GraphQLConf
2026

hosted by GraphQL Foundation

GraphQL Meets LLMs & Agents

Sharon Gorla · Lead Engineer, Starbucks

Agenda

01 Starbucks GraphQL Journey

02 AI Changes the Client

03 What GraphQL Originally Solved

04 Why GraphQL Aligns Naturally with AI

05 Comparing API Patterns for AI Agents

06 Token & Context Efficiency

07 Production Governance

08 AI Readiness & Closing Thoughts

8 Years of Investment. Built for Scale. Perfectly Positioned for AI.

From a single POC to enterprise scale — GraphQL is now the execution layer for AI agents at Starbucks.

2016 — Orchestra GraphQL POC — Web

First GraphQL proof-of-concept for the web channel.

Single team adoption.

2019 — Orchestra GraphQL — Mobile Order & Pay (MOP)

GraphQL expanded to Mobile + Web.

A handful more teams adopt internal GraphQL servers.

2022 — Simplicity POC — Next-Gen POS

GraphQL Server + Client for NGPOS.

8 teams in production, 12 teams championing the GraphQL community at Starbucks.

2023 — Federation POC — Enterprise Platform

Apollo Federation explored for enterprise-wide platform consolidation and distributed team ownership.

2024 — Next-Gen POS — Production GraphQL BFF

Full production GraphQL BFF for Next-Gen POS.

In-store ordering stack runs on GraphQL end-to-end.

2025+ — GraphQL Meets AI Agents

MCP auto-generation from schema.

75x token reduction vs REST.

AI-powered on-call, support, and operations.

8 years of GraphQL investment at Starbucks — now supercharged for the AI era.

A Thriving Community That Drives Innovation and Impact

12+ Teams Across Starbucks

7 in Production
5 Exploring

GraphQL Champions

- Champions with deep domain expertise
- Advocating best practices and driving adoption

Knowledge & Collaboration

- Confluence Space
- #graphql Slack Channel
- Internal conferences & workshops

Events & Celebrations

- Starbucks Innovation Expo
- Partner-led workshops
- GraphQL Week to celebrate teams using GraphQL

External Engagements

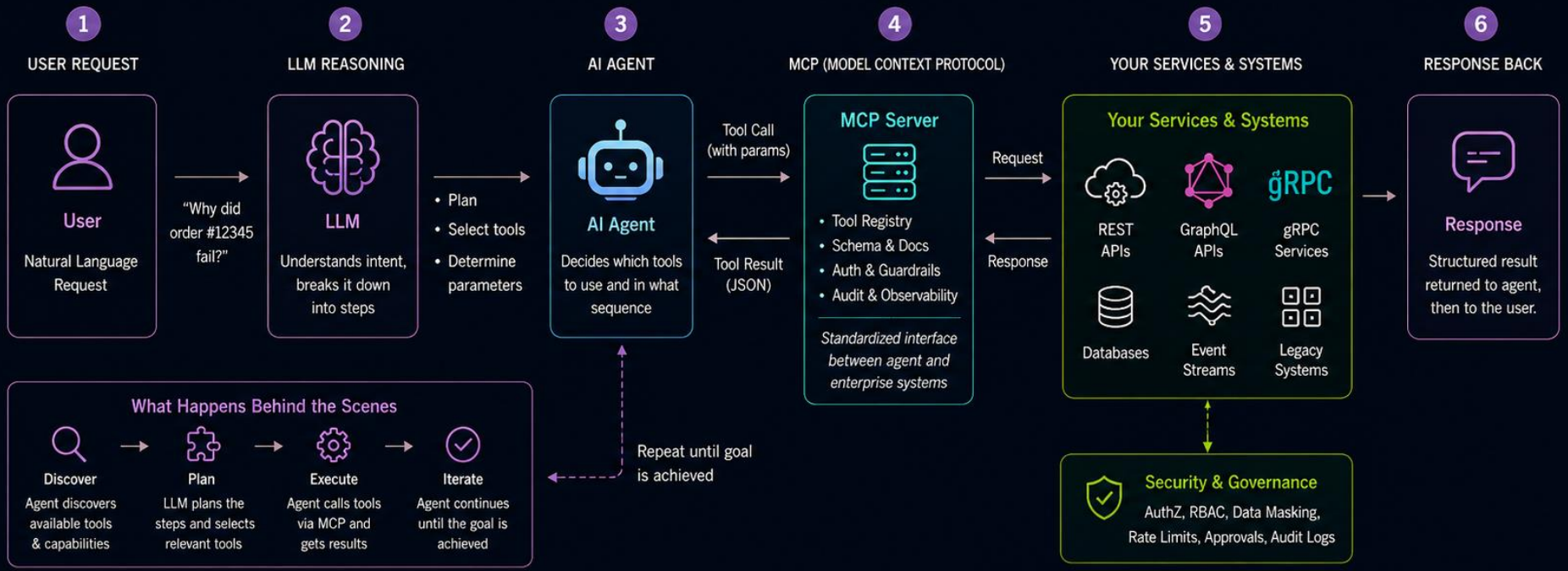
- Lesbians Who Tech Conference
- Management Leadership for Tomorrow
- GraphQL Summit
- Grace Hopper
- Apollo GraphQL Champion Community
- GraphQLConf 2026

> "Champions realize that defeat and learning from it, is even more than from winning — is part of the path to mastery."
>
> STARBUCKS®

Strong community. Shared learning. Enterprise impact.

From Natural Language to Action Using Your Systems

AI Agents use tools (via MCP) to securely discover, access, and act on your existing services and data.



Why This Matters

No New APIs
Leverage what you already have

Secure by Design
Built-in controls at every layer

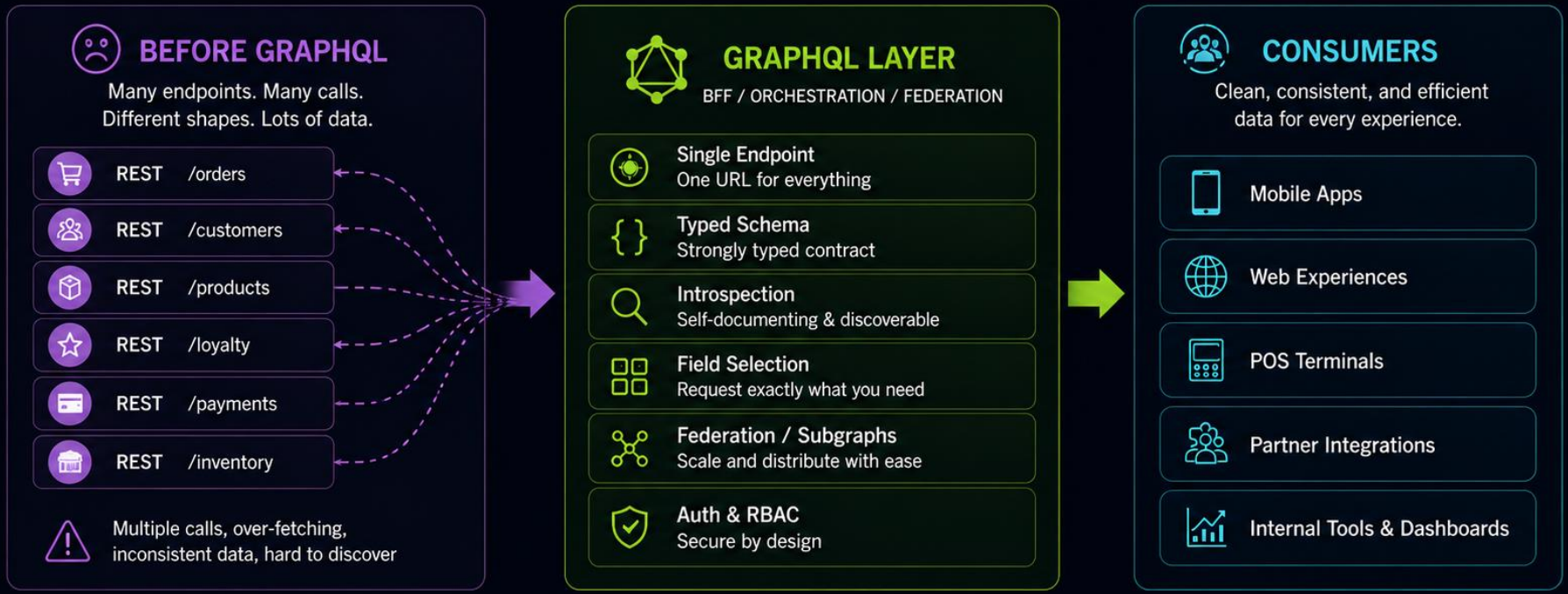
Context Efficient
Agents get exactly what they need

Enterprise Ready
Scales across systems, teams and domains

AI Agents are not replacing your systems.
They're orchestrating them.

What GraphQL Was Built For (Before GenAI)

GraphQL solved a developer problem: too many APIs, too much over-fetching, too little discoverability.



GraphQL gave developers: one contract, one endpoint, typed queries, and real discoverability.

THE CONVERGENCE

GraphQL Was Accidentally Built for AI

The features we built for developers are exactly what AI agents need.

GRAPHQL FEATURES	AI BENEFITS
Introspection Self-describing schema	**Tool Discovery** AI discovers capabilities at runtime
Field Selection Request only what's needed	**Token Efficiency** Dramatically less context, lower cost
Descriptions Schema & field documentation	**AI Documentation** Every description becomes tool guidance
RBAC & Auth Field-level security & permissions	**AI Guardrails** Safe, authorized AI interactions
Federation Distributed schema ownership	**Domain AI** AI works across domains with one graph

GraphQL wasn't built for AI. But it turns out—AI was built for GraphQL.

API TECHNOLOGY COMPARISON FOR AI AGENTS

Dimension	REST	GraphQL	gRPC	Direct DB Access
Interface	Multiple endpoints	Single endpoint	Multiple services	SQL / Driver connection
Data Shape	Fixed by server (over/under-fetching)	Client specifies exact fields	Fixed by .proto (contract)	Full tables / rows
Round Trips for Complex Task	 High (3-10+)	 Low (1)	 Medium (2-5)	 Low (1)
Discoverability for LLMs	Poor (docs needed)	Excellent (introspection)	Good (reflection)	Poor (schema not semantic)
Type Safety & Contracts	Medium (OpenAPI)	Excellent (Strong Schema)	Excellent (.proto)	Low (raw schema)
Versioning Complexity	High	Low	Medium	High
Best For	Simple CRUD, Public APIs	Agentic apps, Composable data, BFFs	High performance, Streaming, Microservices	Internal batch jobs, Analytics, Migrations

For LLM Agents: GraphQL provides the best balance of discoverability, precision, and efficiency.

GraphQL vs Its Peers: Honest Trade-offs

GraphQL is not the right tool for every problem. Here's an honest look at where each approach wins and struggles.

	REST	GraphQL	gRPC	tRPC	OpenAPI (Spec)
Simplicity / learning curve	✓ Low — everyone knows it	⚠ Medium — schema, resolvers, N+1 risk	✗ High — protobuf, codegen required	✓ Low (TypeScript teams only)	✓ Low — just YAML + docs
Over-fetching (token efficiency)	✗ Always over-fetches fixed shapes	✓ Client requests exact fields only	✗ Full protobuf message returned	✗ Full object returned	✗ Endpoint-level no field control
Caching	✓ HTTP cache out of the box	⚠ Complex — POST by default, APQ needed	✓ HTTP/2 framing cache-friendly	⚠ React Query per-team setup	✓ HTTP cache out of the box
Performance (internal services)	⚠ OK, multiple roundtrips	⚠ OK, resolver chain overhead	✓ Binary protocol, lowest latency	⚠ JSON over HTTP, no binary	— Spec only, impl varies
Discoverability & self-documenting	⚠ Needs OpenAPI spec (manual)	✓ Introspection is built-in	⚠ Protobuf schema requires tooling	⚠ TypeScript types only, no runtime	✓ Spec is the documentation
Federation / composition	✗ Manual — no standard	✓ Federation	⚠ Envoy / service mesh required	✗ Monorepo only	✗ No composition standard
AI agent tooling (MCP / tool gen)	⚠ Manual OpenAPI → tool mapping	✓ Auto-generated from introspection	⚠ Proto → tool conversion needed	⚠ TypeScript types only	✓ Good via OpenAPI plugins
Best for	Simple CRUD, public APIs, micro-services	Complex graphs, BFFs, AI agents, multi-consumer APIs	Internal micro-services, low-latency RPC	Full-stack TypeScript, monorepos	API contracts, documentation, mock generation

✓ = clear win ⚠ = works with caveats ✗ = real weakness Note: GraphQL's caching & learning curve are genuine trade-offs worth planning for.

Token Cost Comparison (Live Demo Alternative)

The Question: "What did Alice order?"

Same business question. Different payload efficiency.

Traditional REST-style integration (illustrative over-fetch example) · 72 fields

```
query RESTStyleOverfetch {
  order(id: "1") {
    id orderNumber status subtotal
    tax total paymentMethod notes
    createdAt updatedAt completedAt
    isRefunded
    customer {
      id name email phone
      loyaltyId orderCount
      totalSpent createdAt
    }
    items {
      id quantity priceAtOrder
      subtotal customizations
      product {
        id name category price
        description calories
        inventory
      }
    }
  }
}
```

AI-optimized GraphQL query · 11 fields

```
query AIOptimized {
  order(id: "1") {
    status
    total
    customer {
      name
      loyaltyId
    }
    items {
      quantity
      product { name }
    }
  }
}
```

Answer: Completed · $6.47 · 1 Caramel Macchiato · Alice (LYL-001)

Response Comparison

3 REST API calls

vs

1 GraphQL API calls

606 REST tokens

vs

63 GraphQL tokens

Metric	REST (3 API calls)	GraphQL (1 call)	Reduction
API Calls Required	3	1	Illustrative reduction in roundtrips
Fields Returned	72	11	6.5× fewer fields
Response Size (bytes)	2,203	327	6.7× smaller payload
Estimated Tokens	606	63	**Illustrative token reduction example** ✓

✨ **Example payload reduction: ~90% fewer response tokens in this sample workload**

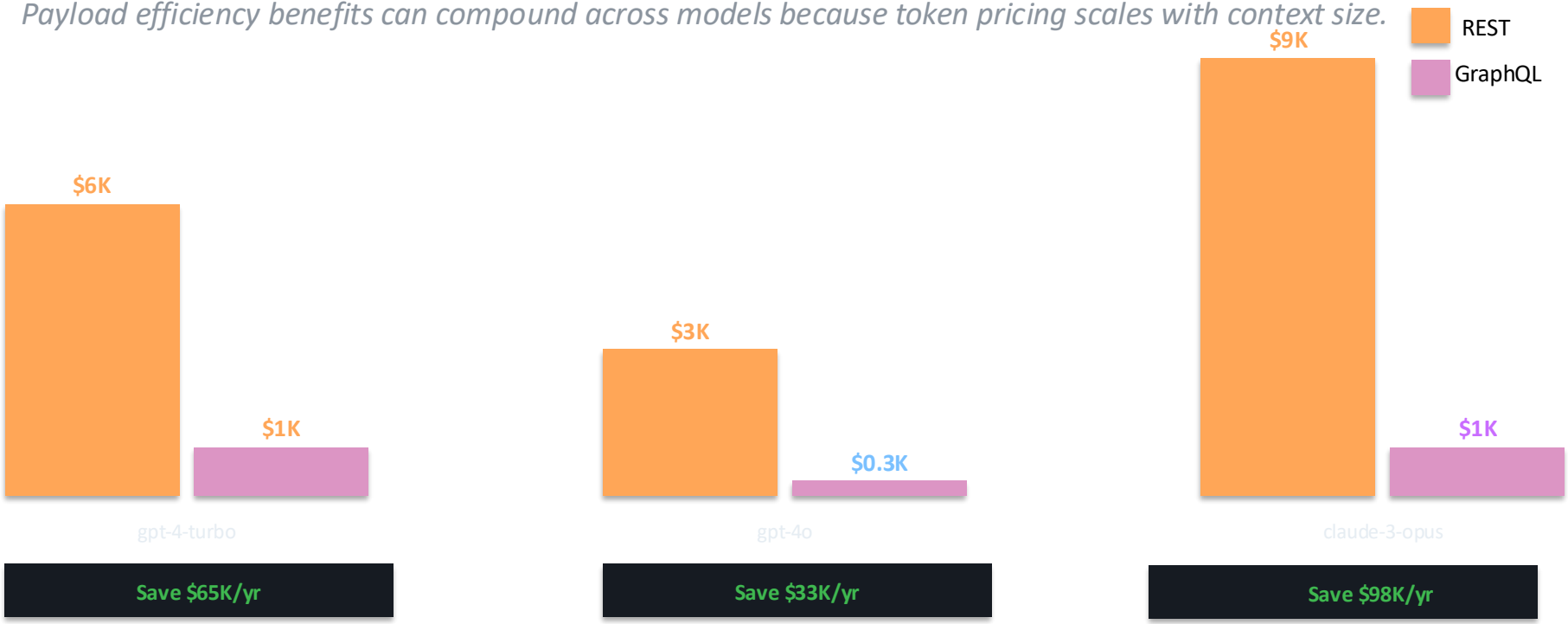
Cost Projections at Scale (GPT-4 Turbo pricing)

Scale	REST Cost	GraphQL Cost	Monthly Savings
Per Query	$0.0061	$0.0006	$0.0054
Per Hour (1K queries)	$6.06	$0.63	$5.43
Per Day (100K queries)	$606	$63	$543
Per Month (1M queries)	$6,060	$630	$5,430
Per Year (10M queries)	**$61K**	**$6,300**	**$54K**

Illustrative cost model showing how selective payloads can materially reduce LLM context costs at scale.

Savings Across Every Major Model (1M queries/month)

Payload efficiency benefits can compound across models because token pricing scales with context size.



Selective payload design can significantly reduce context/token costs across many LLM workloads.

TOKEN COST COMPARISON (EXAMPLE WORKFLOW)

Task: “Show me delayed orders in store 123 with customer name and items”

REST (Multiple Calls)

1) GET /orders?store=123&status=DELAYED
2) GET /customers?ids=...
3) GET /order-items?orderIds=...

Multiple requests + responses

Call 1 (Orders) + Call 2 (Customers) + Call 3 (Items)

Payload Transferred
~12,450 bytes

Estimated Tokens
(Input + Output)

Relative Cost
$ $ $ $ $ $

Low

GraphQL (Single Query)

```
query {
  orders(store: "123", status: DELAYED) {
    id
    customer { name }
    items { name, status }
  }
}
```

Single request + precise response

~1,050 bytes

~650 tokens

$

High

gRPC (Multiple Calls)

1) GetOrders(Request)
2) GetCustomers(Request)
3) GetOrderItems(Request)

Multiple requests + responses

Call 1 + Call 2 + Call 3

~4,200 bytes

~2,800 tokens

$$$

Medium

Direct DB Access (SQL)

```
SELECT o.id, c.name, i.name, i.status
FROM orders o
JOIN customers c ON o.customer_id = c.id
JOIN order_items i ON i.order_id = o.id
WHERE o.store_id = 123 AND o.status = 'DELAYED';
```

Single query + raw result

Payload Transferred	~900 bytes
Estimated Tokens (Input + Output)	~550 tokens
Relative Cost	$
Efficiency	High (but less governed)

TOKEN COST IMPACT (Illustrative)

REST ~8,600 tokens → GraphQL ~650 tokens → UP TO ~13X LOWER TOKEN USAGE

KEY TAKEAWAYS

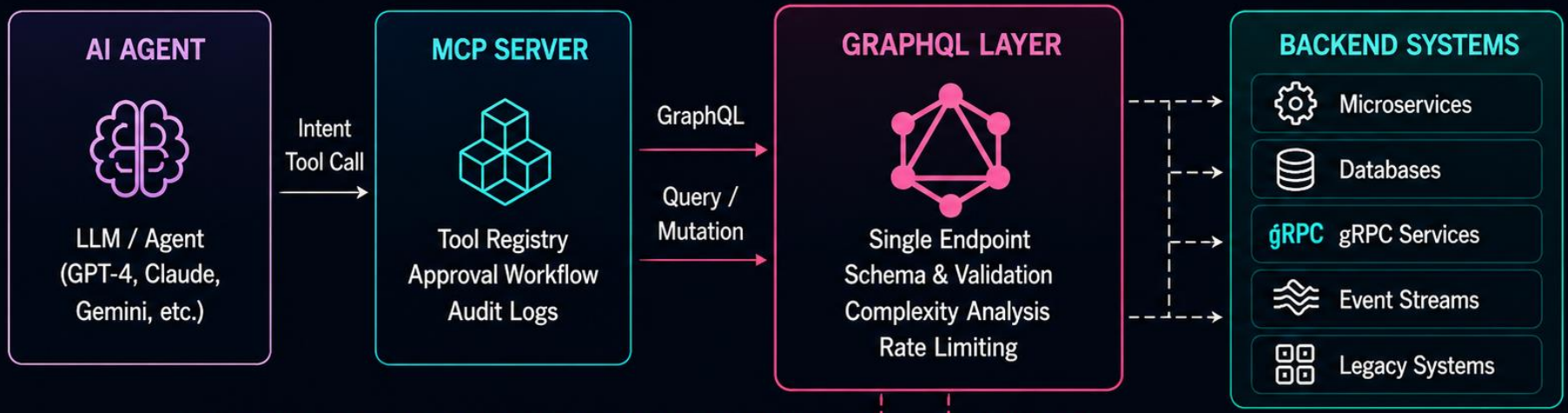
- ✓ More calls = more content, more tokens, more cost
- ✓ Over-fetching is the hidden token killer
- ✓ GraphQL dramatically reduces payload & tokens
- ✓ Direct DB access can be efficient but bypasses governance and business rules
- ✓ Actual numbers vary by use case, schema, and prompt design.

Production Concerns

GRAPHQL AS THE SECURITY LAYER

Secure by Design, Not by Bolt-On

GraphQL acts as the intelligent control plane between AI agents and your systems.



AI can ask anything. GraphQL ensures it **only** gets what it's allowed to.

AI Risks → GraphQL Mitigations

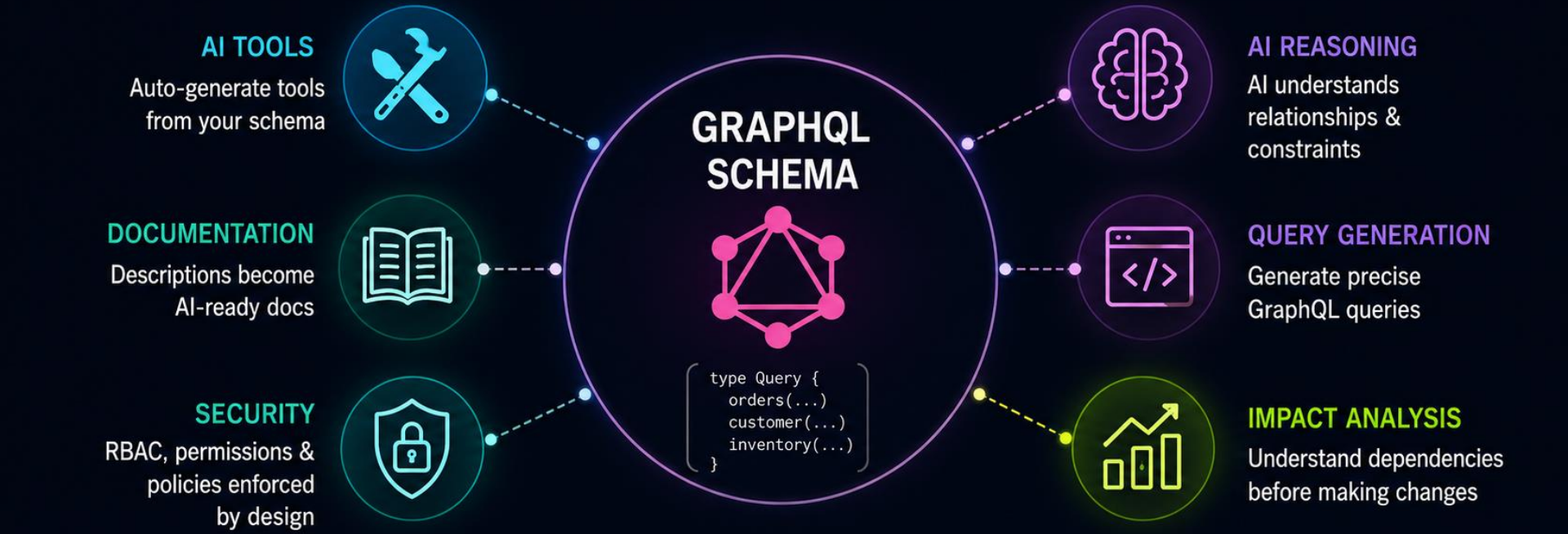
Proactively address risks. GraphQL gives you the control.

AI RISKS	GRAPHQL MITIGATIONS
1. Unauthorized Data Access AI agents may access more data than needed, leading to privacy and security breaches.	**1. Auth & Fine-Grained Access Controls** Enforce field-level permissions and role-based access to ensure least-privilege access.
2. Over-Fetching & Data Leakage AI queries may fetch excessive or sensitive data, increasing exposure.	**2. Query Validation & Depth Limits** Validate queries, limit depth and complexity, and prevent excessive data retrieval.
3. Prompt Injection & Misuse AI agents can be manipulated to run harmful or unintended operations.	**3. Input Validation & Safe Resolvers** Validate inputs, sanitize data, and use safe, intent-driven resolvers.
4. Lack of Auditability & Traceability Hard to track what data was accessed or what actions were taken.	**4. Auditing & Observability** Log queries, track resolver usage, and monitor agent behavior in real time.
5. Uncontrolled Costs & Resource Abuse Complex or malicious queries can cause high costs and system strain.	**5. Rate Limiting & Query Complexity** Limit request rate and query complexity to protect performance and control costs.

What You Can Unlock

WHAT YOUR SCHEMA BECOMES

Your schema stops being documentation. It becomes the AI execution layer.



Your schema is the contract. AI makes it **intelligent.**

YOU'RE CLOSER THAN YOU THINK

YOUR EXISTING GRAPHQL PLATFORM

BFF / Orchestration Layer

Apollo Federation

Single GraphQL Schema

You already have:

✓ Resolvers ✓ Schema ✓ Auth ✓ Domain Knowledge

MCP SERVER (AUTO-GENERATED)

Tooling Layer for AI Agents

AI AGENTS & EXPERIENCES

Support AI

On-call AI

Developer AI

Mobile Assistant

If you already have GraphQL, the hardest part is already done.

WHAT THIS UNLOCKS

New Experiences. Built on Your Existing Graph.

GraphQL + AI agents unlock powerful experiences across your organization.

Conversational Order History

"Show me my last 5 orders" and what happened"

AI Support Agent

Instant, accurate answers from enterprise data

Natural Language Dashboards

"Show me failed payments" by store this week"

AI Incident Detection

Detect anomalies and patterns before customers do

Personalized Mobile Assistant

Proactive suggestions based on real-time context

Secure Self-Service AI

Empower users with safe, approved AI actions

Your GraphQL graph becomes an intelligent interface for every experience.

Where to Start: AI Maturity on GraphQL

You don't need Federation to start. You don't need AI to start. Pick your level.

Level 1

Any GraphQL schema

Effort: Hours

- Enable authenticated introspection
- Add descriptions to your Query fields
- Point an MCP server at your endpoint
- → AI agents can discover and invoke APIs as tools

Level 2

GraphQL + field auth

Effort: Days

- Add @auth directives to sensitive fields
- Create role-scoped AI tokens
- Add query complexity limits
- → More production-safe AI access patterns

Level 3

Federation

Effort: Weeks

- Subgraph teams add AI-friendly descriptions
- Router-level rate limiting for AI traffic
- Cross-domain AI queries out of the box
- → Enterprise-grade AI platform

The Realization

Then GenAI arrived.

And suddenly...

everything GraphQL solved for developers...

...became exactly what AI needed.

Thank You

Sharon Gorla · Lead Engineer, Starbucks

- GraphQL = natural AI interface (introspection + field selection)
- MCP auto-generation = zero-config AI tools from your schema
- Selective GraphQL payloads can materially reduce token usage and AI infrastructure costs
- Complexity limits + RBAC + approvals = safe AI in production

Questions?