Review Engine - Autonomous AI Business Intelligence Platform

> **Enterprise-grade, LangGraph-powered multi-agent system that transforms customer reviews, e-commerce operations, and marketing workflows into actionable intelligence.**

Table of Contents

What Makes This Different

Traditional Approach	Our LangGraph Approach
Microservices + task queues	Multi-agent state machines
Cloud-only LLMs ($$$)	Hybrid: 85% on-prem, 15% cloud
Fragmented dashboards	Qrvey unified intelligence hub
Monolithic features	Modular plugins (reviews, ecommerce, loyalty)
Manual workflows	Autonomous agent orchestration

Core Capabilities

Business Modules (Plugin Architecture)

- **Review Intelligence**: Multi-platform aggregation (Google, Yelp, Facebook, Synup), sentiment analysis, geospatial clustering (H3 hexagons)

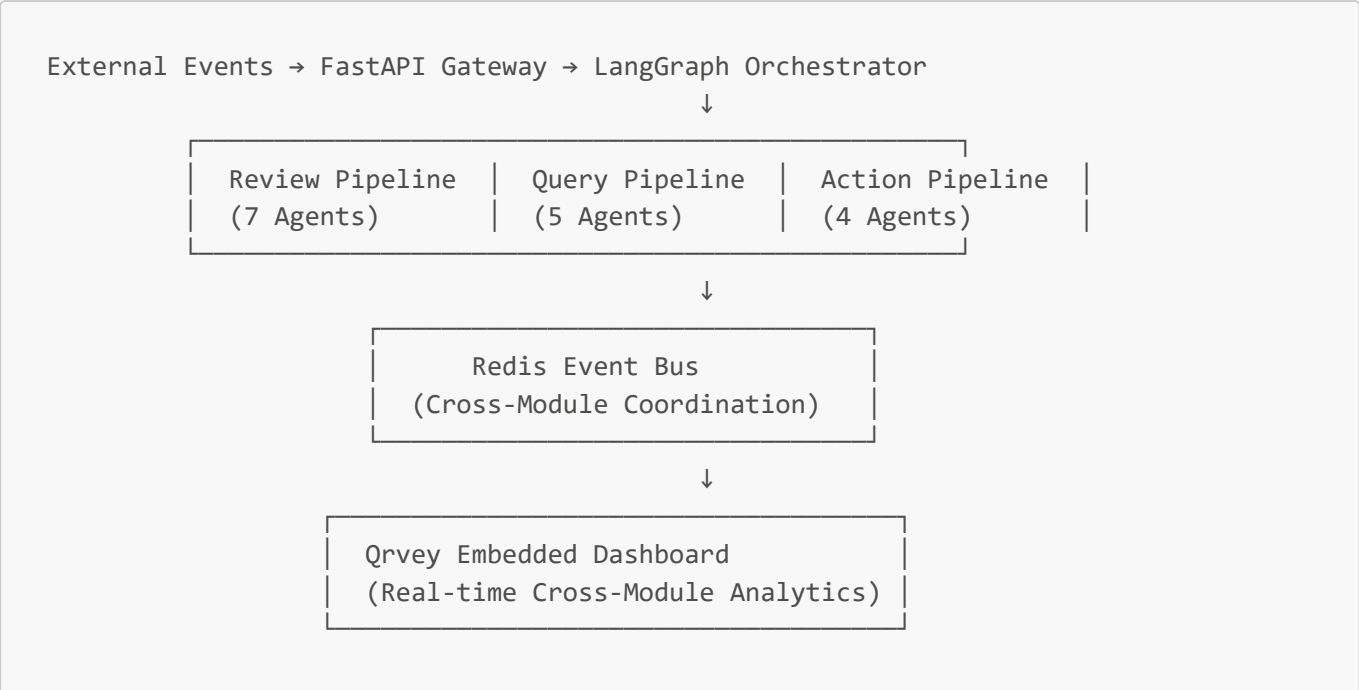
- 🛒 **E-Commerce Operations**: Product catalog management, order processing, inventory tracking, purchase analytics, customer LTV calculation
- 🎁 **Loyalty Automation**: Yotpo integration, automatic point awards based on reviews/purchases, referral program management
- 📧 **Marketing Orchestration**: Klaviyo campaign triggers, GDPR-compliant workflows, automated follow-up sequences
- 📱 **Mobile Apps**: Native iOS/Android applications, offline-first architecture, push notifications, biometric authentication

AI Infrastructure

- 🤖 **LangGraph Orchestration**: 14+ autonomous agents coordinating via state machines and event streams
- 💰 **Cost-Optimized LLM**: 85% on-prem inference (Ollama → vLLM), 15% cloud (Claude/GPT) for complex cases
- 🔍 **RAG Intelligence**: Semantic + SQL + geospatial hybrid retrieval for contextual analysis
- 📊 **Qrvey Dashboard**: Embedded SaaS dashboard aggregating insights across all business modules
- 🔒 **Enterprise Security**: Multi-tenant isolation, row-level security, RBAC, GDPR compliance

---

🏗️ Architecture at a Glance



💡 Business Value Proposition

For Product Teams:

- Ship new features as independent agent modules
- No monolithic refactoring required
- Feature flags enable per-tenant customization

For Finance Teams:

- 75% LLM cost reduction via hybrid on-prem/cloud strategy
- Predictable infrastructure costs
- Clear ROI metrics in Qrvey dashboards

For Customers:

- Single unified dashboard (Qrvey) across all business data
- Real-time insights from reviews, sales, loyalty programs
- White-label branding for multi-tenant deployments

🚀 5-Minute Quick Start

Prerequisites

```
System Requirements:
  - Docker Desktop (20.10+)
  - Python 3.11+
  - Node.js 18+
  - 8GB RAM (16GB recommended for vLLM)
  - Git
```

Step 1: Clone & Start Infrastructure (2 minutes)

```
git clone https://github.com/YourOrg/Review-Engine-Workshop-v2.git
cd Review-Engine-Workshop-v2

# Start core services (PostgreSQL, Redis, Qdrant, Ollama)
docker-compose up -d

# Verify services
docker-compose ps
# Expected: postgres, redis, qdrant, ollama (all healthy)
```

Step 2: Start Backend (2 minutes)

```
# Install dependencies
poetry install

# Run database migrations
poetry run alembic upgrade head

# Set environment variables
cp .env.example .env
# Edit .env with your API keys (Yotpo, Klaviyo, Synup)

# Start LangGraph orchestrator + FastAPI
```

```
poetry run python -m agents.main
# API available at http://localhost:8000
```

Step 3: Start Frontend (1 minute)

```
# New terminal window
cd review-engine-ui
npm install
npm run dev
# Dashboard available at http://localhost:3000
```

Step 4: First API Call

```
# Ingest a test review
curl -X POST http://localhost:8000/api/reviews/ingest \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_DEV_TOKEN" \
  -d '{
    "source": "google",
    "rating": 5,
    "text": "Amazing service! The team went above and beyond.",
    "customer_email": "test@example.com",
    "location": {"lat": 40.7128, "lng": -74.0060}
  }'
```

Expected Result (within 5 seconds):

1. ☑ Review appears in Qrvey dashboard
2. ☑ Sentiment analyzed (on-prem Ollama): 4.8/5
3. ☑ Themes extracted: ["service quality", "team performance"]
4. ☑ H3 geospatial index assigned
5. ☑ Loyalty eligibility checked
6. ☑ (If eligible) Points awarded via Yotpo + email sent via Klaviyo

Access Points

Service	URL	Purpose
Qrvey Dashboard	http://localhost:3000/dashboard	Unified business intelligence
API Docs	http://localhost:8000/docs	Interactive Swagger UI
LangSmith Traces	Set `LANGSMITH_API_KEY` in .env	Agent execution debugging
Qdrant Console	http://localhost:6333/dashboard	Vector database browser

---

🤖 LangGraph Multi-Agent System

Architectural Philosophy

This is NOT a microservices architecture with "agent" labels.

This platform is built on **LangGraph state machines** where each business capability is an autonomous agent that:

- Maintains its own state schema (TypedDict)
- Communicates via shared state objects and event streams
- Makes conditional routing decisions based on business logic
- Checkpoints progress to PostgreSQL for fault tolerance
- Publishes events to trigger downstream agent workflows

Agent Catalog

Agent Module	Responsibility	Input State	Output State	Tools
Ingestion	Pull reviews from Synup/Google/Yelp	`source`, `external_id`	`raw_review`	HTTP clients
Validation	Schema validation, dedup detection	`raw_review`	`is_valid`, `is_duplicate`	Pydantic, hash
Embedding	Generate 768d semantic vectors	`review_text`	`embedding`	nomic-embed
Model Router	Route to Ollama vs Cloud LLM	`review_text`	`model_choice`	Complexity classifier
Sentiment	Extract sentiment + themes	`review_text`	`sentiment_score`, `themes[]`	Llama 3.1 8B
Geo-Clustering	Assign H3 hexagon index	`latitude`, `longitude`	`h3_index`	H3 library
Query Understanding	Parse natural language queries	`user_query`	`query_intent`, `filters`	NLU model
Vector Search	Semantic similarity search	`query_embedding`, `filters`	`similar_reviews[]`	Qdrant client
SQL Agent	Convert filters to SQL queries	`structured_filters`	`sql_results`	PostgreSQL
H3 Geo Agent	Spatial proximity search	`h3_index`, `radius`	`nearby_reviews[]`	H3 KNN
RAG Generator	Context-aware answer generation	`context[]`, `query`	`answer`, `citations[]`	LLM + templates
Product Sync	Sync product catalog from Shopify/WooCommerce	`integration_config`	`products[]`	Shopify API

Agent Module	Responsibility	Input State	Output State	Tools
Inventory Manager	Track stock levels, low-stock alerts	`product_id`, `quantity`	`inventory_status`	Database
Order Processing	Process orders, update fulfillment	`order_data`	`order_status`	E-commerce APIs
Purchase Tracker	Log customer purchases for analytics	`customer_id`, `order_id`	`purchase_record`	Database
LTV Calculator	Calculate customer lifetime value	`customer_id`	`ltv_score`	Aggregation logic
Eligibility	Check loyalty program rules	`review_data`, `customer_data`	`is_eligible`, `reason`	Business rules
Loyalty	Award points via Yotpo API	`customer_id`, `points`	`points_awarded`	Yotpo client
Marketing	Trigger Klaviyo campaigns	`customer_id`, `campaign_id`	`campaign_sent`	Klaviyo client
Notification	Multi-channel alerts (email, SMS, push)	`notification_request`	`delivery_status`	Twilio, Expo

Data Flow Example: High-Rated Review → Loyalty Reward

```
Step 1: External Webhook
  Event: Customer leaves 5★ review on Google
  Synup webhook → POST /api/reviews/webhook
  FastAPI validates signature → Routes to Orchestrator

Step 2: Review Processing Pipeline (Sequential Graph)
  Node 1 - Ingestion Agent:
    Input: {external_id: "google_12345", source: "google"}
    Action: Fetch full review from Synup API
    Output: {raw_review: {...}, rating: 5, text: "Best pizza in NYC!"}

  Node 2 - Validation Agent:
    Input: {raw_review: {...}}
    Action: Check schema, compute hash, check duplicates
    Output: {is_valid: true, is_duplicate: false}

  Node 3 - Embedding Agent:
    Input: {text: "Best pizza in NYC!"}
    Action: Call Ollama nomic-embed → Generate 768d vector
    Output: {embedding: [0.123, -0.456, ...]}
    Side Effect: Store in Qdrant vector DB

  Node 4 - Model Router (Conditional Edge):
    Input: {text: "Best pizza in NYC!"}
    Action: Complexity score = 0.15 (simple sentiment)
```

```
    Decision: Route to Ollama (on-prem)
    Output: {model_choice: "ollama/llama3.1:8b"}

  Node 5 - Sentiment Agent (Ollama):
    Input: {text: "Best pizza in NYC!", model: "ollama/llama3.1:8b"}
    Action: LLM inference via LiteLLM
    Prompt: "Extract sentiment (0-5) and themes from this review: ..."
    Output: {sentiment_score: 4.9, themes: ["food quality", "location"]}

  Node 6 - Geo-Clustering Agent:
    Input: {latitude: 40.7589, longitude: -73.9851}
    Action: Convert to H3 index (resolution 9)
    Output: {h3_index: "892a1072b4bffff"}

  Node 7 - Publish Event:
    Action: Publish to Redis Streams
    Event: review_analyzed {
      review_id: "uuid-123",
      rating: 5,
      sentiment: 4.9,
      h3_index: "892a1072b4bffff",
      customer_email: "john@example.com"
    }

Step 3: Event Bus Routing
  Redis Stream Consumer: automation_pipeline_consumer
  Filter: rating >= 4.5 AND sentiment >= 4.0
  Match: TRUE → Trigger Automation Pipeline

Step 4: Automation Pipeline (Parallel + Sequential)
  Node 1 - Eligibility Agent:
    Input: {review_id: "uuid-123", customer_email: "john@example.com"}
    Action: Check business rules
      - Customer has active account? YES
      - Customer opted into loyalty? YES
      - Review not fraudulent? YES
    Output: {is_eligible: true, points_to_award: 100}

  Node 2 - Loyalty Agent:
    Input: {customer_email: "john@example.com", points: 100}
    Action: POST to Yotpo API /v2/customers/{id}/points
    Output: {points_awarded: 100, new_balance: 350}

  Node 3 - Marketing Agent (Parallel):
    Input: {customer_email: "john@example.com", trigger: "5_star_review"}
    Action: POST to Klaviyo API /v2/event-triggers
    Payload: {event: "Review Submitted", profile: {...}}
    Output: {campaign_sent: true, campaign_id: "abc123"}

  Node 4 - Notification Agent (Parallel):
    Input: {customer_email: "john@example.com", type: "loyalty_points"}
    Action: Send multi-channel notifications
      - Email: "You earned 100 points!"
      - Push notification (if mobile app installed)
```

```
    Output: {email_sent: true, push_sent: true}

Step 5: Qrvey Dashboard Update
  Action: POST to Qrvey Data Feed API
  Payload: {
    module: "reviews",
    metrics: {
      new_review: 1,
      avg_sentiment: 4.9,
      loyalty_triggered: 1,
      h3_index: "892a1072b4bffff"
    },
    timestamp: "2024-12-26T10:30:00Z"
  }
  Result: Qrvey dashboard chart updates in real-time (WebSocket)

Total Execution Time: 3.2 seconds (end-to-end)
```

State Management & Checkpointing

PostgreSQL-Backed Checkpointing:

```python
from langgraph.checkpoint.postgres import PostgresSaver

# Every node execution auto-saves state
checkpointer = PostgresSaver(
    connection_string=DATABASE_URL,
    # Stores: state snapshot, node name, timestamp, parent checkpoint
)

app = review_graph.compile(checkpointer=checkpointer)

# Benefits:
# 1. Fault Tolerance: Resume from last successful node
# 2. Time-Travel Debugging: Replay agent decisions
# 3. Human-in-the-Loop: Pause workflow for approval
# 4. Audit Trail: Complete state history for compliance
```

State Schema Example:

```python
from typing import TypedDict, Optional, Literal

class ReviewState(TypedDict):
    # Input
    review_id: str
    source: Literal["google", "yelp", "facebook", "synup"]
    raw_review: Optional[dict]

    # Validation
    is_valid: Optional[bool]
```

```
    is_duplicate: Optional[bool]

    # Analysis
    embedding: Optional[list[float]]
    sentiment_score: Optional[float]
    themes: Optional[list[str]]
    h3_index: Optional[str]

    # Routing
    model_choice: Optional[str]
    complexity_score: Optional[float]

    # Actions
    loyalty_eligible: Optional[bool]
    points_awarded: Optional[int]
    campaign_sent: Optional[bool]

    # Metadata
    checkpoints: list[str]  # ["ingestion", "validation", ...]
    created_at: str
    updated_at: str
```

🛠 Technology Stack

Core Orchestration Layer

Component	Technology	Version	Purpose
Agent Framework	LangGraph	0.2+	Multi-agent state machines
State Persistence	PostgreSQL	16+	Checkpointing, data storage
Event Bus	Redis Streams	7.4+	Inter-agent pub/sub
Observability	LangSmith	Latest	Agent tracing, debugging

Model Serving Infrastructure (Phased Approach)

Phase 1 (Weeks 1-2): Ollama CPU - MVP

Component	Technology	Purpose
CPU Inference	Ollama 0.3+	Llama 3.1 8B (quantized Q4_0)
Embeddings	nomic-embed-text-v1.5	768d vectors
Performance	~2-5 seconds/request	Sufficient for MVP

Phase 2 (Week 3+): vLLM GPU - Production

Component	Technology	Purpose
GPU Inference	vLLM 0.6+	Llama 3.1 8B/70B (AWQ 4-bit)

Component	Technology	Purpose
Model Gateway	LiteLLM 1.50+	Unified API (Ollama + vLLM + Cloud)
Cloud Fallback	Claude 3.5 / GPT-4o	15% of complex requests
Performance	~200-500ms/request	Production-grade

Cost Comparison:

```
10,000 Monthly Requests:
  Cloud-Only (GPT-4):
    Cost: $300/month

  Hybrid (85% Ollama/vLLM + 15% Claude):
    Ollama/vLLM: $0 (after $200 GPU amortization)
    Claude: $45/month (1,500 requests)
    Total: $45/month + $200 one-time

  Monthly Savings: 85% ($255/month)
  Break-even: Month 1
```

Data & Storage Layer

Component	Technology	Purpose
Primary Database	PostgreSQL 16 + pgvector	Reviews, customers, orders, checkpoints
Vector Store	Qdrant 1.11+	Semantic search (768d embeddings)
Cache	Redis 7.4+	LLM response cache, rate limits, sessions
Geospatial	H3 (Uber)	Hexagonal hierarchical indexing

Why Qdrant over pgvector alone?

- 10x faster similarity search (HNSW index)
- Advanced filtering (metadata + vector)
- Horizontal scaling for >1M vectors

Backend API Layer

Component	Technology	Purpose
API Gateway	FastAPI 0.115+	Thin REST layer, routes to agents
Validation	Pydantic 2.9+	Schema validation, type safety
Async HTTP	httpx	External API calls (Yotpo, Klaviyo, Shopify)
Logging	structlog	JSON structured logs

Frontend & Dashboard

Component	Technology	Purpose
Unified Dashboard	Qrvey (SaaS Embedded)	Cross-module business intelligence
Web Framework	Next.js 15	Module UIs, white-label theming
UI Components	shadcn/ui	Consistent design system
State Management	TanStack Query v5	Server state caching
Charts	Recharts 2+	Custom visualizations

Qrvey Integration:

```
// Embedded via SDK
<QrveyEmbed
  workspaceId="your-workspace-id"
  dashboardId="unified-dashboard"
  token={qrveyAuthToken}
  dataFeeds={[
    { module: "reviews", endpoint: "/api/qrvey/reviews" },
    { module: "ecommerce", endpoint: "/api/qrvey/ecommerce" },
    { module: "loyalty", endpoint: "/api/qrvey/loyalty" }
  ]}
  realtime={true}  // WebSocket updates
/>
```

Mobile Apps

Component	Technology	Purpose
Framework	Expo 52 (React Native 0.76)	iOS + Android native apps
Offline Database	WatermelonDB 0.26+	Local-first data sync
Notifications	Expo Push Notifications	Real-time alerts
Auth	Expo AuthSession + Supabase	Biometric login

External Integrations

Category	Provider	Purpose
Loyalty	Yotpo Loyalty & Referrals	Point awards, referrals
Marketing	Klaviyo	Email campaigns, flows
Review Aggregation	Synup, Google My Business, Yelp	Multi-platform review sync
E-Commerce	Shopify, WooCommerce	Product catalog, orders, inventory

What We Eliminated (Bloat Removal)

❌ **Celery** → Replaced by LangGraph workflows (no separate task queue) ❌ **Auth0** → Using Supabase Auth (simpler, 1/3 the cost) ❌ **Datadog** → LangSmith provides agent-native observability ❌ **Sentry** → LangSmith error tracking sufficient for agents ❌ **Multiple Embedding Providers** → Just nomic-embed (on-prem, 768d, free)

Result: 40% fewer dependencies, simpler architecture, lower costs

---

Module Plugin System

Module Registry

```
# agents/core/module_registry.py
MODULE_REGISTRY = {
    "reviews": {
        "name": "Review Intelligence",
        "description": "Multi-platform review aggregation and sentiment analysis",
        "agents": [
            "ingestion_agent",
            "validation_agent",
            "embedding_agent",
            "sentiment_agent",
            "geo_clustering_agent"
        ],
        "api_prefix": "/api/reviews",
        "qrvey_dashboard_id": "review-analytics-dashboard",
        "enabled_by_default": True,
        "required_permissions": ["view_reviews", "manage_reviews"],
        "external_dependencies": ["synup", "google_mybusiness", "yelp"]
    },

    "ecommerce": {
        "name": "E-Commerce Operations",
        "description": "Product catalog, inventory, orders, purchase analytics",
        "agents": [
            "product_sync_agent",
            "inventory_manager_agent",
            "order_processing_agent",
            "purchase_tracker_agent",
            "ltv_calculator_agent"
        ],
        "api_prefix": "/api/ecommerce",
        "qrvey_dashboard_id": "ecommerce-metrics-dashboard",
        "enabled_by_default": False,  # Opt-in plugin
        "required_permissions": ["view_ecommerce", "manage_products",
"manage_inventory"],
        "external_dependencies": ["shopify", "woocommerce"],
        "configuration_schema": {
            "platform": {"type": "enum", "values": ["shopify", "woocommerce",
"custom"]},
            "api_key": {"type": "string", "encrypted": True},
            "store_url": {"type": "url"}
        }
```

```
    },

    "loyalty": {
        "name": "Loyalty & Rewards",
        "description": "Yotpo integration, automatic point awards, referrals",
        "agents": [
            "eligibility_agent",
            "yotpo_client_agent",
            "referral_tracker_agent"
        ],
        "api_prefix": "/api/loyalty",
        "qrvey_dashboard_id": "loyalty-program-dashboard",
        "enabled_by_default": True,
        "required_permissions": ["view_loyalty", "award_points"],
        "external_dependencies": ["yotpo"]
    },

    "marketing": {
        "name": "Marketing Automation",
        "description": "Klaviyo campaigns, GDPR workflows, triggered emails",
        "agents": [
            "campaign_manager_agent",
            "klaviyo_client_agent",
            "consent_manager_agent"
        ],
        "api_prefix": "/api/marketing",
        "qrvey_dashboard_id": "campaign-roi-dashboard",
        "enabled_by_default": True,
        "required_permissions": ["view_campaigns", "create_campaigns"],
        "external_dependencies": ["klaviyo"]
    }
}
```

E-Commerce Plugin Architecture

Agent Modules:

```
agents/ecommerce/
├── product_sync_agent/        # Sync products from Shopify/WooCommerce
├── inventory_manager_agent/   # Track stock, low-stock alerts
├── order_processing_agent/    # Process orders, update fulfillment
├── purchase_tracker_agent/    # Log customer purchases
├── ltv_calculator_agent/      # Calculate customer lifetime value
└── integrations/
    ├── shopify_client.py      # Shopify Admin API wrapper
    ├── woocommerce_client.py  # WooCommerce REST API wrapper
    └── custom_provider.py     # Extensible for custom storefronts
```

Cross-Module Event Triggers:

```
# Example: Product purchase triggers loyalty + marketing workflows

from agents.core.event_bus import publish_event

@ecommerce_graph.node
async def on_purchase_complete(state: PurchaseState):
    # Publish event to Redis Streams
    await publish_event("product_purchased", {
        "customer_id": state.customer_id,
        "product_id": state.product_id,
        "amount": state.total_amount,
        "order_id": state.order_id
    })

    return state

# Consumed by automation pipeline
@automation_graph.listener("product_purchased")
async def handle_purchase(event: dict):
    # Parallel execution of downstream agents
    await asyncio.gather(
        loyalty_agent.award_purchase_points(
            customer_id=event["customer_id"],
            amount=event["amount"]
        ),
        marketing_agent.trigger_campaign(
            customer_id=event["customer_id"],
            campaign_id="post_purchase_thank_you"
        ),
        review_agent.schedule_review_request(
            customer_id=event["customer_id"],
            product_id=event["product_id"],
            delay_days=7  # Ask for review 7 days after purchase
        )
    )
```

Hybrid LLM Cost Optimization

Phased Model Serving Approach

Phase 1 (Weeks 1-2): Ollama CPU - MVP

```
Setup:
  - Ollama 0.3+ on CPU
  - Models: llama3.1:8b-instruct-q4_0 (4.3GB)
  - Embeddings: nomic-embed-text-v1.5 (262MB)

Performance:
  - Sentiment analysis: ~3-5 seconds/request
  - Acceptable for MVP development
```

```
  - Zero cloud costs

Trade-offs:
  - Slower than GPU
  - Limited concurrency (~5 req/sec)
  - Good enough for single-team development
```

Phase 2 (Week 3+): vLLM GPU - Production

```
Setup:
  - vLLM 0.6+ on GPU (NVIDIA A10/A100)
  - Models:
      - llama-3.1-8b-instruct-awq (4-bit quantized)
      - llama-3.1-70b-instruct-awq (optional)
  - LiteLLM gateway (unified API)

Performance:
  - Sentiment analysis: ~200-500ms/request
  - Concurrency: ~50 req/sec (8B model)
  - Production-grade latency

Infrastructure:
  - AWS EC2 g5.xlarge ($1.006/hour = $730/month)
  - OR Runpod/Vast.ai (~$0.40/hour = $290/month)
```

Traffic Distribution Strategy

```
Model Routing Logic:

85% On-Prem (Ollama → vLLM):
  - Simple sentiment analysis (1-2 sentence reviews)
  - Theme extraction (predefined categories)
  - Query classification
  - Duplicate detection
  - Cost: $0/request (after infrastructure)

15% Cloud (Claude 3.5 Sonnet):
  - Complex multi-paragraph reviews
  - Multi-language analysis (non-English)
  - Sarcasm/irony detection
  - High-stakes customer complaints
  - Cost: $0.003/request
```

RAG (Retrieval-Augmented Generation)

Hybrid Retrieval Architecture:

```

```

```python
class RAGAgent:
    """Combines 3 retrieval strategies for comprehensive context"""

    async def retrieve_context(
        self,
        query: str,
        filters: dict,
        top_k: int = 5
    ) -> list[dict]:
        # Execute 3 strategies in parallel (250ms total)
        semantic, structured, spatial = await asyncio.gather(
            self.semantic_search(query, top_k=10),    # Qdrant vector search
            self.sql_search(filters, limit=10),        # PostgreSQL queries
            self.spatial_search(filters.get("h3"), radius=5)  # H3 geo
        )

        # Reciprocal Rank Fusion (combine rankings)
        merged = self.rrf_merge(
            results=[semantic, structured, spatial],
            weights=[0.5, 0.3, 0.2]  # Semantic most important
        )

        # Re-rank with cross-encoder (on-prem, fast)
        reranked = await self.rerank_with_cross_encoder(
            query, merged, top_k=5
        )

        return reranked
```

Vector Database Configuration:

```
# Qdrant collection schema
{
    "name": "reviews",
    "vectors": {
        "size": 768,  # nomic-embed-text-v1.5 dimension
        "distance": "Cosine"
    },
    "payload_schema": {
        "tenant_id": "keyword",      # Multi-tenant isolation
        "review_id": "keyword",
        "source": "keyword",         # google, yelp, facebook
        "rating": "integer",
        "sentiment_score": "float",
        "themes": "keyword[]",
        "h3_index": "keyword",       # Geospatial filter
        "created_at": "datetime"
    },
    "hnsw_config": {
        "m": 16,                     # Balanced accuracy/speed
        "ef_construct": 100
```

```
    }
}
```

Enterprise Security

Multi-Tenant Data Isolation

Row-Level Security (PostgreSQL RLS):

```
-- Enable RLS on all tenant-scoped tables
ALTER TABLE reviews ENABLE ROW LEVEL SECURITY;
ALTER TABLE customers ENABLE ROW LEVEL SECURITY;
ALTER TABLE products ENABLE ROW LEVEL SECURITY;

-- Policy: Users only access their tenant's data
CREATE POLICY tenant_isolation_reviews ON reviews
    USING (tenant_id = current_setting('app.current_tenant')::uuid);

CREATE POLICY tenant_isolation_customers ON customers
    USING (tenant_id = current_setting('app.current_tenant')::uuid);

-- Set tenant context at session start
SET app.current_tenant = '550e8400-e29b-41d4-a716-446655440000';
```

Agent-Level Enforcement:

```
# Every agent receives tenant_id in config
async def process_review(state: ReviewState, config: dict):
    tenant_id = config["configurable"]["tenant_id"]

    async with get_db_session() as db:
        # Set PostgreSQL session variable (enforces RLS)
        await db.execute(f"SET app.current_tenant = '{tenant_id}'")

        # All queries auto-filtered by tenant_id
        review = await db.fetch_one(
            "SELECT * FROM reviews WHERE id = $1",
            state.review_id
        )
        # RLS ensures only this tenant's data returned
```

Authentication & Authorization

Supabase Auth Flow:

```
1. User Login:
   - Frontend: POST to Supabase Auth
```

```
   - Supabase returns JWT with claims: {user_id, tenant_id, role}

2. API Request:
   - Frontend: Authorization: Bearer <JWT>
   - FastAPI: Validate JWT signature (Supabase public key)
   - Extract tenant_id from claims

3. Agent Invocation:
   - FastAPI: app.invoke(state, config={"tenant_id": tenant_id})
   - LangGraph: Passes tenant_id to all nodes
   - PostgreSQL: SET app.current_tenant (RLS enforced)
```

RBAC Permission Matrix:

```
PERMISSION_MATRIX = {
    "owner": ["*"],  # All permissions
    "admin": [
        "view_all_modules",
        "manage_reviews",
        "manage_products",
        "manage_inventory",
        "create_campaigns",
        "award_points",
        "view_analytics"
    ],
    "manager": [
        "view_all_modules",
        "manage_reviews",
        "create_campaigns",
        "view_analytics"
    ],
    "analyst": [
        "view_all_modules",
        "view_analytics"
    ],
    "reviewer": [
        "view_reviews",
        "respond_to_reviews"
    ]
}
```

Data Protection

```
Encryption:
  At Rest: PostgreSQL Transparent Data Encryption (TDE)
  In Transit: TLS 1.3 for all connections
  API Keys: bcrypt hashed, stored in encrypted columns

Secrets Management:
  - All secrets in environment variables
```

```
  - Never committed to Git
  - Rotated every 90 days
  - Separate secrets per environment (dev/staging/prod)

GDPR Compliance:
  - Data deletion API: DELETE /api/customers/{id}/gdpr-erase
  - Data export API: GET /api/customers/{id}/gdpr-export
  - Consent management in marketing module
  - Audit log of all data access (via LangSmith traces)
```

For detailed security architecture, see SECURITY.md

18-Week Development Roadmap

Development Principles

```
Modularity: Each sprint = 1 complete, testable agent module
Incremental Delivery: Deployable to staging every 2 weeks
Quality-First: 90% test coverage, all checks pass before merge
LangGraph-Native: No Celery, pure agent orchestration
Parallel Development: Reviews, Ecommerce, Frontend can proceed simultaneously
```

Phase 1: Foundation & Core Platform (Weeks 1-6)

Week	Module	Deliverables	Exit Criteria
1	**Foundation**	Project scaffold, Docker Compose, DB schema	☑ All services healthy ☑ First LangGraph agent runs ☑ Ollama serving Llama 3.1 8B
2	**Review Ingestion**	Ingestion + Validation + Embedding agents	☑ Reviews ingest from Synup ☑ Embeddings stored in Qdrant ☑ API endpoints working
3	**LLM Infrastructure**	Model router + Sentiment agent + vLLM setup	☑ vLLM serving (GPU) ☑ LiteLLM gateway routing ☑ Sentiment analysis <500ms
4	**Intelligence**	Query + Vector Search + SQL + RAG agents	☑ Semantic search working ☑ Hybrid retrieval (3 strategies) ☑ Q&A returns cited answers
5	**Geo + Events**	Geo-clustering agent + Event bus	☑ H3 indexing working ☑ Event bus publishing ☑ Cross-agent events triggering

Week	Module	Deliverables	Exit Criteria
6	**Automation**	Eligibility + Loyalty + Marketing agents	☑ Review → Loyalty workflow end-to-end ☑ Yotpo points awarded ☑ Klaviyo email sent

Phase 2: Platform Completion (Weeks 7-10)

Week	Module	Deliverables	Exit Criteria
7	**Qrvey Integration**	Dashboard data feeds, WebSocket updates	☑ Qrvey dashboard embedded ☑ Real-time metrics updating ☑ Review module charts live
8	**Web UI (Reviews)**	Next.js review module UI	☑ Web dashboard deployed ☑ White-label theming working ☑ Module registry UI
9	**Security**	Supabase Auth + RLS + RBAC	☑ Multi-tenant isolation verified ☑ Permission tests passing ☑ JWT auth working
10	**Testing & Docs**	Integration tests, load testing, docs	☑ 90%+ test coverage ☑ Load tests passed (100 req/s) ☑ ARCHITECTURE.md complete

Phase 3: E-Commerce Plugin (Weeks 11-14)

Week	Module	Deliverables	Exit Criteria
11	**Ecom Agents**	Product sync + Inventory + Order agents	☑ Shopify products syncing ☑ Inventory tracking working ☑ Order processing functional
12	**Ecom Workflows**	Purchase tracker + LTV calculator + Events	☑ Purchase events triggering loyalty ☑ LTV calculation accurate ☑ Cross-module workflows working
13	**Ecom UI**	E-commerce module dashboard	☑ Product analytics in Qrvey ☑ Inventory alerts working ☑ Purchase history visible

Week	Module	Deliverables	Exit Criteria
14	**Integration Testing**	End-to-end module testing	☑ All modules tested together ☑ No regressions ☑ Security audit passed

Phase 4: Mobile & Production (Weeks 15-18)

Week	Module	Deliverables	Exit Criteria
15	**Mobile Apps**	Expo iOS + Android apps	☑ Apps in TestFlight + Play Console ☑ Offline sync working ☑ Push notifications sent
16	**Performance**	Optimization, caching, CDN	☑ API P95 <200ms ☑ Redis cache hit ratio >80% ☑ LLM responses <500ms
17	**Security Audit**	Penetration testing, OWASP compliance	☑ Pen test report (no critical) ☑ OWASP Top 10 verified ☑ Compliance checklist
18	**Production Launch**	Kubernetes deployment, monitoring	☑ Production cluster live ☑ LangSmith monitoring ☑ First customer onboarded

🗀 Project Structure

```
Review-Engine-Workshop-v2/
├── agents/                          # LangGraph agent modules
│   ├── core/
│   │   ├── orchestrator.py         # Central workflow coordinator
│   │   ├── module_registry.py      # Plugin system
│   │   ├── event_bus.py            # Redis Streams pub/sub
│   │   └── state.py                # Base state schemas
│   │
│   ├── ingestion/                  # Review ingestion pipeline
│   │   ├── ingestion_agent/
│   │   ├── validation_agent/
│   │   └── embedding_agent/
│   │
│   ├── analysis/                   # Intelligence extraction
│   │   ├── model_router_agent/
│   │   ├── sentiment_agent/
│   │   ├── theme_agent/
│   │   └── geo_agent/
│   │
│   ├── intelligence/               # Query & RAG
│   │   ├── query_agent/
│   │   ├── vector_search_agent/
```

```
│   │   ├── sql_agent/
│   │   ├── h3_geo_agent/
│   │   └── rag_agent/
│   │
│   ├── automation/                # Action triggers
│   │   ├── eligibility_agent/
│   │   ├── loyalty_agent/
│   │   ├── marketing_agent/
│   │   └── notification_agent/
│   │
│   ├── ecommerce/                 # E-commerce plugin (Week 11+)
│   │   ├── product_sync_agent/
│   │   ├── inventory_manager_agent/
│   │   ├── order_processing_agent/
│   │   ├── purchase_tracker_agent/
│   │   ├── ltv_calculator_agent/
│   │   └── integrations/
│   │       ├── shopify_client.py
│   │       └── woocommerce_client.py
│   │
│   └── shared/                    # Reusable components
│       ├── tools/                 # LangGraph tool definitions
│       ├── prompts/               # Prompt templates
│       ├── schemas/               # Pydantic state models
│       └── utils/
│
├── app/                           # FastAPI gateway (thin layer)
│   ├── api/
│   │   ├── reviews.py            # Routes to review agents
│   │   ├── ecommerce.py          # Routes to ecom agents
│   │   ├── analytics.py          # Analytics endpoints
│   │   ├── qrvey.py              # Qrvey data feed API
│   │   └── health.py             # Health checks
│   ├── core/
│   │   ├── auth.py               # Supabase Auth + JWT
│   │   ├── permissions.py        # RBAC middleware
│   │   ├── db.py                 # PostgreSQL session
│   │   └── config.py             # Settings (Pydantic)
│   └── main.py                   # FastAPI app factory
│
├── review-engine-ui/              # Next.js frontend
│   ├── app/[tenant]/
│   │   ├── dashboard/            # Qrvey embedded dashboard
│   │   └── modules/
│   │       ├── reviews/          # Review module UI
│   │       ├── ecommerce/        # E-commerce module UI
│   │       ├── loyalty/          # Loyalty module UI
│   │       └── marketing/        # Marketing module UI
│   ├── components/
│   │   ├── shared/               # Reusable UI components
│   │   └── modules/              # Module-specific components
│   └── lib/
│       ├── module-registry.ts    # Frontend module config
│       ├── qrvey-client.ts       # Qrvey SDK wrapper
```

```
│       └── api.ts                # API client
│
├── review-engine-mobile/          # Expo React Native (Week 15)
│   ├── app/
│   │   ├── (auth)/               # Login screens
│   │   └── (tabs)/               # Tab navigation
│   ├── features/
│   │   ├── reviews/
│   │   ├── loyalty/
│   │   └── offline/
│   └── lib/
│       └── sync.ts               # WatermelonDB sync logic
│
├── infrastructure/
│   ├── docker/
│   │   ├── ollama/               # Ollama Dockerfile (Week 1)
│   │   ├── vllm/                 # vLLM Dockerfile (Week 3)
│   │   └── qdrant/               # Qdrant config
│   ├── kubernetes/               # K8s manifests (Week 18)
│   └── terraform/                # Infrastructure as Code
│
├── tests/
│   ├── agents/                   # Agent unit tests
│   ├── integration/              # Graph simulation tests
│   └── e2e/                      # End-to-end tests
│
├── docs/
│   ├── ARCHITECTURE.md           # Technical deep dive
│   ├── SECURITY.md               # Security & compliance
│   ├── DOCUMENTATION.md          # Documentation strategy
│   ├── API.md                    # API reference
│   └── DEPLOYMENT.md             # Production setup guide
│
├── docker-compose.yml            # Local development
├── pyproject.toml                # Python dependencies (Poetry)
├── .env.example                  # Environment variables template
└── README.md                     # This file
```

🔧 Developer Workflow

Daily Development

Start Services:

```
# Terminal 1: Infrastructure
docker-compose up

# Terminal 2: Backend
poetry run python -m agents.main
```

```
# Terminal 3: Frontend
cd review-engine-ui && npm run dev
```

Run Tests:

```
# All tests with coverage
pytest --cov=agents --cov=app --cov-report=html

# Specific agent
pytest tests/agents/test_sentiment_agent.py -v

# Integration tests
pytest tests/integration/ -v

# View coverage
open htmlcov/index.html
```

Debug Agents:

```
# Enable LangSmith tracing
export LANGCHAIN_TRACING_V2=true
export LANGSMITH_API_KEY=your-key

# Run single agent with debug logs
python -m agents.analysis.sentiment_agent \
  --input '{"text": "Great food!", "rating": 5}' \
  --debug

# View trace in LangSmith UI
# https://smith.langchain.com
```

Environment Variables

```
# .env (copy from .env.example)

# Database
DATABASE_URL=postgresql://postgres:password@localhost:5432/reviewengine
REDIS_URL=redis://localhost:6379

# Model Serving
OLLAMA_BASE_URL=http://localhost:11434
VLLM_BASE_URL=http://localhost:8000
LITELLM_API_KEY=sk-litellm-...

# Vector Store
QDRANT_URL=http://localhost:6333
QDRANT_API_KEY=your-qdrant-key
```

```
# Cloud LLMs (Fallback)
ANTHROPIC_API_KEY=sk-ant-...
OPENAI_API_KEY=sk-...

# Integrations
YOTPO_API_KEY=...
KLAVIYO_API_KEY=...
SYNUP_API_KEY=...
SHOPIFY_API_KEY=...

# Observability
LANGSMITH_API_KEY=...
LANGSMITH_PROJECT=review-engine

# Auth
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_ANON_KEY=...
SUPABASE_SERVICE_ROLE_KEY=...

# Qrvey
QRVEY_WORKSPACE_ID=...
QRVEY_EMBED_TOKEN=...
```

Pre-Commit Checks (Must Pass)

```
# Install pre-commit hooks
poetry run pre-commit install

# Manually run all checks
poetry run pre-commit run --all-files

# Individual checks
poetry run ruff check agents/ app/      # Linting
poetry run mypy agents/ app/ --strict  # Type checking
poetry run pytest --cov                 # Tests (≥90%)
poetry run bandit -r agents/ app/       # Security scan
```

🤝 Contributing

Pull Request Process

1. **Fork & Branch**

   ```
   git checkout -b feature/my-agent-module
   ```

2. **Develop & Test**

   - Write agent code in `agents/my_module/`

- Write tests in `tests/agents/test_my_module.py`
- Achieve ≥90% coverage

3. **Pre-Commit Checks**

```
poetry run pre-commit run --all-files
# Must pass: ruff, mypy, pytest, bandit
```

4. **Commit Convention**

```
git commit -m "feat(agents): add inventory manager agent"
# Prefix: feat, fix, docs, test, refactor, perf, chore
```

5. **Open PR**

- Describe what the agent does
- Link to related issues
- Include test results

6. **Code Review**

- Address reviewer comments
- Maintain ≥90% coverage

7. **Merge**

- Squash commits
- Delete branch after merge

Code Standards

```
Required:
  ✓ Type hints on all functions
  ✓ Docstrings (Google style)
  ✓ Unit tests for all agents
  ✓ Integration tests for graphs
  ✓ No hardcoded secrets
  ✓ Pydantic schemas for all state

Forbidden:
  ✗ print() statements (use structlog)
  ✗ Synchronous I/O in agents (use async)
  ✗ Direct database access (use sessions)
  ✗ Missing error handling
```

📚 Documentation

Core Documentation

- **ARCHITECTURE.md** - Technical deep dive into LangGraph agents, model serving, RAG system, deployment
- **SECURITY.md** - Security architecture, compliance (GDPR, SOC 2), IAM, data protection
- **DOCUMENTATION.md** - Documentation strategy, versioning, changelog management
- **API.md** - REST API reference, authentication, webhooks
- **DEPLOYMENT.md** - Kubernetes setup, CI/CD pipelines, monitoring

Support Channels

- **GitHub Issues**: Bug reports, feature requests
- **GitHub Discussions**: Architecture questions, troubleshooting
- **Developer Slack**: #review-engine channel (internal)

Success Metrics

Technical KPIs

Metric	Target	Tracking
API Uptime	>99.9%	Kubernetes readiness probes
API P95 Latency	<200ms	LangSmith traces
LLM Response Time	<500ms	LiteLLM metrics
Test Coverage	>90%	pytest-cov
Vector Search P95	<50ms	Qdrant metrics

Business KPIs

Metric	Target	Tracking
Time to First Insight	<5 min	Qrvey analytics
Customer Onboarding	<30 min	Product analytics
Mobile App Rating	>4.5 ★	App Store/Play Store
LLM Cost per 10k req	<$50	LiteLLM cost tracking

Version History

Version	Date	Changes
3.0	Dec 2024	LangGraph-native architecture, Qrvey integration, hybrid LLM, modular plugins
2.0	Dec 2024	Complete rewrite with mobile apps, white-label, greenfield approach
1.0	Dec 2024	Initial brownfield refactoring plan

License

Project Status

Current Phase: ☑ Development Ready **Target Launch:** 18 weeks from kickoff **Architecture:** Production-ready LangGraph design **Team Readiness:** Modular sprints enable parallel development

Last Updated: December 26, 2024 **Questions?** Review technical details in ARCHITECTURE.md or security in SECURITY.md

---

Built with LangGraph • Powered by Autonomous AI Agents • Ready to Transform Business Intelligence 🚀