Hanzo: An AI-First Commerce Platform

Zach Kelling
Hanzo Industries
zach@hanzo.ai

2017

Abstract

We present Hanzo, an AI-first commerce platform that integrates machine learning throughout the customer journey. Unlike traditional e-commerce systems that bolt AI capabilities onto existing architectures, Hanzo was designed from inception with artificial intelligence as the core organizing principle. The platform provides real-time personalized recommendations, intelligent search ranking, dynamic pricing optimization, automated fraud detection, and predictive inventory management through a unified AI infrastructure. We describe the system architecture, detail the machine learning models powering each capability, and evaluate performance across production deployments. Results demonstrate significant improvements: 28% increase in average order value through personalization, 45% reduction in fraud losses, and 18% improvement in inventory turnover. Hanzo represents a new paradigm for commerce platforms where AI is not an add-on but the foundational layer.

1 Introduction

The history of e-commerce platforms follows a pattern of accretion. Systems designed for catalog management acquire shopping carts, then payment processing, then analytics, then—belatedly—AI capabilities. Each addition creates integration complexity while the core architecture, optimized for earlier requirements, constrains what AI can achieve.

Hanzo inverts this pattern. We designed an AI-first commerce platform where machine learning models are not peripheral services but central infrastructure. Every customer interaction—search, browse, cart, checkout—flows through AI systems that personalize, predict, and optimize in real time.

This architectural decision yields compounding benefits:

1. **Unified data model**: All events feed into a shared feature store, enabling cross-functional learning.
2. **Consistent inference**: A single model serving infrastructure ensures low-latency predictions across all surfaces.
3. **Rapid iteration**: Standardized ML pipelines accelerate model development and deployment.
4. **Emergent intelligence**: Models learn from each other's predictions, creating feedback loops that improve system-wide performance.

Our contributions are:

1. An AI-first architecture for commerce platforms with unified feature stores and model serving.
2. Production implementations of personalization, search ranking, fraud detection, and dynamic pricing.
3. Empirical evaluation demonstrating significant improvements across key commerce metrics.
4. Open discussion of challenges and lessons learned in deploying AI at commerce scale.

2 Background

2.1 E-Commerce AI Evolution

AI in e-commerce has evolved through three generations:

First generation (1995–2005): Rule-based systems. Simple heuristics drive recommendations ("customers who bought X also bought Y"), fraud rules (velocity checks, blacklists), and pricing (cost-plus margins).

Second generation (2005–2015): Specialized ML. Machine learning models improve specific functions. Amazon's item-to-item collaborative filtering [1] revolutionized recommendations. PayPal deployed neural networks for fraud detection [2]. These systems operate independently with limited cross-functional learning.

Third generation (2015–present): Integrated AI. Modern systems unify ML infrastructure across functions. Alibaba's AI platform [3] demonstrates system-wide intelligence. Hanzo extends this approach to a platform available to any merchant.

2.2 Machine Learning Infrastructure

Production ML requires infrastructure beyond model training:

- **Feature stores**: Centralized repositories for feature computation and serving.
- **Model registries**: Version control for trained models.
- **Serving infrastructure**: Low-latency prediction at scale.
- **Monitoring**: Drift detection and performance tracking.

Hanzo builds upon emerging standards (Feast, MLflow, TensorFlow Serving) while adding commerce-specific extensions.

3 System Architecture

3.1 Design Principles

Hanzo adheres to four architectural principles:

Principle 1: Events as First-Class Citizens. Every user action, system event, and model prediction is an event in a unified stream. Events are immutable, ordered, and available for both real-time and batch processing.

Principle 2: Shared Feature Store. All models draw features from a centralized store. This ensures consistency (the same feature computation serves training and inference) and enables feature reuse across models.

Principle 3: Online-Offline Parity. Models train on historical data and serve in real-time. Infrastructure ensures the same feature transformations apply in both contexts, eliminating training-serving skew.

Principle 4: Continuous Learning. Models update continuously as new data arrives. Feedback loops from predictions to outcomes enable rapid adaptation.

3.2 Core Components

3.2.1 Event Stream

All platform activity flows through Apache Kafka:

Listing 1: Event schema

```
@dataclass
class Event:
    event_id: str
    event_type: str
    timestamp: datetime
    user_id: Optional[str]
    session_id: str
    payload: Dict[str, Any]
    context: Dict[str, Any]  #
        device, location, etc.
```

Event types include:

- `page_view`: User views a page.
- `product_view`: User views a product.
- `add_to_cart`: User adds item to cart.
- `purchase`: Transaction completed.
- `recommendation_shown`: Model prediction served.

- `recommendation_clicked`: User engaged with recommendation.

3.2.2 Feature Store

The feature store provides:

Feature computation. Declarative feature definitions:

Listing 2: Feature definition

```
@feature
def user_purchase_count_7d(user_id:
    str) -> int:
    """Count␣of␣purchases␣in␣last␣7␣
       days."""
    return events.filter(
        event_type="purchase",
        user_id=user_id,
        timestamp__gte=now() - days
           (7)
    ).count()

@feature
def product_view_to_purchase_rate(
    product_id: str) -> float:
    """Conversion␣rate␣from␣view␣to␣
       purchase."""
    views = events.filter(event_type
       ="product_view", product_id=
       product_id).count()
    purchases = events.filter(
       event_type="purchase",
       product_id=product_id).count
       ()
    return purchases / max(views, 1)
```

Online serving. Low-latency feature retrieval (p99 $<$ 5ms):

Listing 3: Feature retrieval

```
features = feature_store.
    get_features(
    feature_names=["
       user_purchase_count_7d", "
       user_avg_order_value"],
    entity_ids={"user_id": "user_123
       "}
)
```

Offline training. Point-in-time correct feature joins for training data:

Listing 4: Training data generation

```
training_data = feature_store.
    get_historical_features(
    feature_names=["
       user_purchase_count_7d", "
       product_popularity"],
    entity_df=labels_df,  # Contains
        user_id, product_id,
       timestamp, label
)
```

3.2.3 Model Serving

Models deploy to a unified serving layer:

Listing 5: Model serving

```
class RecommendationModel:
    def __init__(self, model_path:
       str):
        self.model = tf.saved_model.
           load(model_path)
        self.feature_spec =
           load_feature_spec(
           model_path)

    def predict(self, user_id: str,
       candidate_ids: List[str]) ->
       List[float]:
        features = feature_store.
           get_features(
            feature_names=self.
               feature_spec,
            entity_ids={"user_id":
               user_id, "product_ids
               ": candidate_ids}
        )
        return self.model.predict(
           features)
```

3.3 AI Capabilities

3.3.1 Personalized Recommendations

Recommendations power multiple surfaces: homepage, product pages, cart, and email.

Model architecture. We employ a two-tower neural network [4]:

$$u = f_{\text{user}}(x_{\text{user}}) \tag{1}$$

$$v = f_{\text{item}}(x_{\text{item}}) \tag{2}$$

$$\text{score} = u \cdot v \tag{3}$$

The user tower f_{user} encodes user features (browsing history, purchase history, demographics). The item tower f_{item} encodes product features (category, price, description embeddings). Dot product scoring enables efficient retrieval via approximate nearest neighbors.

Features. Key features include:

- User: Recent views (sequence), purchase history, session duration, device type.
- Item: Category hierarchy, price percentile, description embedding, popularity.
- Context: Time of day, day of week, referral source.

Training. The model trains on implicit feedback:

- Positive: Purchases, add-to-carts, extended views ($>$ 30 seconds).
- Negative: Sampled non-interactions, quick bounces.

We use batch negatives with in-batch sampling for efficient training.

3.3.2 Intelligent Search

Search ranking combines text relevance with personalization.

Architecture. A learning-to-rank model [5] re-ranks candidates from Elasticsearch:

1. **Retrieval**: Elasticsearch returns top 1000 candidates by BM25.
2. **Scoring**: Neural ranker scores candidates using user context.
3. **Re-ranking**: Final ordering by learned scores.

Model. We use a cross-attention transformer:

Listing 6: Search ranking model

```
class SearchRanker(nn.Module):
    def __init__(self, config):
        self.query_encoder =
            TransformerEncoder(config
            )
        self.product_encoder =
            TransformerEncoder(config
            )
        self.cross_attention =
            CrossAttention(config)
        self.scorer = nn.Linear(
            config.hidden_size, 1)

    def forward(self, query_tokens,
        product_features,
        user_features):
        query_emb = self.
            query_encoder(
            query_tokens)
        product_emb = self.
            product_encoder(
            product_features)
        cross_emb = self.
            cross_attention(query_emb
            , product_emb,
            user_features)
        return self.scorer(cross_emb
            )
```

Training data. Click-through logs provide training signal:

- Queries with clicked results (positive).
- Queries with skipped results (negative).
- Purchase after search (strong positive).

3.3.3 Fraud Detection

Fraud detection operates at multiple stages: account creation, checkout, and post-transaction.

Model architecture. An ensemble of gradient boosted trees (XGBoost) and neural networks:

$$\text{fraud\_score} = \alpha \cdot \text{XGB}(x) + (1-\alpha) \cdot \text{NN}(x) \quad (4)$$

XGBoost captures interpretable rules; the neural network captures complex patterns.

Features. Fraud features span multiple categories:

- **Velocity**: Orders per hour, distinct cards per user, shipping addresses per card.
- **Device**: Device fingerprint, IP geolocation, browser anomalies.

- **Behavioral**: Time on site, navigation patterns, form fill speed.
- **Network**: Graph features connecting users, devices, and addresses.

Decision engine. Fraud scores trigger actions:

- Score < 0.3: Approve automatically.
- $0.3 \leq$ Score < 0.7: Additional verification (3D Secure, phone verification).
- Score ≥ 0.7: Manual review or automatic decline.

3.3.4 Dynamic Pricing

Dynamic pricing optimizes prices in real-time based on demand, competition, and inventory.

Model. A demand model predicts quantity sold at each price point:

$$q(p) = \exp(\beta_0 + \beta_1 \log p + \beta_2 x) \tag{5}$$

where p is price and x are contextual features (day of week, competitor prices, inventory level).

Optimization. Given demand model, optimize for profit:

$$p^* = \arg\max_p (p - c) \cdot q(p) \tag{6}$$

where c is cost. Constraints include:

- Minimum margin: $p \geq (1 + m) \cdot c$.
- Price consistency: $|p - p_{\text{prev}}| \leq \delta$.
- Competitor parity: $p \leq \max(\text{competitor prices}) + \epsilon$.

3.3.5 Inventory Optimization

Predictive inventory management reduces stockouts and overstock.

Demand forecasting. A hierarchical time series model [6] predicts demand:

$$y_{t+h} = \text{trend}_t + \text{season}_t + \text{promo}_t + \epsilon_t \tag{7}$$

The model forecasts at multiple aggregation levels (SKU, category, store) with reconciliation ensuring consistency.

Reorder optimization. Given demand forecast and lead times:

$$\text{reorder\_point} = \mu_L + z_\alpha \cdot \sigma_L \tag{8}$$

where μ_L is expected demand during lead time, σ_L is demand standard deviation, and z_α is the safety stock factor for target service level α.

4 Implementation

4.1 Technology Stack

- **Event streaming**: Apache Kafka.
- **Feature store**: Custom implementation on Redis (online) + BigQuery (offline).
- **Model training**: TensorFlow, XGBoost.
- **Model serving**: TensorFlow Serving, custom Python services.
- **Orchestration**: Kubernetes, Airflow.

4.2 Latency Requirements

Commerce AI must meet strict latency requirements:

Table 1: Latency requirements by surface

Surface	p50 (ms)	p99 (ms)
Search ranking	20	50
Recommendations	15	40
Fraud scoring	100	250
Dynamic pricing	50	150

We achieve these through:

- Feature caching with 1-minute TTL.
- Batch prediction for non-real-time surfaces.
- Model quantization and optimization.
- Geographic distribution (models deployed to edge).

4.3 Model Updates

Models update through a continuous training pipeline:

1. New events flow to training data store.
2. Nightly jobs retrain models on recent data.
3. Shadow deployment compares new model to production.
4. Automatic promotion if metrics improve.
5. Rollback if degradation detected.

Listing 7: Model promotion logic

```
def should_promote(shadow_metrics,
    prod_metrics):
    # Require improvement on primary
        metric
    if shadow_metrics.
        conversion_rate <
        prod_metrics.conversion_rate:
        return False

    # Guard rails on secondary
        metrics
    if shadow_metrics.latency_p99 >
        prod_metrics.latency_p99 *
        1.1:
        return False

    # Statistical significance
    if not is_significant(
        shadow_metrics, prod_metrics,
        alpha=0.05):
        return False

    return True
```

5 Evaluation

5.1 Experimental Setup

We evaluated Hanzo across three production deployments:

- **Fashion retailer**: 2M monthly active users, 50K SKUs.
- **Electronics store**: 500K MAU, 10K SKUs.
- **Marketplace**: 1M MAU, 200K SKUs.

Each deployment ran for 6 months. We measured against pre-Hanzo baselines.

5.2 Recommendation Performance

Table 2: Recommendation metrics improvement

Metric	Fashion	Electronics	Marketplace
CTR	+45%	+38%	+52%
Conversion	+22%	+18%	+31%
AOV	+28%	+24%	+32%

Recommendations drive significant revenue improvement. Average order value increases as recommendations surface complementary products.

5.3 Search Quality

Table 3: Search metrics improvement

Metric	Fashion	Electronics	Marketplace
NDCG@10	+35%	+28%	+42%
Zero-result rate	-62%	-55%	-71%
Search-to-purchase	+24%	+19%	+28%

Intelligent ranking improves search relevance. Query understanding reduces zero-result searches through spelling correction and synonym expansion.

5.4 Fraud Detection

Table 4: Fraud metrics

Metric	Fashion	Electronics	Marketplace
Fraud loss reduction	-42%	-51%	-45%
False positive rate	-35%	-28%	-38%
Manual review rate	-55%	-48%	-52%

Improved fraud detection reduces both losses and operational burden. Lower false positive rates mean fewer legitimate customers face friction.

Table 5: Inventory metrics improvement

Metric	Fashion	Electronics	Marketplace
Stockout rate	-38%	-42%	-35%
Inventory turnover	+18%	+22%	+15%
Markdown rate	-25%	-18%	-28%

5.5 Inventory Optimization

Better demand forecasting reduces both stockouts (lost sales) and overstock (markdowns).

5.6 System Performance

Table 6: Production latency (ms)

Component	p50	p99
Feature retrieval	2	8
Recommendation inference	12	35
Search re-ranking	18	45
Fraud scoring	45	180

All components meet latency requirements in production.

6 Lessons Learned

6.1 Data Quality Trumps Model Complexity

Early iterations focused on sophisticated model architectures. We found greater improvements from:

- Fixing data pipeline bugs.
- Adding missing features (device type had been dropped).
- Correcting label noise (returns flagged as purchases).

Simple models on clean data outperformed complex models on noisy data.

6.2 Feedback Loops Require Careful Design

Recommendations create feedback loops: recommended items get more views, which makes them appear more popular, which increases recommendation probability. We address this through:

- Exploration-exploitation balancing.
- Position debiasing in training.
- Diversity constraints in serving.

6.3 Explain ability Builds Trust

Merchants hesitated to trust AI pricing and fraud decisions. Adding explanations ("Price increased due to low inventory and high demand") increased adoption from 34% to 78% of eligible merchants.

7 Related Work

Amazon's personalization [1] pioneered item-based collaborative filtering. Netflix Prize [7] advanced matrix factorization methods. YouTube's recommendation system [4] introduced deep learning at scale. Hanzo synthesizes these advances into an integrated platform.

Alibaba's AI platform [3] demonstrates similar system-wide integration. Our contribution extends these capabilities to a platform available to any merchant, not just large enterprises.

8 Conclusion

Hanzo demonstrates that AI-first architecture yields significant improvements across commerce metrics. By designing machine learning into the platform foundation rather than bolting it on, we achieve:

- 28% average order value increase.
- 45% fraud loss reduction.
- 18% inventory turnover improvement.

The key insight is architectural: unified event streams, shared feature stores, and consistent model serving create compounding benefits impossible with siloed AI additions.

Future work will extend Hanzo with conversational commerce (chatbots with full platform context), visual search (find similar products from images), and automated merchandising (AI-generated product descriptions and imagery).

References

[1] G. Linden, B. Smith, and J. York, "Amazon.com Recommendations: Item-to-Item Collaborative Filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[2] PayPal Engineering, "Fraud Detection with Machine Learning," *PayPal Engineering Blog*, 2013.

[3] Alibaba Group, "Artificial Intelligence at Alibaba," *Alibaba Tech Blog*, 2017.

[4] P. Covington, J. Adams, and E. Sargin, "Deep Neural Networks for YouTube Recommendations," *RecSys*, 2016.

[5] T.-Y. Liu, "Learning to Rank for Information Retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

[6] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, OTexts, 2011.

[7] J. Bennett and S. Lanning, "The Netflix Prize," *KDD Cup and Workshop*, 2007.