *IEEE Computer Architecture Letters*, vol. 23, no. 2, pp. 199–202, 2024.

[32] S. Yun, H. Nam, K. Kyung, J. Park, B. Kim, Y. Kwon, E. Lee, and J. H. Ahn, "CLAY: CXL-based scalable NDP architecture accelerating embedding layers," in *Proceedings of the 38th ACM International Conference on Supercomputing*, ser. ICS '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 338–351. [Online]. Available: https://doi.org/10.1145/3650200.3656595

[33] S.-S. Park, K. Kim, J. So, J. Jung, J. Lee, K. Woo, N. Kim, Y. Lee, H. Kim, Y. Kwon, J. Kim, J. Lee, Y. Cho, Y. Tai, J. Cho, H. Song, J. H. Ahn, and N. S. Kim, "An LPDDR-based CXL-PNM platform for TCO-efficient inference of transformer-based large language models," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 970–982.

[34] R. Abdullah, H. Lee, H. Zhou, and A. Awad, "Salus: Efficient security support for CXL-expanded GPU memory," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 1–15.

[35] J. Dong, J. Rosenblum, and S. Narayanasamy, "Toleo: Scaling freshness to tera-scale memory using CXL and PIM," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2025, pp. 313–328. [Online]. Available: https://doi.org/10.1145/3622781.3674180