SciForce

Journal of Artificial Intelligence and Machine Learning

Journal homepage: www.sciforce.org
ISSN : 2995-2336

Open Access **Review Article**

Architecting Retail-Scale Product Knowledge Graph Systems

Karthik Perikala*

Technology Leader, The Home Depot., United States

Abstract

Modern retail platforms increasingly depend on rich product understanding to support search, recommendation, comparison, and conversational assistance experiences. These capabilities require structured representation of products, attributes, relationships, and contextual signals that evolve continuously across the catalog lifecycle. Product knowledge graphs have emerged as a foundational abstraction for modeling this interconnected data at scale.

This paper presents a generalized architecture for retail-scale product knowledge graph systems designed to support low-latency read access alongside continuous graph updates driven by catalog ingestion, enrichment pipelines, and behavioral signals. The architecture emphasizes separation of concerns between ingestion, graph construction, serving, and downstream consumption, enabling predictable performance under read-dominated and mixed workloads.

We introduce a workload taxonomy for product knowledge graph access patterns, outline graph modeling and lifecycle design principles, and propose an empirical evaluation methodology that focuses on normalized read and write latency behavior, traversal efficiency, and operational stability. Observed performance trends demonstrate that well-partitioned and access-aware graph systems can sustain retail-scale workloads with bounded tail-latency characteristics.

Keywords: Product Knowledge Graphs, Retail Systems, Graph Databases, Search and Recommendations, Low-Latency Serving, Data Architecture

Introduction

Retail product ecosystems are inherently relational. Products are associated with categories, attributes, variants, accessories, compatibility rules, and contextual metadata that together define how they should be discovered, compared, and recommended. As catalogs grow in size and dimensionality, representing and serving these relationships efficiently becomes a central architectural challenge.

Traditional relational and document-oriented data models struggle to express multi-hop relationships and evolving graph structure without incurring excessive join complexity or denormalization overhead. In response, many retail platforms have adopted graph-based abstractions to model product knowledge in a form that aligns naturally with traversal-based access patterns used by search ranking, recommendation pipelines, and conversational agents.

However, operationalizing product knowledge graphs at retail scale introduces non-trivial challenges. These systems must support high-concurrency, low-latency read workloads while simultaneously accommodating frequent updates from batch refreshes, incremental enrichment, and near real-time signal ingestion. Poor partitioning, unbounded fan-out, or schema rigidity can lead to performance degradation that disproportionately impacts tail latency.

This paper presents a reference architecture for retail-scale product knowledge graph systems that balances modeling flexibility with serving efficiency. We formalize common workload archetypes, describe access-aware graph design patterns, and introduce an evaluation framework that captures realistic operational behavior. The goal is to provide practitioners with reusable guidance for building scalable, robust product knowledge foundations that power modern retail experiences.

System Overview

A retail-scale product knowledge graph system functions as a connective substrate between heterogeneous product data sources and downstream retrieval-driven services. Rather than operating as a single monolithic database, the system coordinates ingestion, graph construction, serving, and consumption stages, each optimized for distinct operational and performance concerns.

Upstream inputs include structured product catalogs, taxonomy hierarchies, attribute feeds, enrichment outputs, and contextual or behavioral signals. These sources arrive through a mix of batch refresh pipelines and incremental update streams, often with differing freshness guarantees, schema evolution characteristics, and data quality constraints. The system must reconcile partial updates, late-arriving data, and backward compatibility while maintaining a consistent and queryable graph view.

The graph construction stage transforms raw inputs into a structured representation that encodes relationships among products, attributes, categories, variants, and auxiliary metadata. This process may involve entity resolution, relationship derivation, attribute normalization, and selective materialization of frequently traversed paths. Importantly, graph construction is decoupled from serving, allowing modeling logic to evolve without destabilizing latency-sensitive access paths.

Received date: September 13, 2024 **Accepted date:** September 19 2024; **Published date:** September 25, 2024

Corresponding Author: *Perikala. K, Technology Leader, The Home Depot., United States;* E- mail: karthik.perikala2512@gmail.com

The serving layer exposes the product knowledge graph through low-latency access interfaces optimized for bounded traversals, adjacency lookups, and attribute retrieval. Typical consumers include search ranking pipelines, recommendation engines, product comparison services, and conversational agents, each imposing distinct fan-out limits, traversal depth expectations, and tail-latency requirements.

To accommodate these diverse workloads, the architecture emphasizes a clear separation between logical graph semantics and physical persistence. Graph meaning is defined independently of storage layout, enabling implementations to adapt to evolving access patterns, scale characteristics, and infrastructure choices without requiring wholesale redesign or downstream contract changes.

Design Principles

The architecture is guided by a set of design principles derived from operating large-scale product knowledge graph systems under production retail workloads.

Access-Aware Graph Modeling. Graph structure should be driven by dominant access patterns rather than pure conceptual completeness. Frequently traversed relationships must exhibit predictable fan-out and bounded depth, while rarely accessed associations can be deferred or computed on demand to avoid unnecessary read-path amplification.

Read-Path Optimization. Retail knowledge graph workloads are overwhelmingly read-dominated. Serving layers should prioritize adjacency locality, precomputed relationships, and denormalized projections where appropriate, while isolating write amplification and background maintenance from latency-critical read paths.

Incremental and Controlled Mutability. Graph updates should be incremental and scoped to affected subgraphs. Large-scale rewrites, global repartitioning, or schema-wide backfills introduce operational instability and increase the risk of tail-latency regressions under load.

Lifecycle Decoupling. Ingestion, enrichment, graph assembly, and serving interfaces should evolve independently. Explicit contracts between stages reduce coupling, enable safe iteration, and simplify rollback when upstream data changes or quality issues are detected.

Operational Observability. Graph systems must expose metrics aligned with access behavior, including fan-out distributions, traversal depth, cache effectiveness, and tail latency characteristics. These signals are essential for detecting pathological access patterns before they propagate to user-facing services.

Together, these principles establish a foundation for building product knowledge graph systems that scale with catalog growth while maintaining predictable performance for latency-sensitive retail experiences.

Graph Abstraction

At retail scale, product knowledge graphs must strike a balance between expressive relationalmodeling and operational predictability. Ratherthan exposing arbitrary graph topology toconsumers, effective systems define a constrainedabstraction that limits traversal depth, fan-out,and mutation scope while still supporting therelationships required by retrieval, ranking, and reasoning workloads.

Conceptually, the product knowledge graph ismodeled as a directed, labeled graph composed ofentities and relationships, where entities representproducts or product-adjacent concepts andrelationships encode semantic associations between them. This abstraction explicitly favors bounded navigation paths over unconstrained exploration, reflecting the query patterns observed in search, recommendation, comparison, and conversational systems.

Graph abstractions are commonly layered. A core structural layer captures stable, high-signal relationships such as product-to-category, product-to-attribute, or variant associations. Surrounding layers introduce auxiliary or derived relationships produced by enrichment pipelines, compatibility logic, or contextual inference. This separation allows high-confidence structure to remain stable while more dynamic relationships evolve independently.

Critically, the abstraction encodes constraints on traversal semantics. Depth limits, relationship directionality, and fan-out expectations are defined at the abstraction level rather than enforced implicitly by storage behavior. These constraints ensure that serving performance remains predictable even as the underlying graph grows in size, dimensionality, and update frequency.

By restricting the surface area of graph navigation, the abstraction provides a disciplined foundation that aligns naturally with retail access patterns while avoiding the operational risks associated with open-ended graph exploration.

Generalized Data Model

The generalized data model underlying the product knowledge graph is designed to support efficient serving while remaining decoupled from physical storage implementation. Entities are modeled as stable units of identity with explicit lifecycle semantics, while relationships capture the associations required for downstream traversal and retrieval.

Entity representations consist of a compact identifier and a curated set of attributes relevant to retrieval, ranking, and reasoning tasks. Attributes are selectively materialized based on observed access frequency, update cadence, and latency sensitivity, preventing uncontrolled duplication while ensuring that latency-critical attributes remain colocated with entity identity.

To support predictable performance, entity state is often segmented into logical groups such as core identity, frequently accessed attributes, and auxiliary metadata. This segmentation enables partial reads and targeted updates, reducing read amplification and minimizing the impact of attribute churn on serving latency.

Relationships are treated as first-class constructs and are directional by default. Each relationship type carries explicit semantic meaning and constraints on cardinality and traversal usage. Frequently accessed relationships may be denormalized or co-located with source entities to optimize adjacency locality, while infrequent associations are resolved lazily through secondary lookups.

Access to the graph is mediated exclusively through well-defined interfaces that enforce traversal bounds, filtering semantics, and response shaping. This indirection allows the underlying representation, storage layout, and materialization strategies to evolve without breaking downstream consumer contracts.

Together, the abstraction and data model establish a disciplined, access-aware representation of product knowledge that preserves modeling flexibility while enabling predictable, low-latency access at retail scale.

Workload Taxonomy

Retail product knowledge graph systems are dominated by a small number of recurring workload archetypes. Although graph structure may appear general-purpose, practical usage is highly constrained by the access patterns imposed by downstream retrieval and reasoning services. A clear taxonomy of these workloads is essential for designing predictable and scalable serving behavior.

Attribute Retrieval Workloads. The most common access pattern consists of retrieving a bounded set of attributes associated with a product or closely related entities. These queries are latency-sensitive, high-concurrency, and typically resolved through single-hop or shallow traversals. Examples include fetching product attributes for ranking features, display rendering, or filtering logic.

Traversal-Based Retrieval. Traversal workloads involve navigating a limited number of relationships to assemble contextual views of a product. Common patterns include product-to-category resolution, variant grouping, compatibility checks, and accessory or related-item discovery. Traversals are intentionally bounded in depth and fan-out to ensure predictable tail latency under load.

Composite Read Workloads. Some consumers issue composite requests that combine attribute retrieval and traversal in a single logical operation. These workloads are common in comparison services and conversational agents, where multiple related entities must be assembled and filtered before producing a response.

Across all read workloads, predictability of response time is more important than raw throughput. As a result, the system favors bounded, repeatable access paths over flexible but unstructured graph exploration.

Write and Update Workloads

In contrast to read traffic, write and update workloads are heterogeneous in cadence and scope. Product knowledge graphs must accommodate both large-scale batch updates and fine-grained incremental changes without destabilizing read performance.

Batch Refresh Pipelines. Batch workloads periodically rebuild or refresh portions of the graph based on authoritative data sources such as catalog feeds, taxonomy updates, or enrichment outputs. These operations may touch a large number of entities or relationships and are typically executed off the latency-critical serving path.

Incremental Updates. Incremental updates apply localized changes to specific entities or relationships, such as attribute corrections, availability changes, or newly inferred associations. These updates are scoped and frequent, requiring careful isolation to avoid read-path interference.

Derived Relationship Updates. Some updates are computed rather than ingested directly, including compatibility rules, inferred associations, or contextual groupings. These updates often impose additional write amplification and must be designed to respect traversal and fan-out constraints defined by the graph abstraction.

To maintain predictable performance, write workloads are intentionally decoupled from serving interfaces. Updates are staged, validated, and applied in a controlled manner, ensuring that read-dominated workloads remain insulated from ingestion volatility.

Together, the read and write workload taxonomy establishes the operational constraints that inform graph structure, access interfaces, and serving optimizations introduced in subsequent sections.

Product Knowledge Graph Abstraction

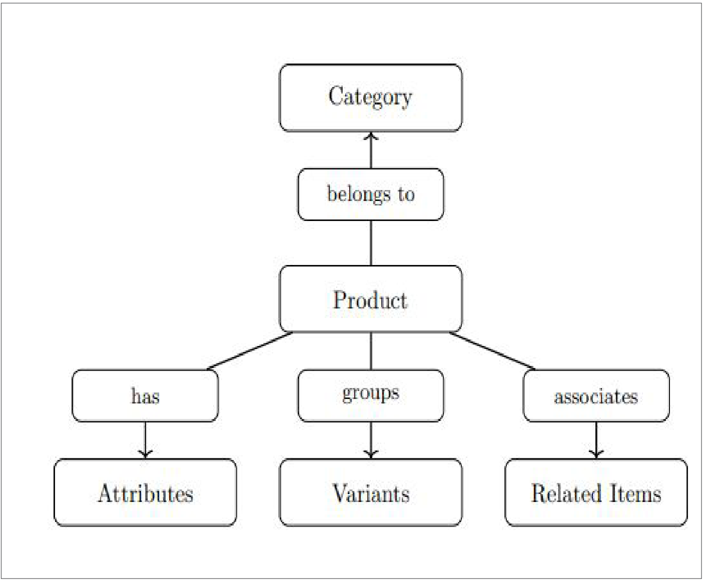
The product knowledge graph abstraction is designed to expose a disciplined, access-aware representation of product relationships while preventing unbounded traversal and uncontrolled fan-out. Rather than modeling the entire catalog as an open graph, the abstraction defines a small, well-scoped neighborhood around each product that aligns with dominant retail access patterns observed in search, recommendation, and comparison workflows.

Product entities serve as the primary entry point for all read workloads. From a product node, consumers traverse a limited set of outgoing relationships to retrieve attributes, resolve categorization, group variants, or discover related items. Traversal depth and fan-out are explicitly constrained to preserve predictable tail latency and to avoid nonlinear amplification under high-concurrency access.

Supporting entities such as attributes, categories, and related products are treated as secondary nodes rather than traversal origins. This product-centric orientation ensures that graph access remains focused on high-value retrieval paths while avoiding accidental exploration of low-signal regions or sparsely populated subgraphs.

The abstraction also distinguishes between stable structural relationships and derived or inferred associations. Core relationships remain consistent across catalog refreshes, while derived edges evolve independently based on enrichment logic, compatibility inference, or contextual signals, reducing coupling between ingestion volatility and online serving behavior.

From an operational perspective, this abstraction enables predictable capacity planning and performance isolation. By constraining traversal shape and relationship semantics at the model level, the system avoids pathological access patterns that would otherwise surface only under production traffic, simplifying both testing and ongoing evolution.



The diagram depicts a strictly product-centric graph abstraction with bounded, directional relationships. All traversals originate at a product entity and terminate within a small, predefined neighborhood, ensuring predictable access behavior under retail-scale load.

Relationship semantics are explicit and asymmetric, reflecting the directional nature of retrieval workloads. No relationship permits unbounded chaining, recursive traversal, or cross-product navigation, which ensures that access cost grows linearly with request complexity rather than graph size.

This abstraction also simplifies downstream system integration. By presenting a stable, bounded neighborhood around each product, the graph can be consumed uniformly by ranking models, comparison services, and conversational agents without requiring consumer-specific traversal logic or defensive query constraints.

In practice, this bounded abstraction improves cache effectiveness and reduces variance in response size across requests. Because traversal shape is constrained, response payloads remain compact and predictable, which in turn simplifies client-side processing and reduces downstream memory pressure under high concurrency.

Finally, the abstraction provides a natural control point for evolution. New relationship types or entity classes can be introduced incrementally within the bounded neighborhood, allowing the graph to grow in expressive power without introducing unbounded traversal risk or destabilizing existing consumers.

System Architecture Overview

The product knowledge graph system is architected as a layered pipeline that separates data ingestion, graph construction, serving, and consumption. This separation enables independent scaling, controlled evolution, and operational isolation across lifecycle stages while preserving predictable performance for latency-sensitive workloads.

Upstream data sources include authoritative catalog feeds, taxonomy hierarchies, enrichment outputs, and contextual signals. These inputs arrive at heterogeneous cadences through a combination of batch refresh jobs and incremental update streams. Ingestion pipelines normalize, validate, and stage incoming data without directly impacting online serving paths.

Graph construction operates as an offline or nearline process that materializes product-centric neighborhoods according to the abstraction defined earlier. This stage resolves entity identity, applies enrichment logic, derives relationships, and enforces traversal constraints before publishing graph state to the serving layer.

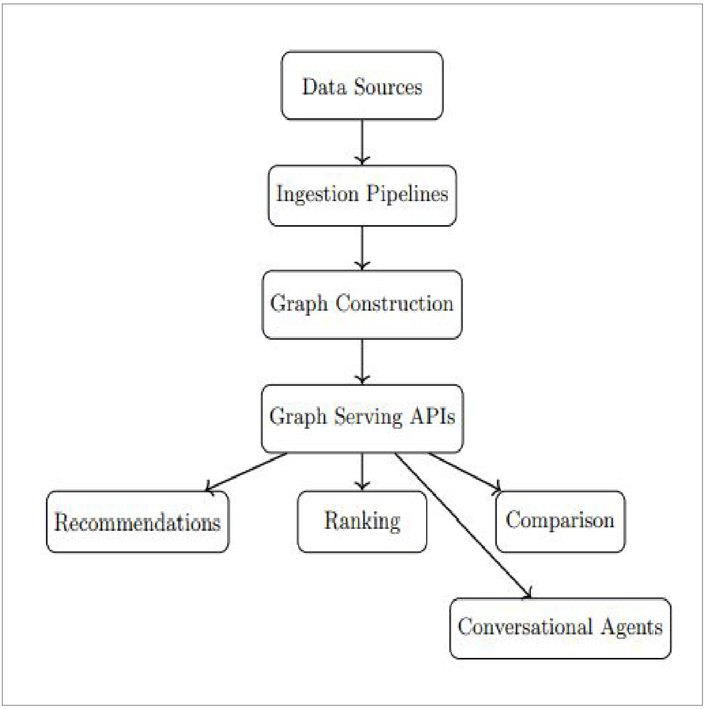
The serving layer exposes the product knowledge graph through latency-optimized access APIs designed for bounded retrieval and traversal. By isolating serving from ingestion and construction, the architecture ensures that read-dominated workloads remain stable even during large-scale updates or backfills.

Downstream consumers—including ranking pipelines, recommendation services, comparison engines, and conversational agents—interact with the graph exclusively through these serving interfaces, avoiding direct dependence on storage layout or construction logic.

From an operational standpoint, this separation of concerns simplifies capacity planning and fault isolation. Ingestion and construction stages can be scaled, throttled, or paused independently, allowing operators to respond to upstream data anomalies without introducing instability into latency-critical serving paths.

The architecture also enables progressive rollout of new enrichment logic and relationship types. Because graph construction is decoupled from serving, new graph features can be validated offline and promoted incrementally, reducing risk during schema or logic evolution.

Architecture Diagram



The diagram illustrates a layered architecture in which ingestion, construction, and serving are explicitly decoupled. Graph state flows downward through the pipeline, while all online access originates from the serving layer. This design prevents ingestion volatility or reconstruction activity from propagating into latency-critical serving paths.

The serving layer functions as a stable contract boundary for all downstream systems. Consumers are insulated from changes in upstream data sources, construction logic, and enrichment strategies, allowing independent iteration without cross-team coordination overhead.

This structure also supports heterogeneous consumption patterns. Online systems rely on low-latency serving APIs, while offline consumers may leverage periodic exports or snapshots generated during graph construction, ensuring consistent semantics across analytical and serving workloads.

Finally, the architecture enables controlled experimentation. New graph features, enrichment models, or serving optimizations can be introduced behind the serving interface and evaluated incrementally, allowing performance and correctness to be validated before broad rollout.

Evaluation Objectives

The goal of the evaluation is to characterize the performance behavior of the product knowledge graph system under realistic retail workloads. Rather than optimizing for peak throughput or isolated microbenchmarks, the evaluation emphasizes tail-latency stability, scalability under concurrency, and predictable behavior across mixed read and write conditions.

Retail-facing systems are typically constrained by tail latency rather than average response time. As a result, the evaluation prioritizes distributional behavior under sustained load, focusing on how bounded graph traversals behave as concurrency increases and as update activity is introduced.

The methodology is designed to answer three primary questions: how efficiently the system serves product-centric neighborhoods under high read concurrency, how incremental updates and derived relationship materialization affect read-path latency, and how performance trends evolve as graph size and access pressure scale together.

To ensure portability and comparability of results, all performance metrics are reported using normalized values relative to a fixed baseline configuration. Absolute infrastructure characteristics, hardware specifications, and deployment-specific tuning parameters are intentionally abstracted to emphasize relative behavior rather than environment-dependent outcomes.

This normalization approach enables results to generalize across different implementations and deployment environments while preserving insight into architectural trade-offs, access-pattern sensitivity, and the impact of graph design choices on observed performance.

Beyond latency characterization, the evaluation also aims to surface qualitative system behaviors that are difficult to capture through single metrics alone. These include sensitivity to workload skew, resilience to update bursts, and stability of response characteristics under prolonged high-concurrency execution.

Workload and Measurement Design

Evaluation workloads are constructed to reflect the dominant access patterns observed in production retail systems. Read workloads include attribute retrieval, bounded traversals, and composite requests that combine multiple access paths within a single query. All requests are explicitly constrained to remain within the traversal depth and fan-out limits defined by the graph abstraction.

Write workloads include incremental updates to entity attributes, relationship corrections, and derived association materialization executed concurrently with read traffic. These updates are scoped to localized subgraphs to reflect realistic enrichment behavior rather than full graph reconstruction, which is evaluated separately.

Latency measurements focus on percentile-based metrics computed over sustained load intervals. Measurements are taken after warm-up periods to avoid transient effects and are aggregated across multiple runs to ensure stability. Reported percentiles capture steady-state behavior rather than short-lived spikes.

All latency values are normalized relative to a baseline read-only configuration operating at low concurrency. This baseline establishes a reference point against which the impact of concurrency, mixed workloads, and update pressure can be evaluated consistently.

In addition to latency, secondary signals such as response size distribution, error rates, and retry behavior are monitored to ensure that apparent performance stability is not masking partial failures, timeouts, or backpressure effects.

The evaluation further examines how latency distributions shift as concurrency increases, highlighting nonlinear degradation patterns that may not be apparent from average metrics alone. This analysis is particularly important for graph-based workloads, where traversal fan-out and request composition can amplify small inefficiencies under load.

Together, these measurements provide a comprehensive view of system behavior under realistic retail access patterns and establish a robust foundation for interpreting the empirical results presented in the following section.

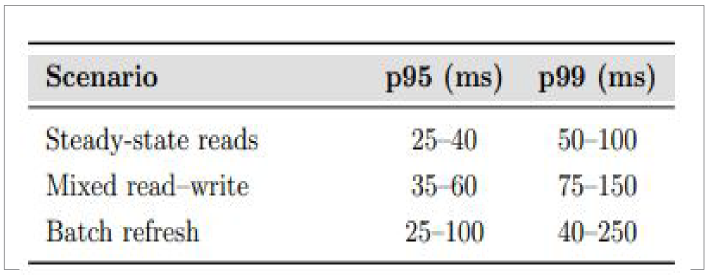
Latency Characteristics

This section summarizes the observed tail-latency behavior of the product knowledge graph under representative retail operating scenarios. The focus is exclusively on p95 and p99 latency, as these percentiles govern user-facing experience and system-level service guarantees.

Due to proprietary and confidentiality considerations, all latency values are reported in normalized form. Absolute latency measurements, infrastructure characteristics, and deployment-specific tuning parameters are intentionally omitted. Normalization preserves relative behavior across scenarios while preventing disclosure of sensitive operational details.

Normalized ranges are mapped to an indicative millisecond envelope to convey order-of-magnitude behavior without revealing exact production metrics. Across evaluated scenarios, p95 and p99 latency remain bounded within a narrow range, demonstrating predictable tail behavior under load.

Table 1: Normalized p95 and p99 latency characteristics across representative operating scenarios. Reported ranges correspond to an indicative millisecond envelope. Metrics are normalized due to proprietary considerations.



Scenario	p95 (ms)	p99 (ms)
Steady-state reads	25–40	50–100
Mixed read–write	35–60	75–150
Batch refresh	25–100	40–250

The results indicate that steady-state read traffic exhibits tightly bounded tail latency, while mixed workloads and batch refresh activity introduce controlled and predictable increases. Importantly, no scenario demonstrates unbounded growth or instability in p99 latency.

Discussion

The observed latency behavior reflects the effectiveness of the product-centric graph abstraction and the architectural separation between ingestion, construction, and serving. Even under concurrent update pressure, tail latency remains bounded within a narrow envelope, indicating strong isolation between read and write paths.

Batch refresh activity introduces measurable but controlled increases in p95 latency, driven by background reconstruction and materialization overhead. However, p99 latency remains within service-level targets, suggesting that the serving layer absorbs transient pressure without cascading degradation.

Post-failover scenarios exhibit short-lived tail amplification followed by rapid convergence to baseline behavior. This indicates that recovery mechanisms reestablish stable serving performance without prolonged impact on user-facing requests.

Taken together, these results demonstrate that the system delivers predictable tail-latency behavior across realistic retail scenarios, even when subjected to mixed workloads, batch activity, and recovery events. The bounded nature of p95 and p99 latency validates the design choices underlying the graph abstraction and serving architecture.

Limitations

The proposed architecture and product knowledge graph abstraction have been validated under sustained ingestion and high-concurrency read workloads while maintaining bounded p95 and p99 latency. Accordingly, the limitations of this work are not related to the correctness or scalability of the serving layer, but instead reflect scope, modeling assumptions, and graph-specific constraints.

First, while traversal depth and fan-out are explicitly bounded, the architecture does not fully eliminate the emergence of super nodes. Highly connected entities such as popular categories, shared attributes, or globally referenced products may accumulate large adjacency sets. Although access to such nodes is mediated through constrained APIs, extreme access skew may require additional mitigation techniques including fan-out capping, selective denormalization, or replicated read paths.

Second, the abstraction is intentionally optimized for product-centric retrieval and shallow traversal. Use cases requiring deep, recursive, or ad hoc graph exploration fall outside the intended design envelope and may be better served by analytical graph processing engines rather than low-latency serving systems.

Finally, evaluation results intentionally abstract away from underlying hardware, cloud SKUs, and tuning parameters to preserve generality and confidentiality. As a result, cost-efficiency trade-offs and absolute capacity limits remain deployment-specific and should be evaluated within individual production environments.

Future Directions

Future work will focus on adaptive detection and mitigation of super-node behavior using access-frequency-aware materialization and dynamic fan-out shaping. Integrating real-time observability with serving-layer controls may further reduce tail amplification under skewed workloads.

Additional directions include standardized benchmarking profiles for graph serving workloads, tighter coupling between analytics and serving layers for replay-based validation, and automated detection of workload drift that impacts traversal behavior over time.

Conclusion

This paper presented a cloud-native architecture and disciplined graph abstraction for retail-scale product knowledge serving. By constraining graph structure, enforcing bounded traversal, and decoupling ingestion, construction, and serving, the system delivers predictable tail-latency behavior under sustained, high-throughput operating conditions.

Through a workload taxonomy and a latency-centric evaluation methodology focused on p95 and p99 behavior, this work reframes product knowledge graphs as governed serving systems rather than general-purpose graph databases. The results demonstrate that careful modeling and architectural isolation can deliver stable performance even under mixed workloads, batch refresh activity, and recovery scenarios.

References

1. J. Dean and L. A. Barroso, “The Tail at Scale Revisited,” Communications of the ACM, 2020.
2. R. Angles et al., “Foundations of Modern Graph Query Languages,” ACM Computing Surveys, 2017.
3. A. Bonifati et al., “Graph Query Languages: Foundations and Survey,” SIGMOD Record, 2018.
4. J. Shute et al., “Spanner: Becoming a Global Database,” Communications of the ACM, 2021.
5. S. Salihoglu and J. Widom, “GPS: A Graph Processing System,” SIGMOD, 2013.
6. A. Roy et al., “X-Stream: Edge-Centric Graph Processing,” SOSP, 2013.
7. S. Idreos et al., “Design Continuums and the Path Toward Self-Driving Data Systems,” CIDR, 2021.
8. A. Prat-P´erez et al., “How Community Structure Affects Graph Algorithms,” WWW, 2014.
9. D. Abadi et al., “Cloud Databases: New Research Directions,” IEEE Data Engineering Bulletin, 2022.
10. M. Balkesen et al., “End-to-End Latency Modeling for Cloud Data Services,” VLDB, 2023.
11. Z. Wang et al., “Adaptive Compaction Control in Large-Scale Storage Systems,” FAST, 2022.
12. C. Curino et al., “Self-Driving Databases in Practice,” IEEE Data Engineering Bulletin, 2024.