Lost in Translation: The Search for Meaning in Network-Attached AI Accelerator Disaggregation

Jaewan Hong
UC Berkeley

Yifan Qiao
UC Berkeley

Soujanya Ponnapalli
UC Berkeley

Shu Liu
UC Berkeley

Marcos K. Aguilera
NVIDIA

Vincent Liu
University of Pennsylvania

Christopher J. Rossbach
UT Austin & Microsoft

Ion Stoica
UC Berkeley

Abstract

Datacenters often underutilize expensive AI accelerators (GPUs, TPUs, etc). A natural solution is disaggregation, where servers borrow network-attached accelerators on demand. However, current approaches to disaggregation suffer from a *semantic translation gap*: as computation descends the software stack, critical application knowledge—like model structure or execution phases—is lost. This forces an undesirable choice between low-level, general-purpose systems that are semantically-blind and inefficient, and high-level, single-workload systems that are efficient but not general.

We argue that the machine-learning framework layer is the natural narrow waist for accelerator disaggregation. We present Genie, a framework-layer architecture centered on a Semantically-Rich Graph (SRG) that decouples *intent capture* from *execution*. By deferring execution, Genie automatically builds a semantically-rich compute graph, enabling it to apply workload-specific optimizations and orchestrate a zero-copy data path for remote execution. Genie demonstrates a new path toward AI infrastructure that is at once general, semantically-aware, and efficient.

CCS Concepts

• **Computer systems organization** → **Distributed architectures**; • **Networks** → *Cloud computing*; • **Computing methodologies** → *Machine learning*.

HotNets '25, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/25/11
https://doi.org/10.1145/3772356.3772379

Keywords

AI accelerators, disaggregation, network-attached accelerators, machine learning frameworks, semantic awareness

ACM Reference Format:
Jaewan Hong, Yifan Qiao, Soujanya Ponnapalli, Shu Liu, Marcos K. Aguilera, Vincent Liu, Christopher J. Rossbach, and Ion Stoica. 2025. Lost in Translation: The Search for Meaning in Network-Attached AI Accelerator Disaggregation. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3772356.3772379

1 Introduction

The rapid proliferation of AI has driven unprecedented investment in accelerators—over $150B in 2023 alone [12]—yet real fleets still report 55–60% average GPU idleness [13]. This severe underutilization is a consequence of today's coarse-grained resource allocation mechanisms and tightly-coupled server-accelerator architectures, both of which strand expensive AI accelerators for applications with fluctuating demands or disproportionate resource requirements.

A natural strategy to avoid this wasted capacity is resource disaggregation, which decouples AI accelerators into network-attached, shareable pools [7–9, 14][1]. Applications can then dynamically claim the exact type and count of accelerators for the duration they need them.

However, existing proposals for GPU disaggregation largely follow patterns from other contexts that are ill-suited for AI workloads. One approach is to hand explicit control over remote resources to the applications themselves [18, 20]. These solutions, while efficient, necessitate extensive hand-tuning or are tailored for individual workloads and model architectures. The other common approach is to emulate the successes of storage disaggregation and operate at low levels of the stack (PCIe or driver-call replay). These schemes are more general and backward compatible, but they are blind to

[1]We focus on hardware disaggregation, where physical accelerators are the shared resource, as distinct from model disaggregation (e.g., model parallelism), where a single large model is partitioned across dedicated accelerators.

semantic information from the application (e.g., phase boundaries, data dependencies, and variances in latency sensitivity), resulting in poor performance. For instance, a network appliance that sees only DMA bursts cannot tell a reusable model weight from a one-off activation; a driver shim cannot identify the difference between an LLM's prefill phase (which is compute-bound and should be parallelized as much as possible) and its decode phase (which is memory-bound and should be co-located with a remote KV cache).

In this paper, we argue that ML frameworks are the ideal *narrow waist* for accelerator disaggregation. ML frameworks are general enough to support a vast and diverse range of AI models and hardware. Crucially, they also observe application intent: model structure, execution phases (e.g., LLM prefill vs. decode), data dependencies, and residency. These semantics enable optimizations that are invisible to lower layers (e.g., co-locating decode with KV cache, pipelining convolutional stages, or recomputing cheap intermediates under congestion) without hard-coding per-application logic. Our thesis is that leveraging these semantics is the key to making accelerator disaggregation practical and efficient.

We introduce Genie[2], a framework-level disaggregation architecture that bridges the semantic translation gap while preserving generality. Genie is built around a Semantically Rich Graph (SRG), a portable abstraction that cleanly separates *what* the application intends from *how/where* it executes. Framework-specific frontends (we prototype PyTorch) capture application intent and construct an SRG, annotating it with high-level cues like execution phase and data residency. A pluggable scheduler consumes the SRG to make intelligent placement and data movement decisions, exploiting semantic cues to reduce data motion and expose parallelism. The scheduler also enables fleet-wide, multi-tenant optimization when used at datacenter scale by talking to a global coordinator. Backends execute the plan on remote accelerators via user-space networking. Genie supports commodity clients (RNIC optional) and assumes RNIC-equipped disaggregated servers; when RDMA and GPUDirect are available, the server datapath achieves NIC-to-GPU zero-copy. Because the SRG represents a complete, replayable lineage of the computation, this architecture also provides a natural foundation for targeted, lineage-based fault tolerance.

Together, these ideas chart a path where developers write standard framework code, while a semantics-aware runtime and scheduler decide what to run, where, and when—turning disaggregation from a hardware trick into an application-informed systems capability.

[2]Freeing accelerators from the lamp of one server, to unleash their potential.

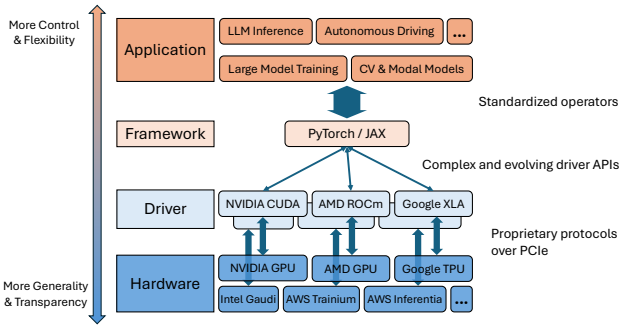

Figure 1: ML Framework provides general support to applications and unifies heterogeneous hardware.

2 The Semantic Translation Gap

To disaggregate, we need to pick a layer of the software stack to break apart the user from the remote accelerator, by redirecting local invocations to remote invocations. On one hand, to hide the overheads of disaggregation, we need to pick a layer that preserves application semantics, to enable optimizations that hide latency. But, as we move down the stack, we lose application semantics—something we call the *semantic translation gap*—this calls for picking a high layer for disaggregation. On the other hand, picking too high a layer reduces generality, forcing developers to manage data movement and management explicitly. We give concrete examples of this trade-off below.

2.1 A Tour of the Disaggregation Landscape

Modern AI software is multilayered (Fig. 1). We summarize three candidates for disaggregation points and the semantics they would *lose*.

PCIe-level disaggregation (e.g., DxPU [6]) operates at the hardware interface, forwarding PCIe transactions over the network. This approach offers full application transparency but is semantically blind. Without application context, it cannot distinguish a critical-path tensor from an ephemeral one, nor can it differentiate latency-sensitive KV cache access from a bulk weight transfer in an LLM. This inability to understand workload intent prevents intelligent scheduling and data management. Furthermore, the PCIe protocol is ill-suited for higher-latency networking; for instance, posted writes can quickly exhaust the limited pool of transaction tags, severely throttling throughput.

Driver-level disaggregation (e.g., [1, 2, 4, 5, 15]) intercepts GPU driver APIs like CUDA and routes them over the network. While this approach avoids the need for specialized hardware for disaggregation, it still misses significant context, e.g., every copy looks urgent, and every kernel looks equal. Another issue is that driver APIs such as CUDA are rapidly changing, which requires constant re-implementation of the disaggregation layer.

Application-specific disaggregation At the highest level of the stack, it is possible for programmers to effectively use remote accelerators by leveraging deep, workload-specific knowledge to integrate orchestration and control operations directly into the applications themselves, albeit at the steep price of generality. This typically manifests in one of two different approaches.

The first is through bespoke systems like DistServe [20] and Prism [18]. These systems are architecturally tailored for a specific workload class, such as LLM inference or recommendation models. Prism, for instance, leverages DLRM semantics to co-design data layout and execution for that model only. Their deep semantic integration yields high efficiency but cannot be repurposed for other model types.

The second form relies on manual developer optimization using toolkits like PyTorch RPC [3]. Although the library is part of the PyTorch framework, effectively using it requires application-specific engineering. Developers must manually refactor code, decide which functions to execute remotely, manage state, and reason about data locality for their particular application. As every application is different, this process can be time-consuming and difficult to get right.

2.2 Consequences of the Translation Gap

Existing general-purpose disaggregation systems operate at too low a level, losing application semantics and optimization opportunities. For example, consider an LLM. Code structure and/or profiling results reveal distinct phases with distinct properties, e.g., prefill (compute-bound; parallelizable) and decode (memory-bound; sequential; dependencies on KV cache). A naive, semantically-blind approach might treat each GPU operation as independent and identical, for example, by spreading each request across available GPU resources with a round-robin or least-loaded policy. The result would be excessive data transfers due to repeated moves of large KV caches. One could do slightly better by considering the data movement costs of each placement and scheduling decision, i.e., treating operations as independent but not necessarily identical. This might save data transfer costs by ensuring that all subsequent calls are scheduled on the same GPUs; however, it would still entirely miss the potential benefits of PD disaggregation [11]. In the end, many potential optimizations are only possible with a broad and deep understanding of the general application structure.

The above limitations will only become more salient as modern AI clusters increasingly serve diverse workloads with fundamentally different characteristics. At a low level, these workloads can vary in the models they use, the size of the requested models, their modalities, their SLO expectations (on-demand vs batch), and their reasoning requirements. More broadly, AI has found its way into a wide range of applications, a small selection of which are illustrated in Table 1. LLMs exhibit sequential dependencies and phase-based computation, computer vision models have regular data flow with layer-wise parallelism, and recommendation systems combine sparse embedding lookups with dense neural computation. Generic disaggregation policies cannot optimize for this diversity without a semantic understanding of each workload's unique characteristics.

The framework layer as the narrow waist. These challenges and missed opportunities across the stack point to a single conclusion: the ideal layer for disaggregation must be both semantically rich and sufficiently general. The *machine-learning framework* is the only layer that satisfies both criteria. Our thesis is to standardize this intent into a *Semantically-Rich Graph (SRG)* that decouples capture (*what* the program wants to compute) from execution (*how/where* it runs). SRGs are dynamic enough to capture runtime shapes and control flow, yet structured enough to drive placement and data movement. Although our prototype targets PyTorch, the SRG abstraction is not PyTorch-specific (JAX/TF would map via StableHLO/MLIR plus lightweight runtime hooks for phases and residency).

This choice has three consequences. First, it exposes a minimal set of semantics—phases, dependencies, residency, and modality—that generalize across workload families. Second, it enables cost-aware policies that reduce data motion, colocate stateful phases, and pipeline independent subgraphs. Third, it creates a portable interface to cluster schedulers and backends, paving the way for multi-tenant, semantics-aware placement at datacenter scale.

3 The Genie Platform

Genie's design is predicated on a clean architectural separation between an application's computational intent and its physical execution. This is enabled by our core abstraction: the Semantically Rich Graph (SRG), a portable intermediate representation that serves as a durable "narrow waist."

The architecture is a three-stage pipeline. Frontends are responsible for the challenging task of transparently capturing application intent and translating it into a standardized SRG. The Scheduler consumes this SRG, treating it as a declarative specification of requirements, and produces an optimized, device-specific execution plan. Finally, Backends are the concrete execution agents that realize this plan on physical hardware.

This design decouples the problem: ML framework experts can design frontends, distributed systems experts can design schedulers, and hardware experts can design backends, all while interoperating through the common SRG interface.

Workload	Computation Pattern	Memory Access	Key Optimization
LLM Serving	Sequential, phased (prefill/decode)	Streaming KV cache	Phase-aware allocation
Computer Vision	Layer-parallel, regular	Predictable feature maps	Pipeline parallelism
Recommendation	Sparse + dense mix	Hot/cold embeddings	Intelligent data tiering
Multi-modal	Cross-modal fusion	Heterogeneous patterns	Modality-aware placement

Table 1: Semantic characteristics of representative AI workloads reveal diverse optimization opportunities inaccessible to semantically-blind systems.

3.1 The Semantically Rich Graph

The SRG is a declarative data structure, not an executable program. It is a directed acyclic graph where nodes represent operations (from a single kernel to a large fused subgraph) and edges represent data dependencies. Unlike traditional computation graphs, the SRG is designed to be a self-contained specification for a scheduler.
Nodes carry a common annotation schema:

- *Phase*: A tag identifying the execution phase (e.g., *llm_prefill*, *llm_decode*). This is crucial for phase-aware resource management.
- *Residency*: Metadata on the intended lifetime and properties of data products (e.g., *persistent_weight*, *ephemeral_activation*, *stateful_kv_cache*). This directly informs caching, placement, and data movement decisions.
- *Modality*: Tags identifying the data type being processed (e.g., vision, text), allowing placement on specialized accelerators.
- *Cost hints*: Profiling- or model-based estimates of computational cost (FLOPs), memory footprint (bytes), and operational intensity.

Edges carry encodings of the data movement costs:

- *Tensor Metadata*: Describes the shape, precision, and layout of the data flowing between nodes.
- *Producer-Consumer Rates*: Specifies data volume changes (e.g., for sampling operators), which are critical for network bandwidth reservation.
- *Criticality*: A tag indicating if this data dependency is on the critical path of execution, helping the scheduler prioritize transfers.

This schema is the contract between the frontend and the scheduler. It provides the scheduler with the necessary information to make intelligent decisions without needing to understand the internals of the source ML framework.

3.2 Frontends (Capturing Intent)

The most significant challenge is capturing high-level intent without burdening the application developer. Our PyTorch frontend employs a multi-tiered approach to bridge this gap, acknowledging that full automation is not always possible.

Automated Graph Construction: At the base layer, we use PyTorch's `__torch_dispatch__` mechanism to defer execution and build a fine-grained graph of operations using LazyTensor proxies. This captures the raw dependency structure and tensor properties automatically.

Automated Structural Annotation: We then use an FX pass to traverse the graph and group operations based on the application's `nn.Module` hierarchy. This automatically reveals structural patterns (e.g., "this sequence of ops belongs to the AttentionBlock module").

Semi-Automated Semantic Annotation: High-level semantics like "decode phase" are often implicit. Our frontend provides two mechanisms here. The primary method uses a library of pattern recognizers that identify common model idioms (e.g., a recurrent loop with a growing KV cache is characteristic of LLM decoding). For novel architectures, developers can provide optional, explicit module-level hooks (e.g., `genie.annotate_phase(self.decoder, "decode")`). This approach provides a practical path to adoption: most common models work out-of-the-box, while new ones require minimal, high-level annotations.

This tiered process culminates in the emission of a clean, portable SRG. While we have prototyped in PyTorch, the SRG is a viable compilation target for other frameworks. For example, a JAX frontend could lower its jaxpr representation to an SRG. Acknowledging the engineering complexity, the key is that the SRG provides a stable, common target for such efforts.

3.3 Scheduler: Semantics-Driven Optimizations

The scheduler is a pluggable policy engine that translates a declarative SRG into a concrete execution plan. Its core interface is a pure function: `plan = schedule(srg, cluster_state, policy)`. It takes an SRG, a representation of the current hardware availability and network topology, and a policy module (e.g., *minimize_latency* and returns an annotated SRG. This output SRG is augmented with concrete device assignments for each node and explicit send/receive instructions for each edge.

Policies implement the system's optimization logic using a cost model that evaluates the end-to-end latency of any given plan. These policies leverage the SRG's semantic annotations

to apply powerful, context-aware optimizations that are not hard-coded:

- **Stateful Co-location.** By identifying the sequential, KV-cache-dependent nature of the decode phase from the graph, Genie ensures the cache and the decoder layers are pinned to the same remote GPU. This eliminates repeated, costly transfers of the entire cache state.
- **Pipelined CNN inference.** For vision workloads, the graph reveals consecutive convolutional stages. Genie can automatically fuse these stages and schedule them as a pipeline across multiple accelerators, effectively overlapping communication and computation.
- **Dynamic recomputation.** By analyzing the graph and querying network conditions, Genie can make intelligent trade-offs. When network contention is high, it can opt to recompute an inexpensive intermediate tensor on the remote device instead of waiting to fetch it across the congested wire.

A pluggable cost model estimates end-to-end latency as a function of compute, transfers, and queuing. The scheduler emits an annotated SRG with device bindings, transfer schedules, and caching directives. Developers can supply policy plugins at three extension points: graph rewrites (prepass), placement policy, and runtime hint adaptation (e.g., using measured RTT).

3.4 Execution Backends: High-Performance Datapaths

The backend translates the scheduler's static plan into dynamic execution. Its interface is: `results = execute(annotated_srg)`.

A critical source of overhead in accelerator disaggregation is redundant data movement. Genie allocates tensors in network-ready pinned buffers at creation time, avoiding reactive pinning and extra copies. Remote-sesident objects (weights, KV caches, etc) are materialized once and referenced by opaque handles.

Genie proactively integrates tensor allocation with a user-space networking stack (DPDK) from the outset. Specifically, when an application creates a tensor, Genie intercepts this operation and directly allocates the tensor in pinned, network-ready host memory managed by DPDK. This proactive approach completely eliminates the initial copy overhead associated with reactive pinning (calling `pin_memory()` post-hoc).

To achieve a true zero-copy data path between GPUs and NICs, Genie leverages DPDK's `gpudev` abstraction, which provides a vendor-agnostic interface for GPU memory management and data transfers. Under the hood, `gpudev` integrates vendor-specific technologies such as NVIDIA GPUDirect RDMA, enabling NICs to directly DMA data into GPU memory without intermediate CPU involvement. Importantly, this capability relies on vendor-supported GPU-NIC integration mechanisms and appropriate OS-level configurations (e.g., IOMMU settings, PCIe ACS configurations). Clients can be equipped with commodity NICs without RDMA capabilities but still take advantage of zero-copy transfers.

The backend executes kernels, orchestrates scheduled transfers, and tracks per-node completion and resource usage.

3.5 Lineage-Based Fault Tolerance

Genie provides a lineage-based fault-tolerance model inspired by dataflow systems [10, 17, 19]. The SRG is the unit of lineage: nodes are deterministic operator invocations; edges are explicit dependencies. Remote resident objects are referenced by opaque handles with epochs. Failures trigger selective recomputation guided by the SRG. Upon detecting a failure, the runtime invalidates affected handles, rebinds to new resources, and replays only the subgraph on the cut induced by the lost state. Idempotence is guaranteed by scoping side effects to handle+epoch and by materializing external outputs only after commit points. Lineage spans phases, enabling recovery of long-running decode loops without restarting prefill.

3.6 Semantics-Aware Global Scheduling

The semantic-rich computation graph constructed by Genie is not merely a local optimization tool—it is the foundational building block for a broader vision of autonomous, semantics-aware resource management at datacenter scale. In this vision, Genie instances act as clients to a global scheduler, providing it with semantic graphs as a rich, first-class description of workload requirements.

Armed with this fleet-wide semantic context, the global scheduler can make resource allocation decisions that are impossible for systems that are blind to application intent. It can determine *where*, *when*, and *how* each operation should execute across thousands of tenants:

- **Where (Heterogeneous Placement):** The scheduler moves beyond simple co-location to perform true heterogeneous placement. It can analyze the semantic graphs to identify workload classes, placing all vision-transformer jobs on memory-bandwidth-optimized GPUs while scheduling recommendation models on accelerators with different characteristics, globally optimizing for hardware affinity.
- **When (Elastic Scaling):** Leveraging semantic phase annotations (e.g., prefill vs. decode) from the graph, the scheduler can dynamically provision and release resources as a workload's needs evolve in real-time. It can scale out accelerators for a burst of parallelizable prefill tasks and scale them back during the sequential decode phase.

- **How (Cross-Workload Orchestration):** The scheduler can use semantic metadata to orchestrate execution *across* tenants. For example, it could identify two separate user requests that use the same public LLM and automatically batch their decode steps together to improve throughput, or prioritize interactive, latency-sensitive VQA queries over long-running batch training jobs.

This interactive, semantics-driven negotiation between Genie and the global scheduler enables unprecedented elasticity and efficiency. Unlike static, compiler-based approaches, this dynamic co-adaptation continuously optimizes resource allocation and execution strategies in response to evolving workloads, network conditions, and resource availability. Our current implementation of Genie lays the critical groundwork for this ambitious vision, demonstrating the feasibility and performance potential of semantic-driven disaggregation at scale.

3.7 Limitations and Scope

The SRG model, like all abstractions, has limitations. Highly dynamic control flow (data-dependent if branches) requires the frontend to either conservatively unroll small loops or insert "re-capture" points where a new SRG is generated mid-execution. Opaque custom operators (e.g., a user-written CUDA kernel) are another challenge; the frontend can capture their I/O signatures but must rely on developer-provided annotations for their internal semantics and cost. Our current work focuses on the large class of models where the computational graph is largely static within a given execution phase.

4 Evaluation

This section answers a key question: *How much does semantic awareness matter when executing a modern LLM across a disaggregated network?* We evaluate four execution modes that span the design space:

Local (Upper Bound) model and KV-cache reside on the same A100-80 GB GPU as the client.

Semantics-Blind, Naïve client re-uploads the entire 12 GB GPT-J model weights on *every* remote call. During the decode phase, this occurs for each generated token, and the KV-cache is not preserved between steps.

Semantics-Blind, ΔKV weights remain remote; each step ships the *delta* slice of the KV-cache (~1.0 MB) without compression.

Semantics-Aware KV-cache is pinned on the remote GPU and referred to by a tiny handle; each round trip moves only the next token and its resulting logits.

Phase / Mode	Latency [s]	Net [MB]	GPU Util. [%]
Prefill			
Local (Upper Bound)	0.21	0.0	100.0
Naïve	216	149,258	0.1
ΔKV	110	4.31	0.2
Semantics-Aware	111	5.56	0.2
Decode (50 tokens)			
Local (Upper Bound)	1.53	0.0	99.1
Naïve	783	95,438	0.3
ΔKV	131	52.3	1.5
Semantics-Aware	116	11.3	1.8

Table 2: End-to-end latency, network traffic, and effective GPU utilization for a 72-token prompt + 50-token decode. While useful GPU work is virtually identical across modes, wall-clock time and utilization vary dramatically due to data movement and RPC overhead.

Setup. The GPT-J model runs on a single A100-80 GB server, while the CPU-only client connects through a 25 Gbps link. We use PyTorch 2.1 with its off-the-shelf TensorPipe [16] RPC transport. No RDMA extensions are enabled, making our comparison conservative for the semantics-aware design. We measure end-to-end **latency** (`/usr/bin/time`), **network volume** (via RPC counters), and **effective GPU utilization**, calculated as the total kernel time divided by the wall-clock latency. Metrics are reported separately for the compute-heavy *prefill* phase (processing a 72-token prompt) and a 50-step autoregressive *decode* loop.

Results. Table 2 presents the results for generating 50 tokens. Three clear trends emerge.

Semantic cues slash network traffic. Blindly re-uploading weights overwhelms the network, accounting for 149 GB in the prefill phase. For the naïve decode, this process generates a total of 95 GB over 50 steps. By understanding that weights are immutable and the KV cache should stay near compute, the Semantics-Aware mode reduces traffic by over 8,400× compared to naïve decode (11 MB vs. 95 GB) and by over 26,000× in the prefill phase.

GPU idles without semantic context. The penalty of semantic blindness is most stark when looking at GPU utilization. While all modes execute a nearly identical amount of useful work (~1.9 s of total kernel time), their end-to-end GPU utilization plummets in the remote scenarios. In the Naïve and ΔKV modes, the GPU is idle over 98% of the time, completely bottlenecked by data transfers. Our Semantics-Aware approach improves utilization by 6× over the Naïve mode, but the GPU still remains heavily underutilized, highlighting the performance is now dominated by the overhead of the RPC transport itself, not data volume.

Mode	Decode latency for N tokens [s]			
	N=50	N=100	N=150	N=200
ΔKV	132.0	159.9	181.8	204.3
Semantics-Aware	114.0	118.4	118.5	119.2

Table 3: Decode latency scaling with generation length.

Latency becomes RPC-bound, not data-bound. The remaining performance gap between the Semantics-Aware and local modes is almost entirely an artifact of the unoptimized Python RPC transport, which requires one synchronous round-trip per token. Replacing this with a zero-copy RDMA path is orthogonal work (§3.4) that would tighten the gap but not change the relative ordering of the designs.

Table 3 shows how latency scales with generation length. The Semantics-Aware mode's latency is dominated by a fixed per-token RPC overhead and thus remains nearly constant. In contrast, the ΔKV mode's latency grows linearly as it must transfer a larger KV-cache slice with each step. By 200 tokens, the Semantics-Aware design is already ∼1.7× faster.

Key takeaways. (1) Semantic awareness eliminates orders of magnitude in network traffic and improves GPU utilization by revealing the true system bottlenecks. (2) With an off-the-shelf RPC stack, our Semantics-Aware mode already closes 88% of the latency gap relative to the next-best semantics-blind approach (ΔKV). (3) Effective accelerator disaggregation is only practical when the runtime understands high-level application intent; blind, general-purpose transports strand expensive GPU resources by wasting bandwidth and time.

These findings motivate Genie: a framework-level, semantically-aware runtime with a zero-copy transport designed to realize the full potential of accelerator disaggregation.

5 Research Challenges

Realizing the full potential of semantic disaggregation requires addressing new research challenges. We identify the following fundamental challenges that must be addressed before framework-layer disaggregation can be considered truly viable.

Semantics-aware multi-tenant scheduling at scale. How can we efficiently represent and reason about semantic graphs at scale? What algorithms can dynamically co-adapt resource allocation across thousands of concurrent, semantically-rich workloads? These questions require new scheduling algorithms that leverage semantic graphs as first-class inputs.

The semantic boundary: Dynamicism and Control Flow. The SRG's power relies on capturing application intent. However, the current SRG model, like most graph-based compilation, excels at static, ahead-of-time (AOT) graphs. This is a severe limitation.

The frontier of ML is intensely dynamic: data-dependent control flow, dynamic shapes, just-in-time (JIT) compilation, and complex conditional logic (e.g., Mixture of Experts) are becoming the norm. The core research challenge is: How do we build an SRG that is both rich enough to optimize and fluid enough to capture true dynamicism?

A purely static graph will fail. A purely JIT-based approach may lack the global view needed for disaggregation. This is a fundamental trade-off between expressiveness and tractability that sits at the heart of the Genie model.

Trust and verifiability in semantic disaggregation. Semantic graphs dictate resource usage and sensitive data movement across networks. Ensuring security, privacy, and verifiability becomes paramount. How can we protect proprietary model architectures encapsulated in semantic graphs, ensure data isolation in multi-tenant remote memory, and develop verifiable computation schemes where clients cryptographically verify remote computations based on semantic plans? Addressing these challenges opens avenues for "confidential AI disaggregation."

The evolving semantic lexicon: beyond hand-crafted rules. How can systems like Genie automatically learn or infer the semantic roles of operations and patterns in novel, unseen AI architectures (e.g., emerging State Space Models, Liquid Neural Networks)? Can we develop techniques for self-evolving semantic vocabularies, perhaps using meta-learning on computation graphs or program synthesis to generate new optimization heuristics for disaggregation, moving beyond manually curated pattern recognizers?

6 Conclusion

We identify the ML framework as the true narrow waist for this problem—the only layer that is both general-purpose and semantically-aware. We present Genie, a framework-layer architecture built upon the core abstraction, the Semantically-Rich Graph (SRG), cleanly decouples computational intent from execution. This allows the system to capture high-level application knowledge, like execution phases and data residency, that is completely opaque to low-level hardware-centric approaches.

Our evaluation demonstrates that this semantic awareness is not an incremental optimization; it is the key to viability. By transforming application intent into an efficient, zero-copy data path, Genie converts the disaggregation problem from fundamentally network-bound to compute-bound, eliminating the orders-of-magnitude in wasteful data transfers that render low-level disaggregation impractical.

Acknowledgements

This work was supported by the National Science Foundation under Grant No. CCF-2326606.

References

[1] Rachata Ausavarungnirun, Joshua Landgraf, Vance Miller, Saugata Ghose, Jayneel Gandhi, Christopher J Rossbach, and Onur Mutlu. Mosaic: a gpu memory manager with application-transparent support for multiple page sizes. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 136–150, 2017.

[2] Broadcom. End of availability (eoa) for vsphere bitfusion, January 2025. Accessed: 2025-01-15.

[3] Pritam Damania, Shen Li, Alban Desmaison, Alisson Azzolini, Brian Vaughan, Edward Yang, Gregory Chanan, Guoqiang Jerry Chen, Hongyi Jia, Howard Huang, et al. Pytorch rpc: Distributed deep learning built on tensor-optimized remote procedure calls. *Proceedings of Machine Learning and Systems*, 5:219–231, 2023.

[4] José Duato, Antonio J Pena, Federico Silla, Rafael Mayo, and Enrique S Quintana-Ortí. rcuda: Reducing the number of gpu-based accelerators in high performance clusters. In *2010 International Conference on High Performance Computing & Simulation*, pages 224–231. IEEE, 2010.

[5] Henrique Fingler, Zhiting Zhu, Esther Yoon, Zhipeng Jia, Emmett Witchel, and Christopher J Rossbach. Dgsf: Disaggregated gpus for serverless functions. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 739–750. IEEE, 2022.

[6] Bowen He, Xiao Zheng, Yuan Chen, Weinan Li, Yajin Zhou, Xin Long, Pengcheng Zhang, Xiaowei Lu, Linquan Jiang, Qiang Liu, et al. Dxpu: Large-scale disaggregated gpu pools in the datacenter. *ACM Transactions on Architecture and Code Optimization*, 20(4):1–23, 2023.

[7] Jaewan Hong, Marcos K. Aguilera, Emmanuel Amaro, Vincent Liu, Aurojit Panda, and Ion Stoica. The dawn of disaggregation and the coherence conundrum: A call for federated coherence, 2025.

[8] Kevin Lim, Yoshio Turner, Jose Renato Santos, Alvin AuYoung, Jichuan Chang, Parthasarathy Ranganathan, and Thomas F Wenisch. System-level implications of disaggregated memory. In *IEEE International Symposium on High-Performance Comp Architecture*, pages 1–12. IEEE, 2012.

[9] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. Gimbal: enabling multi-tenant storage disaggregation on smartnic jbofs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 106–122, 2021.

[10] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 561–577. USENIX Association, 2018.

[11] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. *arXiv preprint arXiv:2311.18677*, 2023.

[12] Research and Markets. Data center accelerators strategic business research report 2024: Global market to reach $395 billion by 2030 - ai-powered accelerators mark promising upheaval with energy-efficient datacenters, September 2024. Accessed: 2025-01-15.

[13] Owen Rogers. Tens of thousands of gpus go under-utilized in the cloud, January 2024. Accessed: 2025-01-15.

[14] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. {LegoOS}: A disseminated, distributed {OS} for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 69–87, 2018.

[15] Lin Shi, Hao Chen, Jianhua Sun, and Kenli Li. vcuda: Gpu accelerated high performance computing in virtual machines. *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–11, 2009.

[16] PyTorch Team. Tensorpipe: A tensor-aware point-to-point communication library. https://github.com/pytorch/tensorpipe, 2020. Commit reference: main branch, accessed October 2025.

[17] Stephanie Wang, William Paul, Eric Liang, Robert Nishihara, Philipp Moritz, Ion Stoica, and Alexey Tumanov. Ownership: A distributed futures system for fine-grained tasks. In *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 995–1010. USENIX Association, 2021.

[18] Lingyun Yang, Yongchen Wang, Yinghao Yu, Qizhen Weng, Jianbo Dong, Kan Liu, Chi Zhang, Yanyi Zi, Hao Li, Zechao Zhang, et al. {GPU-Disaggregated} serving for deep learning recommendation models at scale. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 847–863, 2025.

[19] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[20] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '24, pages 331–348. USENIX Association, 2024.