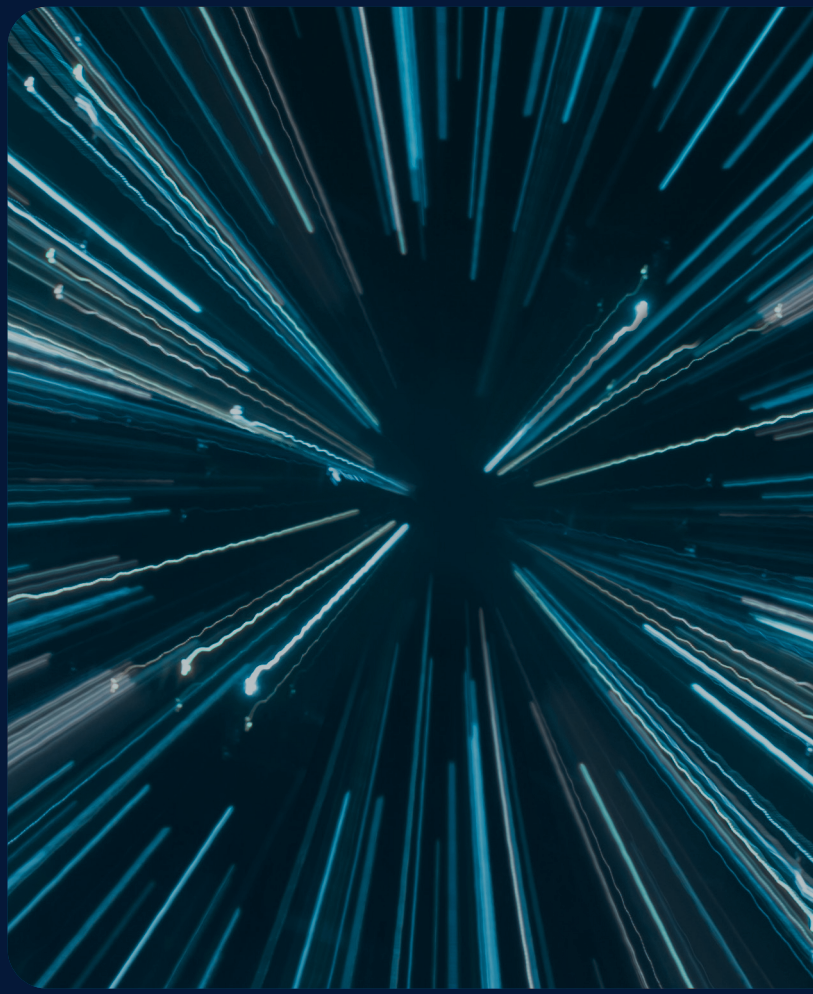
The Complete Playbook for

Using Real-Time Customer Data to Power AI Recommendations

TEALIUM®

Contents



Recommendation engines influenced 19% of all e-commerce orders in 2024, driving $229 billion in global online sales during the holiday season alone. That number will keep climbing. But here's the part most teams don't talk about: the recommendation model gets all the attention, while the data pipeline determines whether it actually works.

A recommendation engine trained on batch-processed data is making decisions about a customer who existed six hours ago. The customer browsing your site right now, adding items to a cart, comparing prices on a competitor's tab? Your model doesn't know about any of that yet. By the time it does, intent has expired. The offer that would have converted becomes noise.

This playbook is for teams that are past the "should we invest in AI recommendations?" stage and into the "how do we actually architect this?" stage. It covers the full pipeline, from behavioral event capture to served recommendation, with specific technical patterns, real benchmarks, and honest trade-offs at every step.

What Real-Time AI Recommendations Actually Require

Sub-100ms p99 latency. That's the production bar for recommendation serving in 2026. Anything slower and the customer perceives a delay. Anything much slower and you're serving recommendations after the moment they were relevant.

Hitting that number requires four things working in concert: streaming data ingestion that captures behavioral signals as they happen, a feature store that makes those signals immediately queryable, model inference fast enough to score candidates in single-digit milliseconds, and an activation layer that delivers the recommendation to the right channel before context changes.

None of these are optional upgrades to a batch architecture. You can't bolt real-time onto a system designed for hourly syncs. This is an architecture decision, not a feature toggle. Teams that try to incrementally add real-time capabilities to batch pipelines end up with the worst of both worlds: the complexity of streaming with the latency of batch. It's one of the most common mistakes in this space, and it usually takes six months and a missed holiday season to fully appreciate why it doesn't work.

The rest of this playbook walks through each layer, with specific technology choices, benchmarks, and the trade-offs you'll actually face in production.

.

The Real-Time Data Pipeline: From Signal to Recommendation

A real-time recommendation pipeline has six stages. Each one introduces latency, and your total budget across all of them is under 100 milliseconds.

Stage 1: Event Capture. Every customer interaction generates a signal: page views, product clicks, cart additions, purchases, scroll depth, search queries, even session timing patterns. These events need to be captured at the source, structured consistently, and emitted as a stream. Tealium's EventStream handles this by ingesting events across web, mobile, and server-side sources in real time, producing consistently structured records regardless of where the interaction originated.

Stage 2: Streaming Ingestion. Events flow into a streaming platform like Apache Kafka, Amazon Kinesis, or Google Cloud Pub/Sub. The key requirement here is throughput under load. During peak traffic (think Black Friday, product launches, flash sales), your ingestion layer needs to handle millions of events per second without dropping data or introducing backpressure.

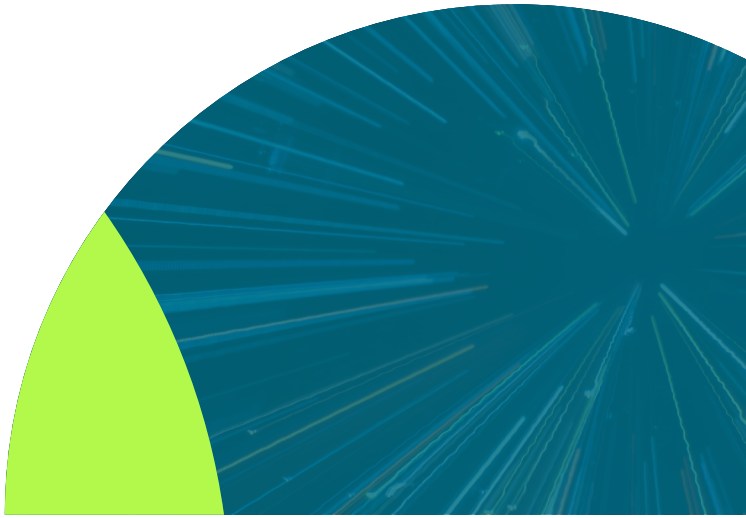
Stage 3: Real-Time Enrichment. Raw events are useful but not sufficient. Enrichment adds business context in-flight: resolving identity across devices, appending customer segment membership, computing embeddings for similarity matching, or flagging consent status. This step is where raw clickstream becomes AI-ready data (structured, enriched, and immediately consumable by a model without further transformation).

Stage 4: Feature Store. Enriched features land in a low-latency feature store where they're immediately queryable. The feature store serves as the bridge between your data pipeline and your model. It stores both real-time features (updated continuously from streaming data) and batch features (computed periodically from historical data). More on this in a dedicated section below.

Stage 5: Model Inference. When a recommendation request fires, the serving layer retrieves the customer's current features, scores candidate items against the model, and ranks results. This step typically needs to complete in under 20 milliseconds to leave room for network overhead and post-processing.

Stage 6: Activation. The ranked recommendations reach the customer through whatever channel they're in: website personalization, mobile push, email, in-store associate screen, call center agent dashboard. The activation layer needs to support multiple channels simultaneously, because the same customer might be interacting across more than one.

The entire sequence, from event capture to served recommendation, completes in under 100 milliseconds in a well-architected system. Netflix does this for 230 million concurrent users. The architecture patterns are proven. The question is whether your data infrastructure supports them.



Capturing Behavioral Data Across Every Channel

On web: page views, product interactions (view, click, add-to-cart, wishlist), search queries and refinements, scroll depth, session duration, referral source, and exit behavior. On mobile: all of the above, plus app-specific events like push notification opens, in-app message engagement, and location signals (with consent). From IoT and connected devices: interaction events, usage patterns, and state changes. From offline channels: point-of-sale transactions, call center interactions, in-store kiosk activity, and loyalty program engagement.

The hard part isn't capturing any one of these. It's capturing all of them in a consistent schema, in real time, with identity stitched across channels. A customer who browses shoes on mobile, researches reviews on desktop, and walks into a physical store is one person with one intent. If your data pipeline treats them as three separate anonymous sessions, your recommendation model is working with a fraction of the signal.

This is where identity resolution becomes foundational. Tealium's AudienceStream CDP resolves cross-channel identities in real time, stitching together device IDs, cookies, login events, and CRM records into a single unified profile. When the recommendation model queries for a customer's features, it gets the full picture, not a device-specific fragment.

Enrichment matters too. Appending business context during ingestion (customer lifetime value, segment membership, consent status, real-time intent scores) means your feature store receives data that's already structured for model consumption. Enrichment in-flight is far more efficient than enrichment at query time. You pay the compute cost once, upstream, instead of repeatedly at inference.

Building a Low-Latency Feature Store

The feature store is the component most teams underestimate. It sits between your data pipeline and your model, and its performance directly determines whether you hit your latency targets.

A feature store manages two types of features. **Real-time features** update continuously from streaming data: the customer's current session behavior, items in their active cart, pages viewed in the last five minutes. **Batch features** are computed periodically from historical data: lifetime purchase history, long-term preference patterns, demographic attributes. Both types need to be retrievable in single-digit milliseconds at inference time.

Technology choices vary based on scale and latency requirements:

Redis is the most common choice for real-time feature serving. Redis 8's vector search benchmarks on 1 billion 768-dimensional vectors show roughly 200ms median latency at 90% precision with 50 concurrent queries, and it sustains 66,000 vector insertions per second. For feature lookup (not vector search), Redis delivers sub-millisecond reads. Relevance AI reported reducing their vector search latency from 2 seconds to 10 milliseconds after migrating to Redis.

Vector databases (Pinecone, Weaviate, Qdrant) specialize in similarity search for embedding-based recommendations. They're purpose-built for the "find items most similar to this user's preference vector" query pattern. Trade-off: they're optimized for vector operations but less flexible for general feature serving.

Cloud data warehouses with caching layers (BigQuery with Bigtable, Snowflake with a Redis cache) work for teams that want to keep their existing warehouse as the system of record while adding a fast serving layer. The architecture is more complex, but it avoids migrating your entire feature pipeline to a new system.

Semantic caching is worth special attention. When recommendation queries show semantic similarity (and roughly 31% of queries do across the industry), a semantic cache can serve previous results instead of rerunning inference. The impact is significant: AWS experiments showed 86% cost reduction and 88% latency improvement with semantic caching. One healthcare company achieved a 70% cache hit rate, saving 70% on LLM inference costs. Redis LangCache delivers up to 15x faster responses on cache hits compared to running full model inference.

The trade-off with aggressive caching is freshness, and it's sneaky. A cached recommendation from 30 seconds ago is almost certainly fine. A cached recommendation from 30 minutes ago might reflect stale intent. The problem is you won't notice the staleness in your aggregate metrics until it's been silently degrading conversion for weeks. Your cache TTL strategy needs to balance cost savings against recommendation relevance, and the right answer depends on how quickly your customers' intent shifts. Start conservative (short TTLs), measure the impact, and extend from there.

Hybrid Models: Getting Recommendation Accuracy Right

No single modeling approach wins across all recommendation scenarios. The systems that perform best in production combine multiple techniques.

Collaborative filtering analyzes patterns across users. “Customers who bought X also bought Y.” It’s effective when you have dense interaction data but struggles with new users (the cold-start problem) and new items that haven’t accumulated enough interaction history.

Content-based filtering takes the opposite approach. Instead of looking at user behavior patterns, it analyzes item attributes: brand, price range, category, description keywords. “This product shares characteristics with items you’ve purchased.” Cold-start isn’t a problem here because new items have attributes from day one. The downside is narrowness. A pure content-based system recommends more of what you already like, which sounds helpful until your “personalized” experience becomes a hall of mirrors.

Hybrid approaches combine both. They’re also what most production systems actually run. Netflix’s recommendation engine is a hybrid system. So is Amazon’s. The two-tower architecture has become a common pattern: one neural network tower encodes user features, another encodes item features, and the model learns to predict relevance by comparing the two embedding vectors. This architecture scales well across large catalogs because you can pre-compute item embeddings offline and only compute the user embedding in real time.

Context matters as much as method. Time of day, device type, location, session depth, and recency of interactions all influence what recommendation will be relevant. A customer browsing at 9 AM on a desktop during a work break has different intent than the same customer browsing at 9 PM on their phone. Your model needs real-time contextual features (from your feature store) to distinguish between these scenarios.

One practical note on model architecture: deep learning approaches, specifically neural collaborative filtering, produce strong offline metrics. Google Research has noted, however, that they can be too computationally expensive for production serving at scale. Many production systems use deep learning for candidate generation (narrowing millions of items to hundreds) and simpler, faster models for final ranking. This two-stage approach lets you capture the accuracy benefits of deep learning without blowing your latency budget.

Serving Recommendations at Scale

Your model is trained. Your feature store is fast. Now you need to serve recommendations to millions of concurrent users without missing your latency SLA.

The serving architecture typically looks like this: a user action triggers a recommendation request. The serving layer fetches the user's current features from the feature store (1-5ms). The model scores candidate items and ranks them (5-20ms). Business rules apply post-processing: diversity constraints, inventory availability, frequency capping, promotional boosts (2-5ms). The response returns to the client.

Total inference time: 10-30ms, leaving comfortable headroom within a 100ms end-to-end budget for network overhead.

Scaling this requires a few specific patterns:

Continuous batching dynamically manages request queues without waiting for full batches to accumulate. Traditional batching optimizes throughput but introduces latency spikes. Continuous batching maintains throughput while keeping tail latency predictable.

Model quantization trades a small amount of precision for significant speed gains. Moving from FP32 to FP16 doubles inference speed. FP8 (supported on NVIDIA H100 GPUs) delivers 4x improvement. INT4 quantization, useful for edge deployment, can reach 10-15x speedup with carefully managed accuracy degradation.

Disaggregated serving separates the candidate generation and ranking stages across different compute resources, each optimized for its workload. This is how Netflix and Spotify maintain sub-100ms latency at global scale.

For production rollout, resist the urge to deploy to 100% of traffic on day one. Start with a small segment. Run A/B tests against your existing system (or against no recommendations at all) with clearly defined success metrics: conversion rate, click-through rate, revenue per session. Monitor for model drift, where performance degrades over time as user behavior shifts away from the training distribution. And watch for feedback loops, where the model's own recommendations influence future training data, creating a narrowing cycle that reduces recommendation diversity.

Explainability tools like SHAP and LIME help you understand why the model makes specific recommendations. This matters for debugging, for building trust with business stakeholders, and increasingly for regulatory compliance. If a customer asks why they were shown a particular product, "the model said so" isn't an acceptable answer.

Measuring What Matters

You need two categories of metrics: business outcomes and system health.

Business metrics tell you whether the recommendations are working. Conversion rate (CVR) on recommended items versus non-recommended alternatives is the most direct signal. Average order value (AOV) for sessions with and without recommendations tells you whether the engine is expanding baskets or just redirecting existing spend. Click-through rate (CTR) on recommendation widgets measures whether customers even notice what you're serving. And revenue attribution, the percentage of total revenue that touches a recommendation somewhere in the purchase path, tells you how load-bearing the system has become. Product recommendations drive up to 31% of e-commerce site revenue in mature implementations. If you're in single digits, there's room to run.

System metrics tell you whether the infrastructure is healthy. Track p50 and p99 latency for recommendation serving. Track cache hit rates (if you're running semantic caching, a 70%+ hit rate is a strong indicator of efficiency). Track feature freshness: how old is the most recent feature update when inference runs? If your "real-time" features are consistently 30 seconds stale, you have a pipeline bottleneck to investigate. Track model accuracy over time to catch drift early.

Build real-time dashboards for both categories. Automated alerting on latency spikes, accuracy drops, or sudden changes in CTR will catch problems before they become customer-facing incidents. The recommendation engine is a revenue-generating system. Treat its monitoring with the same rigor you'd apply to your payment processing pipeline.

Privacy, Consent, and Governance

Real-time behavioral data collection at this scale creates real privacy obligations. Treat them as architectural requirements, not compliance afterthoughts.

GDPR, CCPA, and sector-specific regulations like HIPAA establish clear requirements: explicit consent for data collection, transparent disclosure of how data is used, the right to access and delete personal data, and limitations on data retention. These aren't edge cases. They apply to every behavioral event flowing through your recommendation pipeline.

The architectural implication is that consent status needs to travel with the data, not be checked after the fact. When an event enters your streaming pipeline, consent should be resolved at ingestion time. Events from users who haven't consented to recommendation-related data use should never reach your feature store or model training pipeline. Tealium's consent management architecture enforces this at the data layer, evaluating consent preferences in real time as events are processed. Think of it less as a compliance checkbox and more as a circuit breaker: non-consented data never propagates downstream because the system won't let it through.

Beyond regulatory compliance, explainability and opt-out mechanisms build customer trust. If customers can see why they received a recommendation and can opt out of recommendation-driven experiences, you reduce the "creepy factor" that undermines personalization efforts. Recommendations should feel helpful, not surveilled.

Data governance also covers model inputs. Document which features your recommendation model consumes, which data sources feed those features, and what consent basis applies to each. When regulators or internal auditors ask how your AI makes decisions, you need to trace the full lineage from raw event to served recommendation. Building this traceability from the start takes weeks. Retrofitting it after you've been running in production for a year takes months and a lot of uncomfortable conversations about what data went where.

Activating Recommendations Across Your Stack

A recommendation that only works on your website is leaving value on the table. The same customer profile that powers web personalization should drive email content, mobile push timing, call center agent prompts, and paid media suppression.

This requires two things: a unified customer profile that's accessible across channels, and an integration layer that can activate recommendations in every system where customers interact with your brand.

The activation flow in practice: AudienceStream maintains the unified profile, continuously updated with real-time behavioral data. When the recommendation model produces results, those results (or the audience segments they inform) need to reach your email platform, your ad networks, your CMS, your mobile engagement tool, and your customer service platform. Tealium connects to 1,300+ downstream platforms, which means the activation layer isn't a custom integration project for every channel. The same recommendation signal can trigger a personalized email, suppress a retargeting ad (because the customer already converted), and surface relevant products on the next website visit.

Operationally, centralized audience management prevents the fragmentation that kills recommendation consistency. If your email team, web team, and paid media team are each building their own recommendation logic independently, customers get conflicting experiences. A single recommendation signal, activated across channels through a shared integration layer, keeps the experience coherent.

One pattern worth implementing early: next-best-action logic that considers channel context. A product recommendation delivered via mobile push at 2 AM is interruption, not personalization. The same recommendation surfaced when the customer opens your app the next morning is helpful. Channel-aware activation rules, informed by real-time behavioral signals, make the difference.

What's Next: Agentic AI and the Future of Recommendations

The recommendation architecture described in this playbook is already evolving. Three shifts are worth watching.

Agentic AI in recommendation workflows. AI agents that can autonomously query customer profiles, evaluate context, select recommendation strategies, and activate across channels are moving from research prototypes to early production deployments. These agents need real-time access to unified customer data with sub-second latency. The data infrastructure requirements are the same as (or more demanding than) traditional recommendation serving. Tealium's MCP integration is designed for exactly this pattern: giving AI agents real-time, consented access to customer context.

Generative AI for recommendation content. Instead of selecting from a fixed catalog of recommendation templates, generative models can produce personalized explanations, product descriptions, and offer copy tailored to individual customer context. This requires the same real-time feature pipeline described above, now feeding a generative model alongside (or instead of) a traditional ranking model.

Multi-modal recommendation signals. Voice interactions, video engagement patterns, and image-based search are creating new behavioral signals that recommendation models can consume. The pipeline architecture accommodates these signals as additional event types, but the enrichment and feature engineering layers need to handle unstructured data, typically through embedding models that convert multi-modal inputs into vector representations.

What stays constant across all three: you still need fresh, consented, unified customer data delivered in real time. The models and channels will keep changing. The data foundation won't become less important. It'll become more.

Build the Foundation Before the Model

Every AI recommendation conversation eventually focuses on the model. Which architecture. Which algorithm. Which framework.

Those choices matter. But they matter less than the data feeding the model. A sophisticated hybrid model running on stale, fragmented, non-consented data will underperform a simple collaborative filter running on fresh, unified, properly governed data. Every time.

The teams that get recommendation systems right don't start with the model. They start with the pipeline. They invest in real-time event capture, identity resolution at ingestion, in-flight enrichment, and a feature store fast enough to keep up with customer intent as it shifts. The model comes last because it's the easiest part to swap out. The data foundation is the part that compounds.

Recommendation models will keep getting better. The architecture described in this playbook will outlast any individual model generation. The hard problem has always been the same: can you get the right data to the right place before the moment passes? Teams that solve that problem will swap models in and out like tires. Teams that don't will keep blaming the algorithm.

REFERENCES

1. Redis. "Real-Time AI Recommendation Systems." redis.io/blog/real-time-ai-recommendation-systems.
2. Tinybird. "Real-Time Recommendation System." tinybird.co/blog/real-time-recommendation-system.
3. HumAI. "Real-Time AI Inference 2026: Complete Guide to Sub-100ms Models." humai.blog/real-time-ai-inference-2026.
4. DesignGurus. "How Would You Design a Real-Time Recommendation Engine Using Machine Learning." designgurus.io.
5. BrainForge. "How Netflix Uses Machine Learning to Create Perfect Recommendations." brainforge.ai/blog/how-netflix-uses-machine-learning.
6. Redis. "Real-Time AI Recommendation Systems." redis.io/blog/real-time-ai-recommendation-systems.
7. Catchpoint. "Semantic Caching: What We Measured, Why It Matters." catchpoint.com/blog/semantic-caching-what-we-measured-why-it-matters.
8. AWS. "Lower Cost and Latency for AI Using Amazon ElastiCache as a Semantic Cache with Amazon Bedrock." aws.amazon.com/blogs/database/lower-cost-and-latency-for-ai-using-amazon-elasticache.
9. Envive. "63 AI Personalization in eCommerce Lift Statistics." envive. ai/post/ai-personalization-in-ecommerce-lift-statistics.

About Tealium

Tealium helps companies collect, govern, and enrich their customer data in real-time to power AI initiatives and delight customers in the moments that matter. Tealium's turnkey integration ecosystem supports more than 1,300 built-in connections from the world's most prominent technology experts. Tealium's solutions include a real-time customer data platform (CDP) with intelligent AI data streaming, tag management, and an API hub. Tealium's data collection, management, and activation capabilities enable enterprises to accelerate operating performance, enhance customer experiences, drive better outcomes, and support global data compliance. More than 850 leading businesses globally trust Tealium to power their customer data strategies.

For more information, visit

tealium.com