Vectors In Orbit

Financial-Aware Collaborative Filtering for E-Commerce

Project Report and Executive Summary

January 2026

January 26, 2026

Contents

1 Problem Statement and Solution Idea

1.1 The Problem

Traditional e-commerce recommendation systems suffer from two critical blind spots:

1. **Budget Invisibility**: Recommendations ignore user financial constraints. A low-income user browsing for laptops is presented the same premium $5000 devices as a high-earning executive, resulting in frustration and abandoned carts.
2. **Financial Misalignment in Collaborative Filtering**: Standard CF treats all users as equivalent. Two users with vastly different purchasing power and income levels are considered neighbors if they share interactions, despite having fundamentally incompatible affordability profiles.
3. **Real-Time Responsiveness Gap**: Users expect immediate personalization. A purchase should instantly refine recommendations, yet most systems update recommendations on next-session basis (hours or days later).

1.2 The Solution: Financial-Aware Collaborative Filtering (FA-CF)

We introduce **Vectors In Orbit**, a recommendation engine that:

- **Respects Budget Constraints**: Excludes unaffordable products from collaborative signals. A user with $500 budget is never influenced by another user's $5000 purchases.
- **Aligns on Affordability**: Computes CF similarity not just by interaction overlap, but by affordability ratio alignment. Users who prefer ˜20% budget items are neighbors; users who prefer 5% items are not.
- **Enables Real-Time Feedback**: Self-interaction boost provides immediate score increases for recent high-intent actions (purchase, add-to-cart) with 1-day time decay.
- **Blends Five Signals**: Combines semantic relevance (40%), affordability (25%), user preferences (15%), collaborative filtering (15%), and popularity (5%) for balanced recommendations.

1.3 Key Innovation: Affordability Ratio

For each user-product interaction, we compute:

$$\rho = \frac{\text{product\_price}}{\text{available\_balance} + \text{credit\_limit}}$$

This dimensionless metric represents what fraction of total budget a product consumes. It enables:

- Budget-aligned neighbor selection
- Financial constraint validation before logging
- Real-time responsiveness to affordability shifts

2 System Architecture

2.1 High-Level Overview

The system is organized in three layers:

1. **Presentation Layer**: Streamlit UI with interaction hooks
2. **Business Logic Layer**: Search Pipeline, FA-CF Engine, Interaction Logger
3. **Data Layer**: Qdrant Cloud with 4 specialized collections

All components communicate asynchronously through the Qdrant API, enabling low-latency inference and real-time updates without blocking user interactions.

2.2 Core Components

Component	Function	File
Semantic Search	Vector similarity retrieval	`search_pipeline.py`
FA-CF Scoring	Affordability-aware CF	`cf/fa_cf.py`
Interaction Logging	Real-time, non-blocking writes	`interaction_logger.py`
Popularity Cache	5-min TTL trending system	`interaction_logger.py`
Product Reranking	5-signal blending	`search_pipeline.py`
Streamlit UI	User-facing interface	`app.py`

Table 1: Core components and responsibilities.

2.3 Qdrant Integration

All data is stored in **Qdrant Cloud**, a managed vector database:

1. **products_multimodal** (4000+ products)
   - Vector: 384-D SentenceTransformer embedding
   - Payload: product metadata (name, price, category, brand, in_stock, image_url)
   - Indexing: HNSW on vector + exact match on in_stock, product_id
2. **interaction_memory** (user behavior log)
   - Vector: behavioral text embedding or product vector
   - Payload: user_id, product_id, interaction_type, timestamp, financial context (balance, credit, affordability_ratio)
   - Indexing: HNSW on vector + range on timestamp + exact match on user_id
   - Write Strategy: Non-blocking upsert (wait=False) for latency optimization
3. **user_profiles** (1000 users)
   - Vector: preference embedding
   - Payload: name, location, risk_tolerance, preferred_categories, preferred_brands
4. **financial_contexts** (user budget profiles)
   - Payload: available_balance, credit_limit, current_debt, eligible_installments

2.4 Data Flow

1. User types search query ‘‘`laptop for coding`’’
2. Embed query: $\vec{q} =$ `SentenceTransformer`$(query)$ [4-5ms on GPU]
3. Semantic search: retrieve top 30 products from products_multimodal
4. Compute 5 ranking signals:
   - **Semantic**: $cosine\_similarity(\vec{q}, \text{product\_vectors})$
   - **Affordability**: $1 - \text{affordability\_ratio}$ (capped at 1)
   - **Preference**: category/brand alignment with user profile
   - **CF**: FA-CF scores from interaction_memory [150-200ms for neighbor search + scoring]
   - **Popularity**: trending products from 5-min cache [1-5ms cache hit]
5. Blend signals: $final = 0.40 \cdot sem + 0.25 \cdot aff + 0.15 \cdot cf + 0.15 \cdot pref + 0.05 \cdot pop$
6. Return top 10 ranked products
7. Log interaction asynchronously: `upsert_to_qdrant`(interaction_memory, wait=False)

End-to-End Latency: Typically 150-250ms (semantic + CF + reranking)

3 Data Pipeline

3.1 Data Sources

Real product data from four e-commerce platforms:

- Amazon: ˜2000 products
- Lazada: ˜1500 products
- Shein: ˜800 products
- Walmart: ˜700 products
- **Total**: ˜5000 products consolidated in `data/all_products_payload.json`

Each product includes: `id, name, description, price, categories, brand, in_stock, region, image_url`

3.2 Four-Stage Preprocessing

3.2.1 Stage 1: Product Embedding and Insertion

For each product:

- Concatenate: $text = name + description + categories + brand + region$
- Embed: $\vec{e} =$ `SentenceTransformer`$(text)$ [384-D vector]
- Batch encode: 5000 products in 50 seconds on GPU (batch_size=128)
- Insert to Qdrant: `upsert`(products_multimodal, {vector, payload})

3.2.2 Stage 2: User Profile Generation

Generate 1000 synthetic users **correlated to products**:

- Sample preferred categories/brands from real product distribution
- Assign risk_tolerance (Low, Medium, High) uniformly
- Insert to user_profiles collection

3.2.3 Stage 3: Financial Context Generation

For each user, derive budget from preferred products:

- Compute average price in preferred categories: $\bar{p}$
- Scale available_balance: $\bar{p} \times U(0.5, 2.0)$
- Scale credit_limit: $\bar{p} \times U(2.0, 5.0)$
- Ensures financial context matches product affinity
- Insert to financial_contexts collection

3.2.4 Stage 4: Interaction Generation

For each user, generate 2-3 correlated interactions:

- 80% from preferred categories, 20% random
- Sample interaction type: view (40%), click (30%), add_to_cart (20%), purchase (10%)
- Embed behavioral text: "user [verb] [product] [price] [query]"
- Insert to interaction_memory with financial context

3.3 Key Pipeline Features

Feature	**Benefit**
GPU-accelerated batch encoding	400x speedup vs. sequential
User-product correlation	Realistic co-purchase patterns
Budget scaling to products	Financial context matches user interests
Non-blocking Qdrant upserts	No blocking during inference

Table 2: Pipeline optimizations.

4 Project Timeline

4.1 Completed Work

✓ **Core FA-CF Algorithm**

- Affordability ratio computation
- Financial alignment scoring
- Budget-aware neighbor selection

✓ **Semantic Search Pipeline**

- SentenceTransformer integration (all-MiniLM-L6-v2, 384-D)
- Qdrant vector search with filtering
- GPU-accelerated inference

✓ **Real-Time Interaction Logging**

- Non-blocking writes to interaction_memory
- Financial context validation
- Time-decay for historical weighting (7-day half-life)
- Self-interaction boost (1-day half-life, intent-weighted)

✓ **Product Reranking**

- 5-signal blending (semantic 40%, affordability 25%, CF 15%, preference 15%, popularity 5%)
- Normalization and score combination
- Budget constraint enforcement

✓ **Popularity/Trending System**

- 5-minute cache with TTL
- Exponential time decay (6-hour half-life)
- Invalidation on high-intent interactions

✓ **Data Pipeline**

- Real product data loading (5000 products)
- Synthetic user generation (1000 users, correlated to products)
- Financial context scaling
- Interaction generation (2-3 per user)

✓ **Streamlit UI**

- User persona selection (Student, Professional, Executive, Custom)
- Search and filter interface
- Real-time interaction logging hooks (view, click, add-to-cart, purchase)
- Trending products display

✓ **Comprehensive Testing**

- Budget divergence test: expensive items zero-boosted for low-budget users
- Financial alignment test: similar affordability ratios increase CF scores
- Real-time interaction test: post-purchase score ¿ pre-purchase score
- CF comparison test: CF-enabled rankings visibly differ from semantic-only

✓ **Documentation**

- Full technical report (22 pages)
- Architecture diagrams
- API documentation
- Test validation results

4.2 Future Work (Roadmap)

4.2.1 Phase 2: Production Hardening

- **Redis Caching Layer**
  - Replace in-memory popularity cache with Redis
  - Enable shared caching across multiple backend instances
  - Implement cache invalidation patterns
- **A/B Testing Framework**
  - Experiment with signal weight variations
  - Track treatment/control metrics: CTR, conversion, revenue per user
  - Automated reporting dashboard
- **Monitoring and Observability**
  - Logging pipeline for all search requests
  - Latency tracking (p50, p95, p99)
  - Cache hit rate monitoring
  - Anomaly detection for budget divergence

4.2.2 Phase 3: Feature Expansion

- **Multi-Modal Search**
  - Integrate CLIP for image embeddings
  - Support image-based product search
  - Fuse text + image similarities: $0.6 \times text + 0.4 \times image$
- **Explainability Features**
  - Score breakdown for each recommendation
  - Human-readable explanations: "We recommend this because..."
  - Feature importance visualization
- **Cold-Start Handling**
  - Content-based fallback for new users
  - Popularity-heavy weighting for cold products
  - Explore-exploit framework (20% random, 80% best-predicted)
- **Buy-Now-Pay-Later (BNPL) Integration**
  - Extend affordability scoring for installment plans
  - Monthly income-based affordability: $\rho_{monthly} = payment/income$
  - Recommend products previously unaffordable via BNPL

4.2.3 Phase 4: Scale and Optimization

- **Distributed Training**
  - Fine-tune embeddings on transaction data
  - Improve semantic relevance for e-commerce domain
- **Deployment Scaling**
  - Kubernetes orchestration for multi-region deployment
  - Load balancing across multiple Streamlit instances
  - Qdrant sharding for datasets ¿1M products
- **Advanced Analytics**
  - Cohort analysis: how different user segments respond
  - Budget trajectory tracking: alert users to affordability shifts
  - Category affinity evolution over time

4.3 Known Limitations and Mitigation

Limitation	Impact	Mitigation
Cold-start for new users	Limited CF signal	Content-based + popularity fallback
Sparse interaction history	Weak neighbor signal	Increase explore phase in recommendations
Budget drift	Stale affordability data	Refresh financial context on session init
Popularity bias	Filter bubbles	Diversity constraint in final ranking

Table 3: Known limitations and mitigation strategies.

5 Key Metrics and Results

5.1 Performance Metrics

Metric	Value	Notes
Semantic search latency	4-5 ms	GPU-accelerated query embedding
CF scoring latency	150-200 ms	Neighbor search + affordability alignment
Popularity cache (hit)	1-5 ms	In-memory dict lookup
Popularity cache (miss)	150-200 ms	Full time-decay aggregation
End-to-end recommendation	150-250 ms	Search + reranking

Table 4: Latency metrics.

5.2 Test Results

All four FA-CF validation tests pass:

1. **Budget Divergence**: Premium items (price ¿ budget) receive zero CF boost. ✓PASS
2. **Financial Alignment**: Products with similar affordability ratios get boosted. ✓PASS
3. **Real-Time Interaction**: Post-purchase CF score ¿ pre-purchase score. ✓PASS
4. **CF Comparison**: CF-enabled rankings differ visibly from semantic-only. ✓PASS

5.3 Scalability

- 5000 products: embeddings in 50 seconds (batch encoding)
- 1000 users: interaction generation in 30 seconds
- Qdrant: supports millions of vectors with sub-millisecond retrieval
- Non-blocking writes: enables handling 100+ interactions/second

6 Conclusion

Vectors In Orbit successfully demonstrates a novel approach to budget-aware recommendations. By introducing affordability-aware collaborative filtering and real-time self-interaction boost, the system delivers personalized recommendations that respect user financial constraints while maintaining search relevance.

The project is production-ready for MVP deployment with clear roadmap for feature expansion and scaling. Future phases will focus on multi-modal search, explainability, and advanced analytics to further improve user trust and engagement.