LLM and MCP driven e-commerce assistant over graph databases

Andrii Baran, Iryna Spivak and Svitlana Krepych

West Ukrainian National University, 11 Lvivska Str., Ternopil, 46009, Ukraine

Abstract

Traditional e-commerce search-and-filter interfaces often lead to high cart abandonment rates due to rigid keyword matching and complex navigation. This paper presents an architecture for conversational e-commerce that integrates large language models (LLMs), Model Context Protocol (MCP) servers, and a Neo4j graph database to enable natural language product discovery. The system employs an LLM-powered assistant that securely executes Cypher queries over a graph-structured product catalog, with MCP servers enforcing validation to prevent direct database exposure. A near-real-time synchronization pipeline maintains consistency between the Neo4j knowledge graph and the transactional PostgreSQL database. The architecture supports multi-turn conversations with persistent context, enabling personalized recommendations grounded in explicit product relationships. By combining secure orchestration, graph-based reasoning, and conversational context, the proposed system reduces search effort while maintaining transparency and explainability.

Keywords

conversational e-commerce, Neo4j graph database, large language model, Model Context Protocol, software architecture, quality

1. Introduction

E-commerce has experienced substantial growth in recent years, with online catalogs now containing hundreds of thousands or even millions of products. However, the typical buyer journey has remained largely unchanged: search, filter, browse, compare, add to cart. This multi-step process is illustrated in figure 1, where the customer begins by browsing the catalog, refines results using search or filters, reviews product details, and only then proceeds to checkout. As the diagram shows, optional detours, such as comparing specifications or checking reviews, frequently add to the complexity of the journey. Longer and more complex interactions increase user frustration and abandonment rates. Studies estimate that the so-called "search abandonment", when a consumer gives up after unsuccessful product discovery, costs retailers over $2 trillion annually worldwide, including $234 billion in the US market alone [1]. More broadly, research shows that 70.22% of online shopping carts are abandoned before purchase, often due to poor site search and usability issues [2]. The surveys also reveal that 80% of consumers will not return after a poor customer experience, underscoring the stakes for retailers [3].

The root cause lies in a persistent mismatch between user intent and the limitations of current UIs and tools. Traditional site search engines remain largely keyword-driven, requiring exact matches to product titles or attributes. Such systems fail to accommodate natural human queries involving synonyms, misspellings, or multiple criteria. For instance, a request for a "long floral dress with short sleeves and comfortable fit" is often misinterpreted or ignored by conventional search, forcing the shopper into trial-and-error reformulation. Faceted filters, while intended to help, frequently demand that users adopt the catalog's own taxonomy, which can be confusing or opaque. In one usability

CS&SE@SW 2025: 8th Workshop for Young Scientists in Computer Science & Software Engineering, November 21, 2025, Kryvyi Rih, Ukraine
https://doi.org/10.55056/cssesw2025/paper27
baran.andrii13@gmail.com (A. Baran); i.spivak@wunu.edu.ua (I. Spivak); s.krepych@wunu.edu.ua (S. Krepych)
https://www.wunu.edu.ua/educational-subdivisions/faculty/fkit/department-kn-fkit/staff-kn-fkit/7027-spivak-iryna-yaroslavivna.html (I. Spivak); https://www.wunu.edu.ua/educational-subdivisions/faculty/fkit/department-kn-fkit/staff-kn-fkit/8416-krepych-svitlana-yaroslavivna.html (S. Krepych)
0009-0005-8266-7814 (A. Baran); 0000-0003-4831-0780 (I. Spivak); 0000-0001-7700-8367 (S. Krepych)

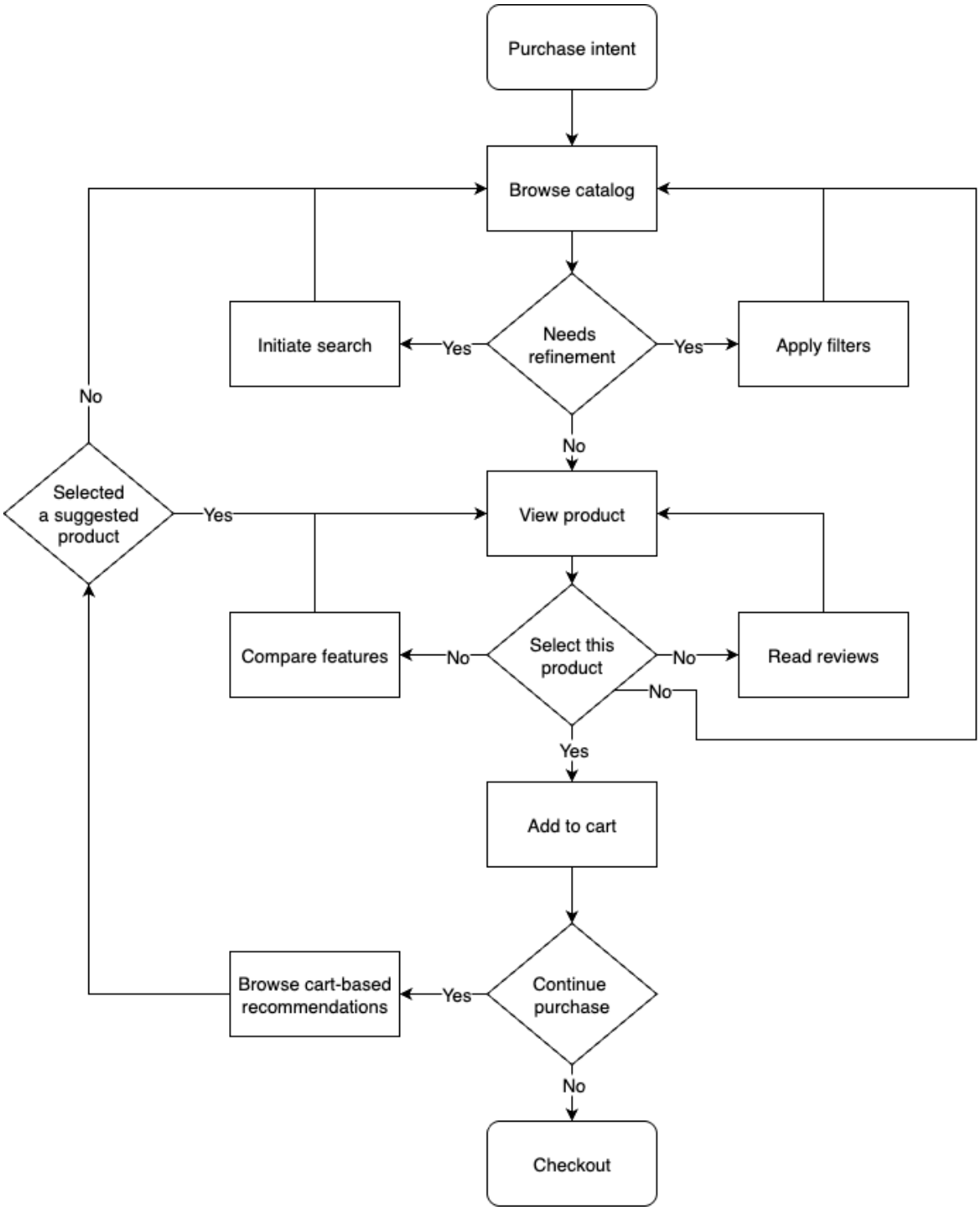

Figure 1: Typical e-commerce interaction flow.

study by the Baymard Institute, researchers observed that many e-commerce sites employ unclear or overly technical filter labels, causing users to overlook relevant options because the terminology does not match their expectations. For example, 62% of tested sites used filters with unexplained terms, significantly reducing the likelihood that users would apply the correct filter [4]. Baymard's comprehensive 2024 study on product finding further reports over 1,000 serious UX issues related to on-site search and filtering, emphasizing that such usability flaws remain widespread in contemporary e-commerce platforms [5]. Mobile devices further exacerbate these issues: smaller screens display

fewer products at once and require more navigation through nested filters, significantly increasing cognitive load. Although recommendation widgets such as "Customers also bought…" attempt to address discovery gaps, they are typically opaque systems that lack transparency and fail to consider the shopper's immediate context, such as budget or urgency. As a result, recommendations may feel irrelevant, contributing to dissatisfaction and lost sales opportunities.

Conversational e-commerce addresses these limitations by allowing users to interact in natural language. Instead of forcing users to fit into predefined catalog structures, it allows them to talk to a virtual assistant, similar to communicating with a store employee. Thanks to recent progress in LLMs, this approach can now be applied at scale, providing continuous and personalized support for shoppers. In 2023, 36% of consumers said they had made a purchase through a messaging app, a 227% increase since 2021 [6]. The global conversational commerce market is also growing fast – from about USD 11.26 billion in 2025 to USD 20.28 billion by 2030, with an expected annual growth rate of 12.47% [7]. These trends indicate growing adoption of conversational interfaces in retail.

While LLMs provide the linguistic and contextual capabilities that make conversational commerce feel natural, their direct interaction with enterprise data has historically been constrained. These models excel at interpreting synonyms, colloquial phrasing, and maintaining context across multiple turns, which allows them to narrow options through dialogue rather than forcing trial-and-error searches. However, enabling LLMs to securely query business systems is non-trivial: generating correct database queries requires strict schema awareness, and exposing raw access carries risks of malformed queries, data leakage, or manipulation. Consequently, early deployments often limited LLMs to shallow keyword retrieval or static product indexes, restricting their effectiveness in high-stakes commercial environments [8, 9].

The emergence of MCP servers provides a structured way to connect LLMs with enterprise systems. MCP servers act as controlled intermediaries that expose a predefined set of safe operations – such as parameterized graph queries, catalog lookups – that an LLM can invoke without direct database access [10, 11]. When combined with a graph database like Neo4j, this architecture enables the conversational agent to ground its responses in a live product knowledge graph reflecting the latest catalog, categories, and user interactions. Neo4j efficiently supports multi-hop queries and explicit relationship traversal, while the MCP layer enforces validation and guardrails, maintaining data security and consistency. Case studies show that retailers using Neo4j can generate more accurate recommendations, detect product affinities, and react to inventory changes in real time [12]. This combination provides a foundation for conversational e-commerce: LLMs interpret user intent, MCP servers orchestrate safe execution, and Neo4j provides structured product data.

In this work, the integration of LLMs into an existing e-commerce architecture is examined through the use of MCP servers and a Neo4j graph database. The proposed approach extends a Vendure-based commerce platform by synchronizing the product catalog with Neo4j to support reasoning and recommendation tasks, while MCP servers mediate query execution and enforce access control. This work presents a reference architecture that links natural language interfaces with structured data sources, emphasizing security, explainability, and operational reliability in real-world deployments.

2. Related work

Conversational e-commerce has been actively studied for over a decade. Early systems relied on rule-based and scripted bots to guide product discovery, but were limited by weak language understanding and tight coupling to specific backend systems [13]. With the advent of LLMs, recent research demonstrates significant improvements in handling natural dialogue, maintaining context across multiple turns, and adapting recommendations in real time.

Recent work on Text-to-SQL systems shows that LLMs can generate structured queries from natural language when provided with schema information [8], though enterprise deployments face challenges related to hallucination, security, and auditability [9]. Standards such as the MCP have emerged to address these concerns through secure orchestration layers [14], with early production deployments

demonstrating viability for AI-driven applications [10, 11].

Parallel research has explored graph databases for representing product catalogs, relationships, and user behavior in e-commerce. Prior studies confirm their suitability for capturing structures like variants, bundles, and co-purchase patterns, and for powering recommendation systems [15]. Production retail deployments demonstrate improved recommendation accuracy and query performance for multi-hop relationship traversal [12].

This work extends a Vendure-based headless commerce platform developed in the author's prior work. The original system provided transactional capabilities and GraphQL APIs; this research augments it with Neo4j for knowledge representation and MCP servers for secure LLM orchestration. The resulting architecture demonstrates how conversational interfaces can be integrated into existing e-commerce infrastructures while maintaining security, explainability, and production readiness.

3. Proposed approach

3.1. General overview

The proposed approach restructures the e-commerce journey by shifting operational complexity away from the user and into a conversational assistant. In the traditional model (figure 1), the customer must operate the system unaided, relying solely on manual interaction – a design that amplifies cognitive load and decision fatigue. The alternative model (figure 2) introduces a dialogue-driven process in which all actions take place within a single conversational interface.

The conversational paradigm offers several operational advantages over traditional interfaces. Users express intent through natural language rather than learning catalog taxonomies or navigating filter hierarchies. The system maintains context across dialogue turns, allowing references to previous recommendations without re-stating requirements – particularly valuable on mobile devices where screen space is limited.

Instead of navigating menus, users engage in natural dialogue with the assistant. They can request recommendations, compare features, reference previous results, and refine criteria through follow-up questions. The assistant interprets each request and executes catalog searches through secure MCP functions, retrieving product data from a graph database. This unified flow simplifies the shopping experience while maintaining transparency (figure 2).

3.2. Architecture overview

The system is built on top of Vendure, a modern headless commerce framework. Vendure handles the core transactional logic and exposes a GraphQL API connected to a PostgreSQL database. This foundation covers essential e-commerce functionality such as product management, checkout, and customer data. To extend the system with AI-driven interactions, a dedicated assistant component was added on top of this architecture (figure 3).

The AI assistant is integrated via a Vendure plugin, which extends the existing API with additional GraphQL queries and optional UI components. The plugin itself remains minimal and delegates all inference tasks to an external AI microservice. This separation ensures that Vendure continues to focus on commerce logic and access control, while the assistant evolves independently.

The AI microservice functions as the orchestration layer for all conversational interactions. Internally, it leverages a LLM, which interprets user intent and generates follow-up actions. These actions are not directly executed against the commerce backend. Instead, they pass through a layer of MCP servers embedded within the assistant architecture, which act as secure mediators between the LLM and the data. The MCP layer enforces constraints and ensures safe execution of system-level requests.

For search, retrieval, and reasoning, the assistant uses a separate knowledge database tailored for AI workloads. This Neo4j-based store contains structured and semantically indexed representations of products, categories, attributes, and interactions. A dedicated sync service maintains consistency

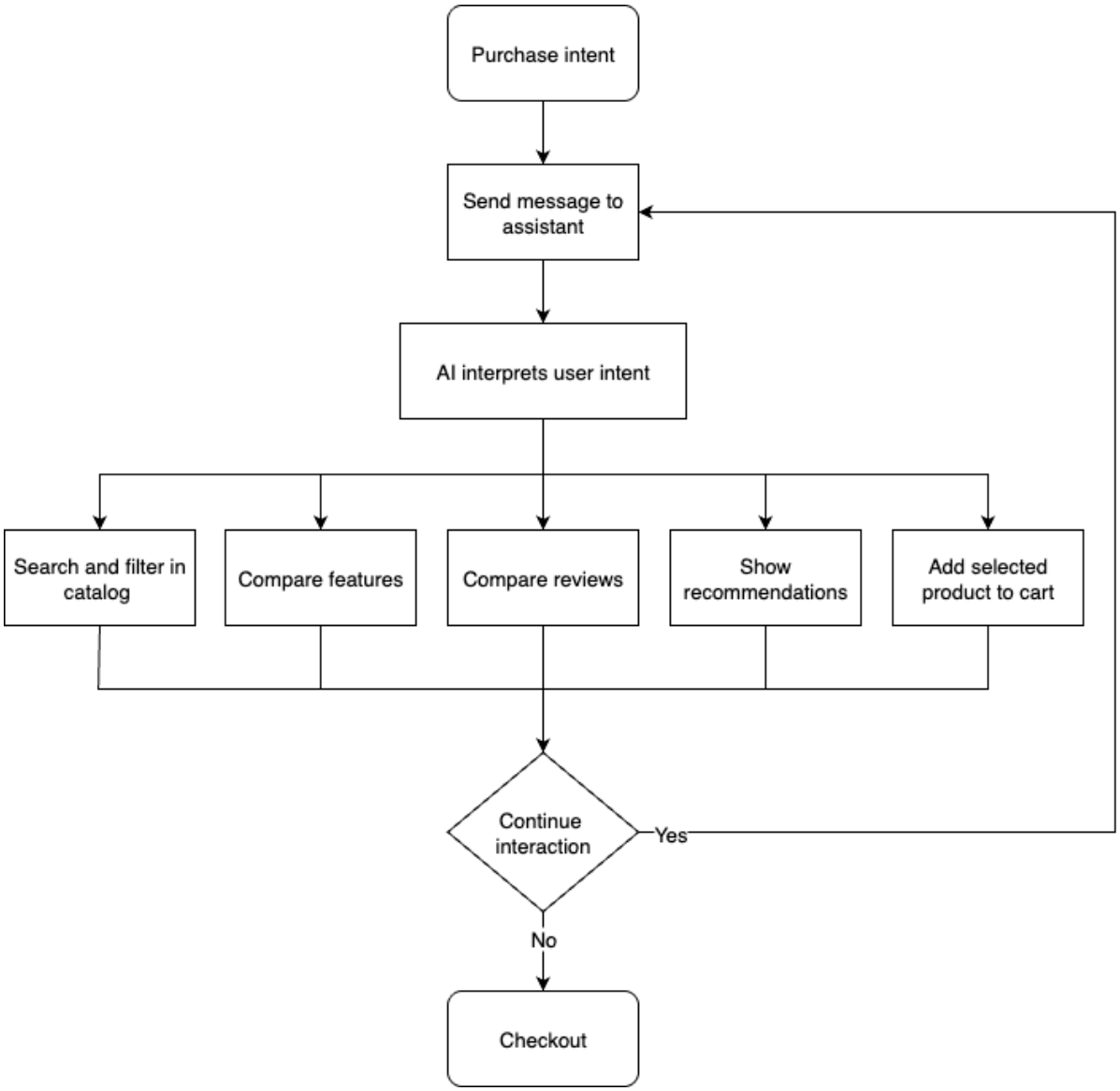

Figure 2: Conversational e-commerce interaction flow.

between the transactional PostgreSQL database and the knowledge graph, ensuring that product updates are reflected in the AI's reasoning layer.

The assistant receives a prompt, interprets the intent, optionally queries the knowledge database through validated MCP calls, and returns a response in a defined format referencing existing entities in the system. The storefront then fetches product data from Vendure using standard APIs. This ensures consistency and maintains security boundaries. The modular architecture also supports independent scaling, fault isolation, and gradual rollout of AI features without affecting core platform stability.

3.3. Knowledge database

3.3.1. Argumentation of graph database usage

LLM-based assistants require efficient traversal of product relationships to answer semantic queries like *"show black running shoes similar to Nike under $120, often bought with water bottles"*. Graph databases model data as nodes and edges, directly mirroring how entities relate in the real world – products connect to categories, variants link to base items, and co-purchase patterns form explicit edges. This structure facilitates translation from conversational queries to executable graph patterns [12].

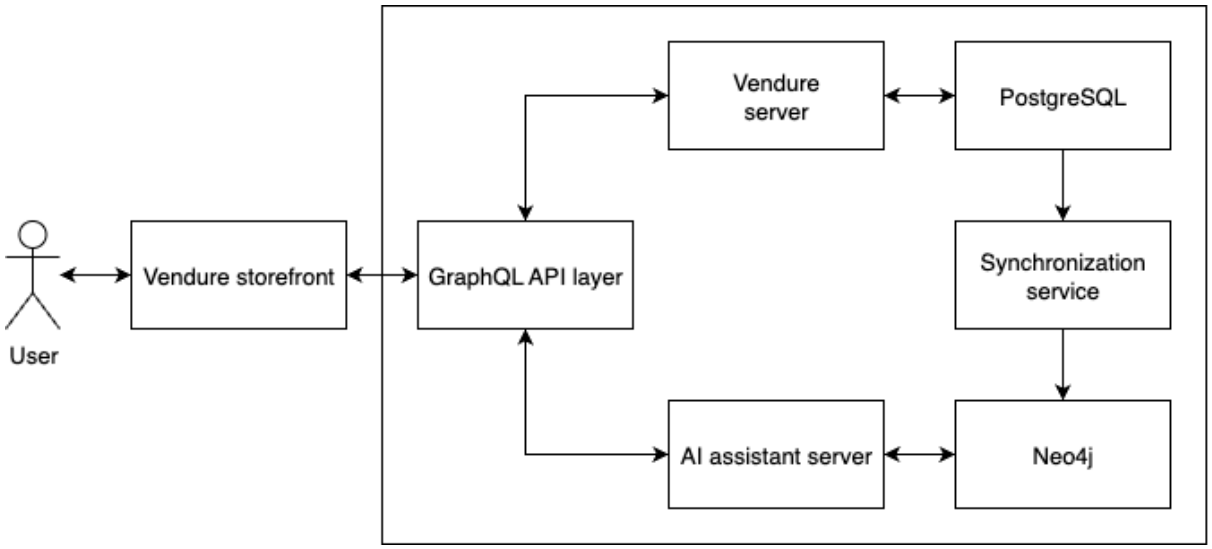

Figure 3: Architecture overview.

Graph-based modeling approaches have demonstrated effectiveness across diverse application domains. Research has shown their utility in representing complex structural dependencies in software systems [16], optimizing data processing pipelines through hybrid architectures [17], and enabling advanced quality assessment methodologies [18]. These applications underscore the versatility of graph representations for capturing multi-hop relationships and facilitating pattern-based reasoning across different problem domains.

By contrast, SQL requires nested JOINs across products, attributes, and purchase histories, with query complexity increasing significantly for multi-hop relationships, while document stores provide limited support for cross-document joins. Graph traversal operates via index-free adjacency, accessing only directly connected nodes regardless of graph size [15]. Table 1 provides a comparison of database paradigms.

Table 1
Comparison of database paradigms for LLM-driven queries.

Aspect	Graph	Relational	Document
Schema flexibility	Add nodes/edges freely	Migrations required	Moderate (JSON)
Relationship storage	First-class edges	Foreign keys + JOINs	Manual references
Multi-hop queries	Index-free adjacency	Multiple JOINs	Multiple lookups
Query expressiveness	Pattern matching	Complex SQL	Pipeline scripts
LLM integration	Direct pattern translation	Transactional focus	Single-entity queries

Neo4j was selected for its 44% graph DBMS market share, deployment by 84% of Fortune 100 companies, and production-ready ecosystem [19]. Cypher's pattern-matching syntax – e.g., `MATCH (product)-[:SIMILAR_TO]->(alt)-[:BOUGHT_WITH]->(accessory)` – maps intuitively to conversational queries, making it straightforward for LLMs to construct semantically correct graph traversals.

ACID compliance ensures concurrent safety, while the Graph Data Science library provides built-in collaborative filtering and similarity algorithms [15]. The synchronization service replicates product metadata and interaction patterns from PostgreSQL into Neo4j, where MCP-validated queries execute semantic traversals without expensive JOIN operations. Explicit relationship paths enable explainable recommendations while isolating AI workloads from transactional systems.

3.3.2. Graph schema design

The knowledge graph is built from a minimal subset of Vendure entities sufficient for conversational product discovery, augmented with user interaction data. Figure 4 illustrates a representative subgraph centered on a single product – not the entire database, but rather the contextual neighborhood that the LLM traverses during a typical product discovery session.

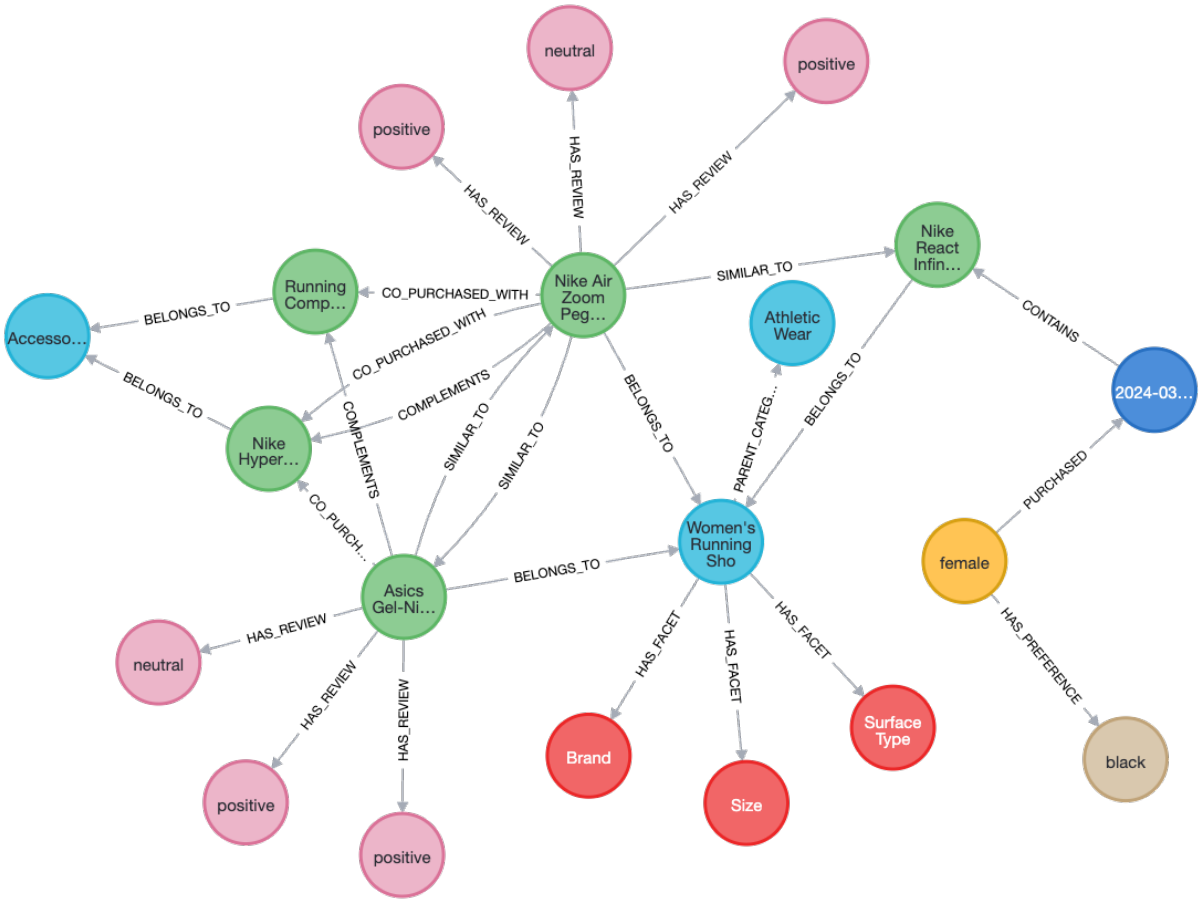

Figure 4: Example product subgraph.

Core node types. The schema defines seven fundamental entity types that capture e-commerce semantics and user context:

`Product` Catalog items representing purchasable products.

Properties: `id`, `name`, `brand`, `price`, `size`, `color`, `suitableFor` (array of usage contexts), `avgRating`, `reviewCount`, `category` (denormalized for efficient filtering), `inStock`.

`Category` Hierarchical taxonomy nodes with properties `name` and `slug`.

Organized through PARENT_CATEGORY relationships to enable upward generalization ("Women's Running Shoes" → "Athletic Wear"). Products connect via BELONGS_TO edges.

`Facet` Attribute descriptors that define filterable dimensions within categories.

Properties: `name` (e.g., "Surface Type"), `options` (array of possible values), `priority` (for question ordering), `discriminative` (whether this facet narrows product selection). Categories connect to facets via HAS_FACET edges to enable dynamic clarification questions.

User Customer profiles with properties `id` and `gender`.

Connected to preference nodes and order history for context-aware personalization.

Preference User-specific constraints stored as key-value pairs (e.g., `{key: 'color', value: 'black'}`).

Connected to users via HAS_PREFERENCE edges to enable automatic filtering without explicit user input during each session.

Order Purchase records with properties `id`, `date`, and `total`.

Connected via PURCHASED (User→Order) and CONTAINS (Order→Product) relationships to enable purchase history retrieval and preference inference.

Review Product feedback with properties `sentiment` ('positive', 'neutral', 'negative'), `rating` (1.0–5.0), `text`, `author`, `date`.

Connected to products via HAS_REVIEW edges for comparative analysis.

Derived relationships. Beyond direct schema mappings, three relationship types enhance semantic reasoning by materializing patterns from historical sales data:

CO_PURCHASED_WITH Connects products frequently appearing in the same orders, weighted by `frequency` property representing co-occurrence count.

Supports recommendations like "customers who bought X also bought Y" without runtime aggregation across order history.

SIMILAR_TO Links products sharing significant attribute overlap within the same category, weighted by similarity `score` (0.0–1.0).

Enables "find alternatives" queries without runtime similarity computation. Similarity is computed periodically based on shared facets and category membership.

COMPLEMENTS Identifies cross-category pairings from purchase pattern analysis, such as running shoes with compression socks.

Enables upsell recommendations by identifying products that complete a purchase rather than replace it.

These materialized relationships trade storage for query performance. Pre-computing and storing these relationships as explicit edges eliminates the need for expensive runtime joins or similarity calculations during conversational interactions.

3.4. Data synchronization pipeline

The effectiveness of AI-driven recommendations depends critically on data freshness. Newly launched items must appear in recommendations immediately, out-of-stock products should not be suggested, and trending patterns need to be detected in realtime. To ensure that the knowledge graph reflects the live state of the transactional database, the system implements a near-realtime change data capture (CDC) pipeline that propagates every PostgreSQL transaction to Neo4j within seconds.

The architecture (figure 5) consists of three components working in sequence:

Debezium connects to PostgreSQL's write-ahead log (WAL) and captures inserts, updates, and deletes in exact commit order without imposing load on the application layer [20, 21]. On startup, it performs a consistent snapshot of tables, then switches to streaming mode. Each event includes the full row state before and after the change, plus a logical sequence number (LSN) for ordering.

Apache Kafka serves as the durable event backbone [22]. CDC events are partitioned by entity type (`ecommerce.products`, `ecommerce.orders`) and replicated across brokers with configurable

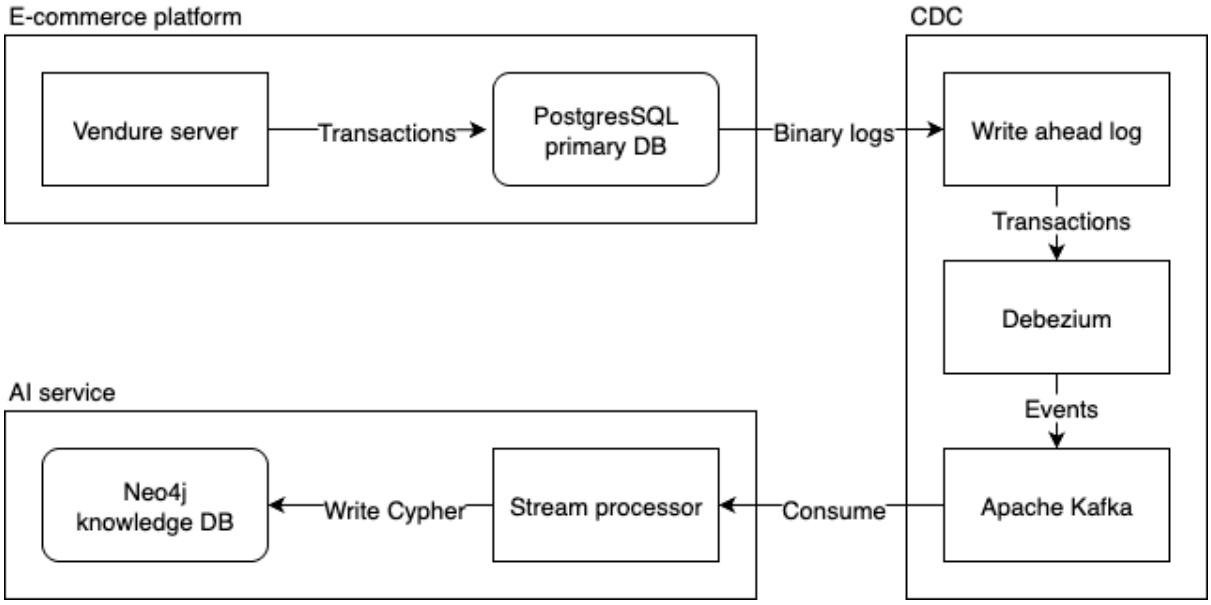

Figure 5: Data synchronization pipeline.

retention (e.g., 7 days). This allows reprocessing of historical changes and ensures zero data loss even when the stream processor is temporarily offline.

Stream processor – a service that subscribes to Kafka topics and transforms events into Neo4j operations:

- INSERT: Creates nodes (`(:Product)`, `(:Order)`) and relationships to categories by matching foreign keys.
- UPDATE: Applies changes using Cypher MERGE for idempotency, ensuring replayed events do not create duplicates.
- DELETE: Marks nodes with `deleted: true` to preserve historical context.

This pipeline maintains sub-second synchronization latency under normal load [22], ensuring the AI assistant operates on fresh data for accurate recommendations.

3.5. AI assistant

The AI assistant is deployed as an independent microservice, decoupled from the core e-commerce stack for modularity and fault isolation. The service exposes a minimal API that accepts three fields:

- `message` – the user's natural-language utterance
- `chatId` – the conversation/session identifier for multi-turn continuity
- `userId` – the persistent customer identifier for personalization

These parameters support multi-turn conversations with personalization based on user history stored in the knowledge graph.

3.5.1. Components and workflow

The AI assistant operates as a coordinated system of four specialized components:

- **AI Service Layer** – API gateway that receives client requests, validates input parameters, orchestrates communication between the LLM and MCP servers, enforces rate limiting per user/session, and validates generated responses by cross-referencing product IDs against Neo4j query results to prevent hallucinations.

- **LLM** – Claude 3.5 Sonnet operating in tool-augmented mode with 200K token context window. Interprets user queries and generates responses by invoking MCP tools via JSON-RPC 2.0 protocol to retrieve context, query the knowledge graph, and maintain state across multi-turn conversations.
- **MCP Memory Server** – Neo4j MCP server that provides persistent memory capabilities through graph database integration. Maintains conversation history, user preferences, and constraints as interconnected graph nodes across multiple sessions.
- **MCP Cypher Server** – Neo4j MCP server that enables Text2Cypher workflows. Exposes tools: `read_neo4j_schema` (returns node labels, relationship types, and property keys for query generation), `read_neo4j_cypher` (executes parameterized read queries). Validates all queries for injection patterns, enforces result size limits, and truncates responses to prevent token overflow. Operates in read-only mode by default to prevent unintended database modifications.

Figure 6 illustrates the complete request-response cycle across six coordinated phases.

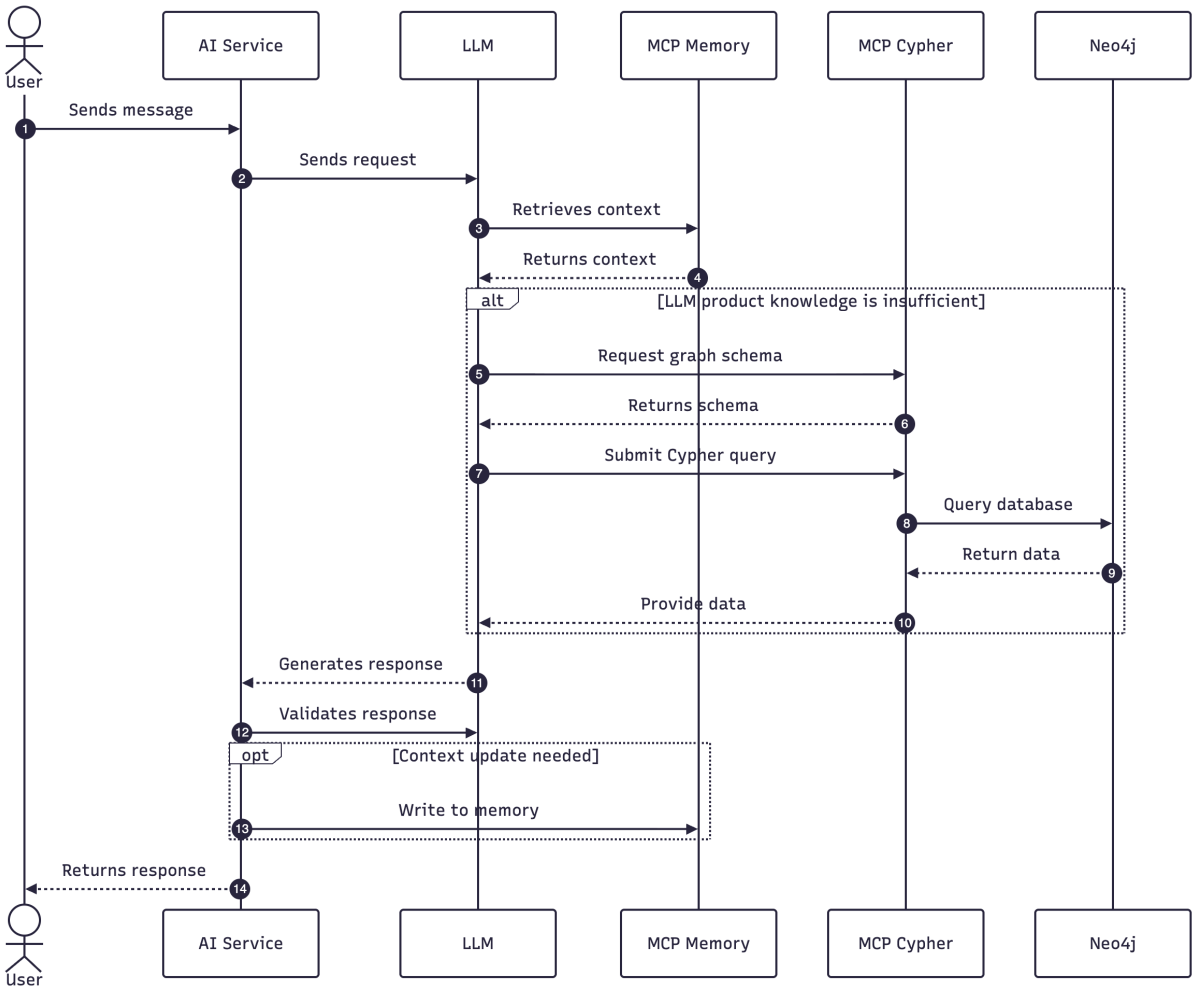

Figure 6: Request-response flow showing interaction between user, AI service, LLM, and MCP servers.

Context assembly (Steps 1–4). The AI Service forwards user messages to the LLM, which retrieves the conversation subgraph from MCP Memory Server. This returns dialogue history, extracted constraints, and user preferences, ensuring responses incorporate both current intent and accumulated knowledge across sessions.

Schema discovery (Steps 5–6). When product information is needed, the LLM requests the current graph schema from MCP Cypher Server. Schema introspection allows the LLM to adapt to catalog changes – new categories, attributes, or relationships become queryable without system updates.

Knowledge graph query (Steps 7–10). The LLM constructs a parameterized Cypher query combining constraints from the current message and conversation history. After validation, the MCP Cypher Server executes it against Neo4j and returns structured product data for synthesis.

Response generation (Steps 11–12). Query results are transformed into natural language with embedded URI schemes (`product:<id>`, `category:<slug>`, `compare:<id1>,<id2>`). To prevent hallucinations, the AI Service validates each response using a secondary evaluation prompt. The LLM assesses whether the response corresponds to the user's intent and remains within allowed data scope. If the evaluation fails – due to factual inconsistency or irrelevant disclosure – the response is discarded, and the system returns a generic fallback message.

Context update (Step 13). When the conversation reveals persistent facts – constraints, preferences, or feedback – the LLM invokes MCP Memory Server to store these as graph nodes and relationships, enabling future sessions to apply learned preferences.

Client rendering (Step 14). The enriched Markdown response is parsed on the client, converting URI schemes into interactive elements – product cards, navigation links, comparison buttons – while maintaining conversational coherence.

The MCP layer provides several advantages over direct LLM-to-database integration. Validation layers enforce read-only semantics, parameter binding, and resource limits, preventing unauthorized access or arbitrary command execution. Schema introspection enables automatic adaptation to catalog changes without prompt updates or retraining. Complete audit trails are maintained through logged tool invocations with parameters and results. Finally, components evolve independently – LLM upgrades, MCP tool additions, and schema expansions require no coordinated changes across the stack.

4. Demonstration

To illustrate the system's capabilities, there is a step-by-step walkthrough of a typical product discovery session. When a user expresses interest in purchasing running shoes (figure 7), the assistant proactively requests clarification on running surface, foot size, and brand preferences. After receiving specifications – size 42, road running, black color, no brand preference – the assistant queries the knowledge graph and returns two personalized recommendations (figure 8).

When the user requests a comparison, the assistant synthesizes review data and suggests the Nike model based on road running requirements. Product references render as clickable cards, and the comparison request triggers structured attribute analysis.

Step 1: Context retrieval. Before processing the first message, the assistant retrieves historical context from Neo4j. The LLM generates the following Cypher query:

```
MATCH (u:User {id: $userId})
OPTIONAL MATCH (u)-[:HAS_PREFERENCE]->(p:Preference)
OPTIONAL MATCH (u)-[:PURCHASED]->(o:Order)-[:CONTAINS]->(prod:Product)
RETURN u.gender, collect(p.key + ':' + p.value) as prefs,
       collect(DISTINCT prod.category) as history
```

This returns `{gender: "female", prefs: ["color:black"]}`, enabling automatic filtering without explicit user input.

Step 2: Dynamic clarification. The assistant queries Neo4j to identify discriminative facets for running shoes. LLM-generated Cypher:

```
MATCH (c:Category {name: "Women's Running Shoes"})-[:HAS_FACET]->(f:Facet)
WHERE f.discriminative = true
RETURN f.name, f.options, f.priority ORDER BY f.priority
```

Using this schema, it generates tailored questions about surface type, size, and brand preferences.

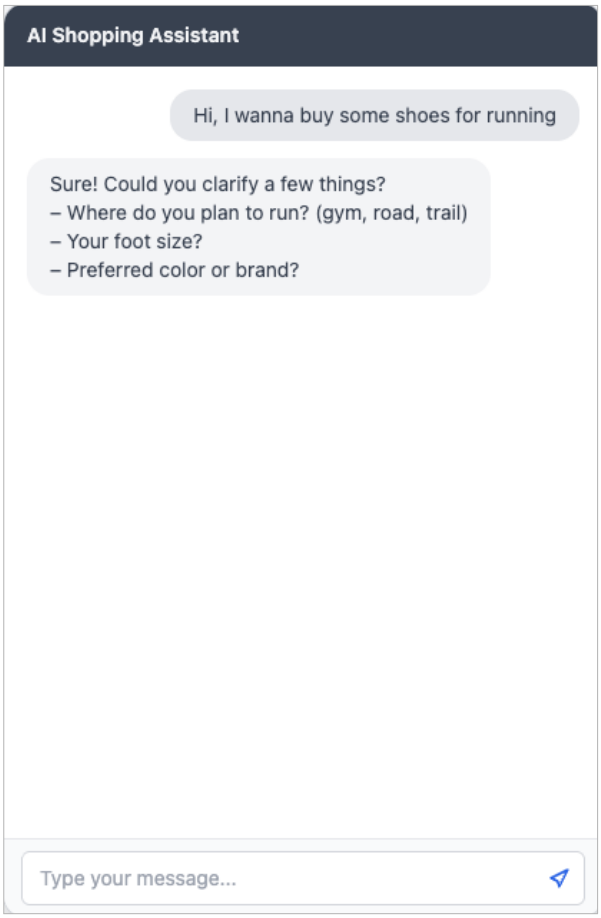

Figure 7: Assistant initiates clarification.

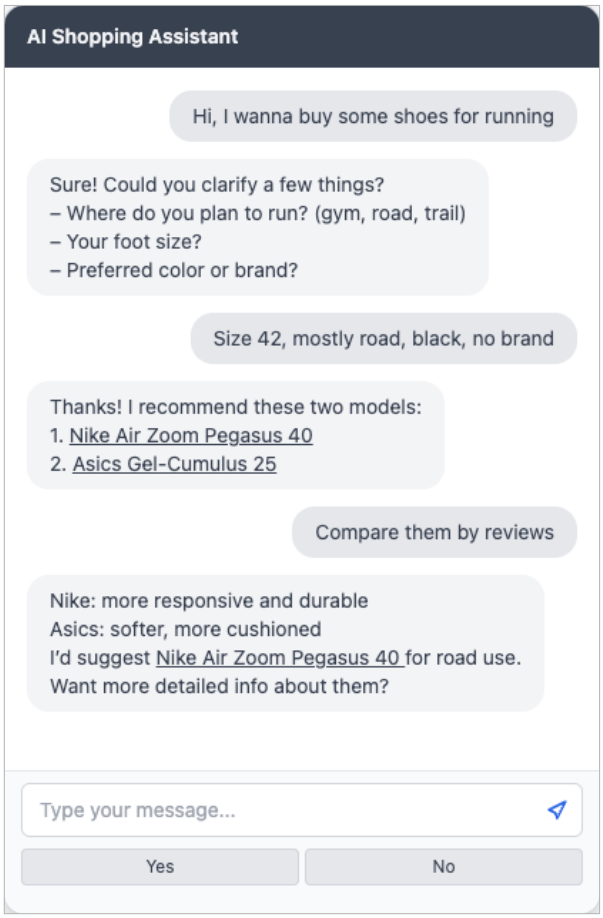

Figure 8: Assistant returns recommendations.

Step 3: Product retrieval. User constraints ("Size 42, mostly road, black, no brand") are parsed and stored in the conversation graph. The assistant then executes a filtered product query combining user context and inventory availability:

```
MATCH (p:Product)-[:BELONGS_TO]->(:Category {name: "Women's Running Shoes"})
WHERE p.size = 42 AND p.color CONTAINS "black"
  AND "road" IN p.suitableFor AND p.inStock = true
WITH p, p.avgRating * p.reviewCount as score
ORDER BY score DESC LIMIT 2
RETURN p.name, p.id, p.brand, p.price
```

Step 4: Recommendation presentation. The assistant returns two models with clickable product links constructed from Neo4j-retrieved IDs, enabling seamless transition to the storefront.

Step 5: Comparative analysis. When requested to "Compare them by reviews", the assistant retrieves aggregated review data with LLM-generated Cypher:

```
MATCH (p:Product)-[:HAS_REVIEW]->(r:Review)
WHERE p.id IN [$nikeId, $asicsId]
RETURN p.name, collect(r.sentiment), avg(r.rating)
```

The LLM synthesizes this into natural language ("Nike: more responsive and durable; Asics: softer, more cushioned") and recommends Nike based on the user's road running preference, applying domain knowledge that road runners prioritize responsiveness over maximum cushioning.

This demonstration highlights several architectural characteristics:

- **Context-aware personalization** – User history retrieval allows automatic filtering (women's shoes) before the first response.
- **Schema-driven adaptation** – Clarification questions are generated by querying the product taxonomy, adapting to catalog structure without code changes.
- **Natural language flexibility** – The assistant interprets colloquial phrasing ("shoes for running") and incomplete constraints ("mostly road") without requiring users to adopt catalog terminology.
- **Query-grounded recommendations** – All suggestions derive from explicit Cypher queries rather than generated content. MCP validation ensures only valid, in-stock items are returned.
- **Explainable reasoning** – Comparative analysis provides explicit justification ("for road use") based on query results rather than opaque ranking algorithms.
- **Conversational continuity** – Context persists across turns in the Neo4j conversation graph, avoiding repeated information requests.

During development testing, the system exhibited response latencies ranging from 0.8s to 3.2s depending on query complexity and the number of MCP tool invocations. Neo4j Cypher queries executed in 50–200ms, while the primary latency came from LLM inference (800–1500ms per Claude API call) and MCP protocol overhead (approximately 200ms per tool invocation). The synchronization pipeline maintained sub-5-second propagation delay under typical catalog update volumes. Query complexity scaled linearly with the number of graph hops rather than exponentially as would occur with equivalent SQL `JOIN`s, confirming the architectural benefit of graph-based traversal for multi-constraint product discovery.

5. Conclusion

This paper presented an architecture for conversational e-commerce that integrates LLMs, MCP servers, and graph databases. The system addresses the limitations of traditional search interfaces by enabling natural language interactions backed by structured knowledge graphs.

The implementation consists of several interconnected components. A modular architecture integrates Vendure, Neo4j, LLM, and MCP servers with clear separation of concerns. A CDC-based synchronization pipeline uses Debezium and Kafka to maintain consistency between PostgreSQL and Neo4j. The context-aware dialogue system retrieves user preferences and maintains conversational state. Graph-based knowledge representation encodes product relationships and user interactions for multi-hop queries. The demonstration illustrated how these components enable personalized product discovery, with the LLM autonomously determining required context and constructing validated Cypher queries to retrieve accurate results grounded in explicit database relationships.

This work presents a proof-of-concept implementation with several limitations. The system has not been evaluated under concurrent load or validated through A/B testing with real traffic. Security mechanisms have not undergone penetration testing. User studies are needed to validate usability assumptions. Production deployment would require comprehensive benchmarking and controlled experiments.

Future development should address these limitations while extending system capabilities. The graph schema could incorporate additional relationship types to support queries like outfit assembly or gift recommendations. Background workers could pre-process review data to reduce real-time token consumption. Security hardening requires implementing prompt injection prevention, data isolation, output validation, rate limiting, and query complexity scoring. Multi-modal support could extend the system to image-based search and voice input. An evaluation framework with quantitative benchmarks would enable systematic assessment of improvements and support the transition to production.

The proposed approach demonstrates that conversational interfaces can be integrated into existing e-commerce platforms through modular design, improving user interaction and reducing search complexity. While significant work remains before production deployment, this implementation provides a reference point for building AI-powered product discovery systems.

Author contributions

Conceptualization and writing – Andrii Baran; review and editing – Iryna Spivak and Svitlana Krepych. All authors have read and agreed to the published version of the manuscript.

Funding

This research received no external funding.

Data availability statement

No new data were created or analysed during this study. Data sharing is not applicable.

Conflicts of interest

The authors declare no conflict of interest.

Declaration on Generative AI

The authors used ChatGPT5 and Claude Opus 4.1 to support language editing, grammar correction, and structural refinement. All AI-generated outputs were thoroughly reviewed and adjusted by the authors. The authors bear full responsibility for the accuracy, originality, and integrity of the content presented in this publication.

References

[1] C. Tharp, New research: Search abandonment continues to vex retailers worldwide, 2023. URL: https://cloud.google.com/blog/topics/retail/new-research-on-search-abandonment-in-retail.

[2] Baymard Institute, 50 Cart Abandonment Rate Statistics 2026, 2026. URL: https://baymard.com/lists/cart-abandonment-rate.

[3] I. Zdatny, T. Mitchell, Understanding customer experience ROI, 2025. URL: https://www.qualtrics.com/articles/customer-experience/qualtrics-servicenow-customer-service-research/.

[4] S. Sousa, Always Explain Industry-Specific Filters (62% Don't), Baymard Institute, 2024. URL: https://baymard.com/blog/explain-industry-specific-filters.

[5] E. Scott, 2024 E-Commerce Product Finding: Expanded and Updated Research Findings, Baymard Institute, 2024. URL: https://baymard.com/blog/product-finding-2024-launch.

[6] What is Conversational Commerce?, Salesforce, 2023. URL: https://www.salesforce.com/commerce/conversational-commerce/.

[7] Mordor Intelligence, Conversational Commerce Market Size, Share & Trends Report 2031, 2026. URL: https://www.mordorintelligence.com/industry-reports/conversational-commerce-market.

[8] Y. Huang, J. Guo, W. Mao, C. Gao, P. Han, C. Liu, Q. Ling, Exploring the Landscape of Text-to-SQL with Large Language Models: Progresses, Challenges and Opportunities, CoRR abs/2505.23838 (2025). doi:10.48550/ARXIV.2505.23838. arXiv:2505.23838.

[9] J. Bodensohn, U. Brackmann, L. Vogel, A. Sanghi, C. Binnig, Unveiling Challenges for LLMs in Enterprise Data Engineering, Proc. VLDB Endow. 19 (2025) 196–209. URL: https://www.vldb.org/pvldb/vol19/p196-bodensohn.pdf.

[10] D. Mazurek, MCP Servers: The Key Integration Layer for Enterprise AI, Software Mind, 2025. URL: https://softwaremind.com/blog/mcp-servers-the-key-integration-layer-for-enterprise-ai/.

[11] M. Vizard, Visa Adds MCP Server to Bring AI Agents to Next Gen E-Commerce Platform, TechStrong.ai, 2025. URL: https://techstrong.ai/agentic-ai/visa-adds-mcp-server-to-bring-ai-agents-to-next-gen-e-commerce-platform/.

[12] P. Rathle, Driving Innovation in Retail with Graph Technology: How Top Retailers Use Neo4j, White Paper, Neo4j Inc., 2021. URL: https://neo4j.com/whitepapers/retailers-graph-technology-neo4j.

[13] J. Sidlauskiene, Y. Joye, V. Auruskeviciene, AI-based chatbots in conversational commerce and their effects on product and price perceptions, Electronic Markets 33 (2023) 24. doi:10.1007/s12525-023-00633-8.

[14] Anthropic, Introducing the Model Context Protocol, 2024. URL: https://www.anthropic.com/news/model-context-protocol.

[15] G. Stalidis, I. Karaveli, K. Diamantaras, M. Delianidi, K. Christantonis, D. Tektonidis, A. Katsalis, M. Salampasis, Recommendation Systems for e-Shopping: Review of Techniques for Retail and Sustainable Marketing, Sustainability 15 (2023) 16151. doi:10.3390/su152316151.

[16] V. Krutko, I. Spivak, S. Krepych, An approach to assessing the reliability of software systems based on a graph model of method dependence, in: S. O. Semerikov, A. M. Striuk (Eds.), Proceedings of the 6th Workshop for Young Scientists in Computer Science & Software Engineering (CS&SE@SW 2023), Virtual Event, Kryvyi Rih, Ukraine, February 2, 2024, volume 3662 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 37–47. URL: https://ceur-ws.org/Vol-3662/paper11.pdf.

[17] O. Poliarush, S. Krepych, I. Spivak, Hybrid approach for data filtering and machine learning inside content management system, Advanced Information Systems 7 (2023) 70–74. doi:10.20998/2522-9052.2023.4.09.

[18] S. Krepych, I. Spivak, S. Spivak, R. Krepych, The method of assessing the reliability of software systems based on a graphic model of the dependence of methods of the system under test, Advanced Information Systems 9 (2025) 58–67. doi:10.20998/2522-9052.2025.2.08.

[19] Neo4j Inc., Neo4j Surpasses $200M in Revenue, Accelerates Leadership in GenAI-Driven Graph Technology, 2024. URL: https://neo4j.com/press-releases/neo4j-revenue-milestone-2024/.

[20] G. Morling, Practical Change Data Streaming Use Cases with Apache Kafka & Debezium, InfoQ, 2020. URL: https://www.infoq.com/presentations/data-streaming-kafka-debezium/.

[21] The PostgreSQL Global Development Group, PostgreSQL: Documentation: 18: 28.3. Write-Ahead Logging (WAL), 2026. URL: https://www.postgresql.org/docs/current/wal-intro.html.

[22] M. Kleppmann, Making Sense of Stream Processing: The Philosophy Behind Apache Kafka and Scalable Stream Data Platforms, O'Reilly Media, Inc., 2016. URL: https://martin.kleppmann.com/papers/stream-processing.pdf.