# **Contemplative Orb Rating System - Implementation Planning Document**

## **System Overview**

### **Rating System Architecture**

The Contemplative Orb provides a philosophically-aligned, minimalist user feedback mechanism that seamlessly integrates with the existing AI Router quality assessment pipeline. The expandable orb design maintains aesthetic integrity while capturing nuanced user satisfaction data.

### **Rating Scale Definition**

* **-1 (Unclear/🌫️)**: Response obscured understanding or created confusion
* **0 (No Rating)**: User did not provide feedback (default state)
* **1 (Illuminating/💡)**: Response provided helpful clarity or insight
* **2 (Transformative/⭐)**: Response significantly shifted perspective or provided profound insight

## **Database Schema Extensions**

### **New Table: user\_rating\_interactions**

sql

CREATE TABLE user\_rating\_interactions (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

response\_id UUID NOT NULL REFERENCES ai\_responses(id),

user\_id UUID REFERENCES users(id),

session\_id VARCHAR(255) NOT NULL,

rating\_value INTEGER NOT NULL CHECK (rating\_value IN (-1, 1, 2)),

philosopher\_tone VARCHAR(50) NOT NULL,

question\_complexity\_score INTEGER,

selected\_concepts TEXT[],

model\_used VARCHAR(100) NOT NULL,

*-- Interaction Behavior Metrics*

hover\_duration\_ms INTEGER,

time\_to\_rate\_ms INTEGER,

reading\_time\_before\_rating\_ms INTEGER,

rating\_changed BOOLEAN DEFAULT FALSE,

*-- Quality Correlation Data*

predicted\_quality\_score DECIMAL(5,2),

actual\_engagement\_score DECIMAL(5,2),

*-- Timestamps*

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

*-- Indexes for performance*

CREATE INDEX idx\_user\_rating\_response ON user\_rating\_interactions(response\_id);

CREATE INDEX idx\_user\_rating\_user ON user\_rating\_interactions(user\_id);

CREATE INDEX idx\_user\_rating\_philosopher ON user\_rating\_interactions(philosopher\_tone);

CREATE INDEX idx\_user\_rating\_model ON user\_rating\_interactions(model\_used);

CREATE INDEX idx\_user\_rating\_created ON user\_rating\_interactions(created\_at);

### **Extend Existing Tables**

#### **ai\_responses Table Updates**

sql

ALTER TABLE ai\_responses ADD COLUMN user\_rating INTEGER DEFAULT 0;

ALTER TABLE ai\_responses ADD COLUMN rating\_timestamp TIMESTAMP WITH TIME ZONE;

ALTER TABLE ai\_responses ADD COLUMN user\_engagement\_duration\_ms INTEGER;

#### **response\_quality\_metrics Table Integration**

sql

ALTER TABLE response\_quality\_metrics ADD COLUMN user\_rating\_correlation DECIMAL(5,2);

ALTER TABLE response\_quality\_metrics ADD COLUMN rating\_prediction\_accuracy DECIMAL(5,2);

## **Service Architecture Implementation**

### **1. UserRatingService**

#### **Core Responsibilities**

* Capture and validate rating interactions
* Calculate engagement metrics
* Trigger quality correlation analysis
* Manage rating state persistence

#### **Key Methods**

**submit\_rating()**

python

async def submit\_rating(

response\_id: UUID,

rating\_value: int,

interaction\_metrics: RatingInteractionMetrics,

session\_context: SessionContext

) -> RatingSubmissionResult:

"""

Process user rating submission with full context capture

"""

*# Validate rating value and response existence*

*# Calculate engagement metrics*

*# Store rating with interaction data*

*# Trigger async quality correlation analysis*

*# Update response quality metrics*

*# Send analytics events to PostHog*

**get\_rating\_analytics()**

python

async def get\_rating\_analytics(

philosopher\_tone: str,

time\_range: DateRange,

model\_filter: Optional[str] = None

) -> RatingAnalytics:

"""

Generate comprehensive rating analytics for AI Router optimization

"""

*# Aggregate rating distributions*

*# Calculate correlation coefficients*

*# Generate quality-rating alignment metrics*

*# Provide model performance insights*

### **2. QualityCorrelationService**

#### **Core Responsibilities**

* Analyze correlation between predicted and actual user satisfaction
* Update quality prediction algorithms
* Provide feedback to AI Router decision matrix

#### **Key Methods**

**analyze\_quality\_rating\_correlation()**

python

async def analyze\_quality\_rating\_correlation(

response\_id: UUID,

user\_rating: int,

predicted\_quality: float

) -> CorrelationAnalysis:

"""

Analyze correlation between system quality predictions and user ratings

"""

*# Calculate prediction accuracy*

*# Update correlation metrics*

*# Identify prediction improvement opportunities*

*# Feed insights back to QualityAssessmentService*

**update\_quality\_prediction\_weights()**

python

async def update\_quality\_prediction\_weights(

philosopher\_tone: str,

correlation\_data: List[CorrelationPoint]

) -> WeightUpdateResult:

"""

Adjust quality prediction algorithms based on user rating feedback

"""

*# Analyze correlation patterns*

*# Calculate optimal weight adjustments*

*# Update QualityAssessmentService parameters*

*# Validate improvement through A/B testing*

### **3. RatingAnalyticsService**

#### **Core Responsibilities**

* Process rating data for PostHog analytics
* Generate engagement correlation insights
* Support AI Router optimization decisions

#### **Key Methods**

**track\_rating\_event()**

python

async def track\_rating\_event(

rating\_interaction: UserRatingInteraction

) -> None:

"""

Send structured rating events to PostHog analytics

"""

*# Format rating data for PostHog*

*# Include contextual metadata*

*# Track engagement correlation metrics*

*# Send async to prevent response delays*

**generate\_engagement\_insights()**

python

async def generate\_engagement\_insights(

time\_period: TimePeriod

) -> EngagementInsights:

"""

Analyze rating impact on user engagement and retention

"""

*# Calculate rating-to-retention correlation*

*# Identify engagement patterns by philosopher tone*

*# Generate quality improvement recommendations*

*# Support cost-quality optimization decisions*

## **Frontend Integration Specifications**

### **Component Architecture**

#### **ContemplativeOrb Component**

typescript

interface ContemplativeOrbProps {

responseId: string;

philosopherTone: string;

modelUsed: string;

predictedQuality: number;

onRatingSubmit?: (rating: RatingSubmission) => void;

}

interface RatingSubmission {

responseId: string;

rating: -1 | 1 | 2;

interactionMetrics: {

hoverDuration: number;

timeToRate: number;

readingTimeBeforeRating: number;

ratingChanged: boolean;

};

}

#### **Interaction Tracking Implementation**

* **Page Focus Time**: Track time spent reading response before rating appears
* **Hover Analytics**: Measure hover duration over orb and individual rating zones
* **Engagement Patterns**: Log scroll behavior and time on response
* **Rating Decisiveness**: Track if users change their rating selection

### **State Management Integration**

#### **Rating State Schema**

typescript

interface RatingState {

[responseId: string]: {

currentRating: -1 | 0 | 1 | 2;

isSubmitted: boolean;

interactionStartTime: number;

hoverStartTime?: number;

readingStartTime: number;

};

}

## **PostHog Analytics Integration**

### **Event Schema Design**

#### **Core Rating Events**

**rating\_submitted**

json

{

"event": "rating\_submitted",

"properties": {

"response\_id": "uuid",

"rating\_value": -1 | 1 | 2,

"philosopher\_tone": "string",

"model\_used": "string",

"question\_complexity": 1-10,

"predicted\_quality\_score": "number",

"hover\_duration\_ms": "number",

"time\_to\_rate\_ms": "number",

"reading\_time\_ms": "number",

"rating\_changed": "boolean",

"user\_tier": "string",

"session\_id": "string"

}

}

**rating\_interaction\_started**

json

{

"event": "rating\_interaction\_started",

"properties": {

"response\_id": "uuid",

"philosopher\_tone": "string",

"time\_since\_response\_ms": "number",

"scroll\_position": "number",

"viewport\_visible": "boolean"

}

}

**rating\_correlation\_analysis**

json

{

"event": "rating\_correlation\_analysis",

"properties": {

"philosopher\_tone": "string",

"model\_name": "string",

"predicted\_vs\_actual\_correlation": "number",

"quality\_prediction\_accuracy": "number",

"sample\_size": "number",

"time\_period": "string"

}

}

### **Dashboard Metrics Configuration**

#### **Key Performance Indicators**

* **Rating Distribution**: Percentage breakdown of -1, 1, 2 ratings per philosopher
* **Quality Correlation**: Predicted vs actual satisfaction correlation coefficient
* **Engagement Impact**: Rating behavior correlation with session duration
* **Model Performance**: Rating distribution comparison across AI models
* **Cost-Quality Balance**: Rating satisfaction vs cost optimization impact

#### **Cohort Analysis Setup**

* **High Engagement Users**: Users who consistently rate responses
* **Quality Sensitive Users**: Users with higher proportion of "Transformative" ratings
* **Cost-Conscious Segments**: Users affected by cost optimization routing

## **AI Router Integration Points**

### **Quality Assessment Service Updates**

#### **Enhanced Quality Scoring**

python

async def calculate\_enhanced\_quality\_score(

response: AIResponse,

historical\_ratings: List[UserRating]

) -> EnhancedQualityScore:

"""

Incorporate user rating patterns into quality assessment

"""

*# Base quality calculation (existing)*

base\_score = await calculate\_base\_quality\_score(response)

*# User rating correlation adjustment*

rating\_correlation = await get\_rating\_correlation\_factor(

response.philosopher\_tone,

response.model\_used

)

*# Adjusted quality prediction*

enhanced\_score = base\_score \* rating\_correlation

return EnhancedQualityScore(

base\_score=base\_score,

correlation\_factor=rating\_correlation,

enhanced\_score=enhanced\_score,

confidence\_level=calculate\_confidence(historical\_ratings)

)

### **Decision Matrix Service Integration**

#### **User Satisfaction Weighting**

python

async def calculate\_user\_satisfaction\_weight(

philosopher\_tone: str,

model\_options: List[str]

) -> Dict[str, float]:

"""

Add user satisfaction weighting to model selection

"""

satisfaction\_weights = {}

for model in model\_options:

recent\_ratings = await get\_recent\_ratings(

philosopher\_tone=philosopher\_tone,

model=model,

days\_back=30

)

*# Calculate satisfaction score (weighted toward transformative ratings)*

satisfaction\_score = calculate\_weighted\_satisfaction(recent\_ratings)

satisfaction\_weights[model] = satisfaction\_score

return satisfaction\_weights

### **Cost Optimization Service Enhancement**

#### **Quality-Cost Balance with User Feedback**

python

async def optimize\_with\_user\_satisfaction(

user\_budget\_status: BudgetStatus,

quality\_requirements: QualityRequirements,

satisfaction\_history: SatisfactionHistory

) -> OptimizationStrategy:

"""

Balance cost optimization with proven user satisfaction patterns

"""

if user\_budget\_status.is\_critical():

*# Even under cost pressure, avoid models with poor satisfaction history*

return await cost\_optimize\_with\_satisfaction\_floor(satisfaction\_history)

elif user\_budget\_status.is\_warning():

*# Prefer models with high satisfaction-to-cost ratio*

return await optimize\_satisfaction\_per\_dollar(satisfaction\_history)

else:

*# Prioritize satisfaction with cost awareness*

return await maximize\_satisfaction\_within\_budget(satisfaction\_history)

## **Implementation Timeline**

### **Phase 1: Foundation (Week 1-2)**

1. Database schema implementation and migration
2. Core UserRatingService development
3. Basic ContemplativeOrb component implementation
4. PostHog event structure setup

### **Phase 2: Integration (Week 3-4)**

1. QualityCorrelationService implementation
2. AI Router service integration points
3. Analytics dashboard configuration
4. Frontend interaction tracking completion

### **Phase 3: Optimization (Week 5-6)**

1. Quality prediction algorithm enhancement
2. Decision matrix user satisfaction weighting
3. Cost optimization with satisfaction factors
4. Performance monitoring and caching optimization

### **Phase 4: Validation (Week 7-8)**

1. A/B testing framework setup
2. Correlation analysis validation
3. User experience testing and refinement
4. Analytics accuracy validation

## **Success Metrics & Monitoring**

### **Technical Performance Targets**

* Rating submission response time < 200ms
* PostHog event delivery success rate > 99.5%
* Quality correlation calculation accuracy > 90%
* Cache hit ratio for rating analytics > 85%

### **Business Impact Measurements**

* User rating participation rate target: 25-30%
* Quality prediction accuracy improvement: 15-20%
* User satisfaction correlation coefficient: > 0.7
* AI Router decision accuracy improvement: 10-15%

### **User Experience Validation**

* Rating interaction completion rate > 80%
* User satisfaction with rating mechanism > 4.2/5
* Minimal impact on response reading flow
* Philosophical aesthetic alignment maintained

This implementation plan ensures seamless integration of the Contemplative Orb rating system while providing rich data for AI Router optimization and maintaining the app's philosophical design principles.

Retry

Claude does not have the ability to run the code it generates yet.

[Claude can make mistakes.](https://support.anthropic.com/en/articles/8525154-claude-is-providing-incorrect-or-misleading-responses-what-s-going-on)

[Please double-check responses.](https://support.anthropic.com/en/articles/8525154-claude-is-providing-incorrect-or-misleading-responses-what-s-going-on)