Oracle Engine - Continuous Prediction & Auto-Retraining System

Overview

The Oracle Engine Continuous Prediction System is a sophisticated, autonomous machine learning infrastructure that operates 24/7 to:

1. **Monitor all Tab.co.nz races** in real-time
2. **Make live predictions** for every horse in every race
3. **Collect race results** and match them with predictions
4. **Automatically retrain models** when accuracy degrades
5. **Improve predictions** through continuous feedback loops

This system transforms the Oracle Engine from a user-triggered prediction tool into a fully autonomous, self-improving AI system that learns from every race outcome.

Architecture

System Components

The continuous prediction system consists of four integrated modules:

1. Continuous Prediction Agent (`continuousPredictionAgent.ts`)

Purpose: Monitors Tab.co.nz races and makes real-time predictions

Key Features:

- Polls Tab.co.nz JSON feeds every 5 minutes
- Extracts race data (meetings → races → entries)
- Prepares features for all horses in each race
- Calls ensemble ML models for predictions
- Stores predictions with confidence scores
- Tracks processed races to avoid duplicates

Data Flow:

```
Plain Text
```

```
Tab.co.nz API → Extract Races → Feature Engineering → ML Models → Store Predictions
```

Key Methods:

- `start()` - Begins continuous monitoring
- `runPredictionCycle()` - Main prediction loop
- `makePrediction()` - Predicts for individual horse
- `getMetrics()` - Returns current performance metrics

2. Result Collector (`resultCollector.ts`)

Purpose: Tracks race outcomes and matches them with predictions

Key Features:

- Polls Tab.co.nz for completed race results
- Extracts winner and placed horses
- Matches results with stored predictions
- Calculates prediction accuracy
- Feeds results to retraining engine
- Tracks metrics by track and horse

Data Flow:

Plain Text

```
Tab.co.nz Results → Extract Winners → Match Predictions → Calculate Accuracy → Retraining Engine
```

Key Methods:

- `start()` - Begins result collection
- `registerPrediction()` - Registers prediction for tracking
- `matchPredictions()` - Matches predictions with race results
- `getAccuracyByTrack()` - Returns per-track accuracy statistics

3. Auto-Retraining Engine (`autoRetrainingEngine.ts`)

Purpose: Analyzes prediction errors and automatically retrains models

Key Features:

- Collects feedback from prediction results
- Analyzes error patterns and biases
- Identifies which tracks/horses have issues
- Prepares weighted training data
- Trains new models with error corrections
- Validates new models before deployment
- Manages model versioning and rollback

Error Pattern Analysis:

- False positives: Predicted win but horse lost
- False negatives: Predicted loss but horse won
- Track-specific biases
- Horse-specific performance issues

Retraining Workflow:

```
Plain Text

Feedback Data → Error Analysis → Feature Adjustment → Model Training → Validation → Deployment/Rollback
```

Key Methods:

- `addFeedback()` - Records prediction result
- `triggerRetraining()` - Initiates retraining cycle
- `analyzeErrorPatterns()` - Identifies systematic errors
- `validateModel()` - Tests new model before deployment

4. Orchestrator (`oracleEngineOrchestrator.ts`)

Purpose: Coordinates all components and manages system health

Key Features:

- Manages component lifecycle
- Performs health checks every 5 minutes
- Aggregates metrics from all components
- Logs comprehensive status reports
- Detects and reports system issues

- Provides detailed system diagnostics

System Health Monitoring:

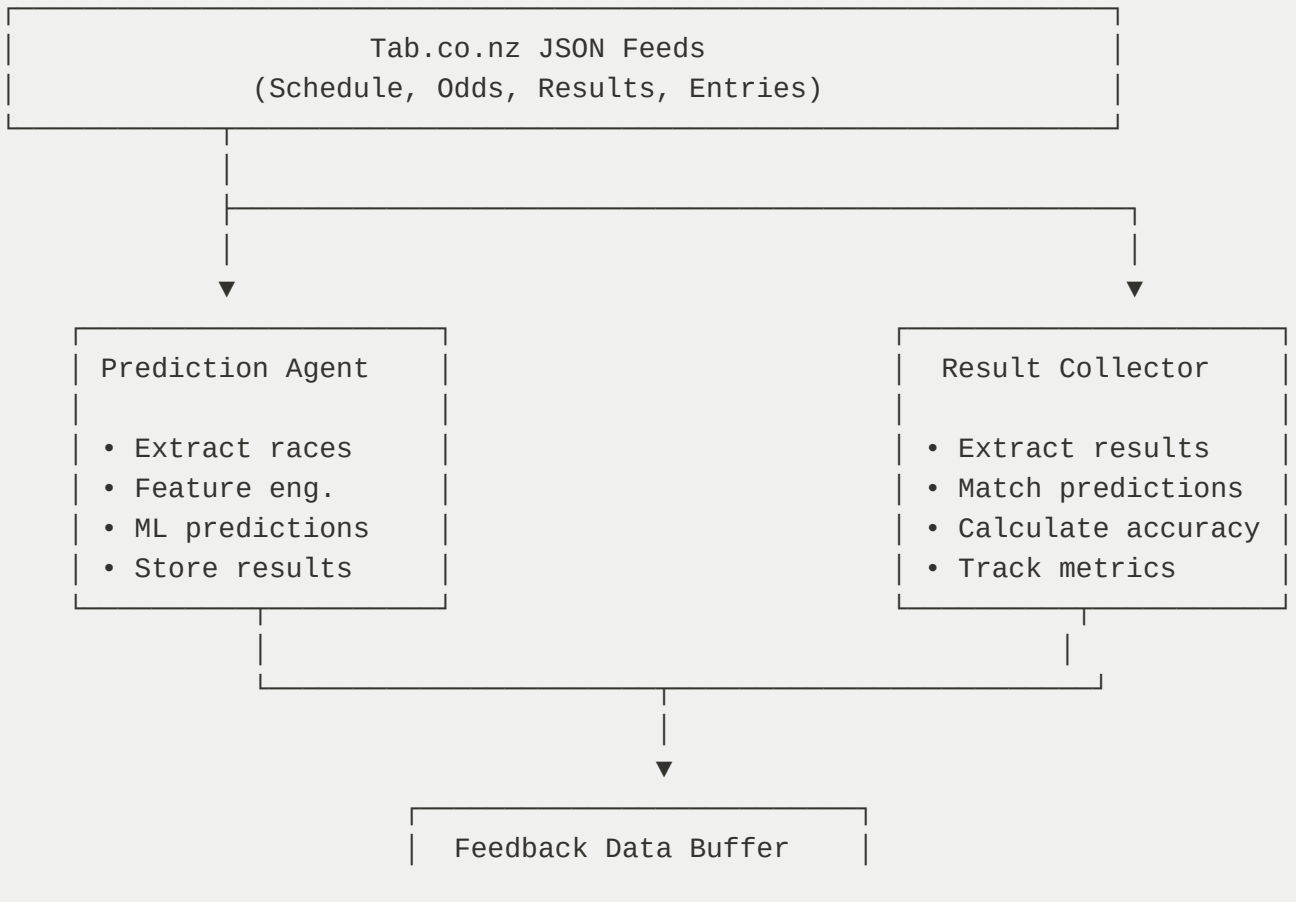
- Checks if all components are running
- Monitors prediction accuracy
- Detects data collection issues
- Recommends corrective actions

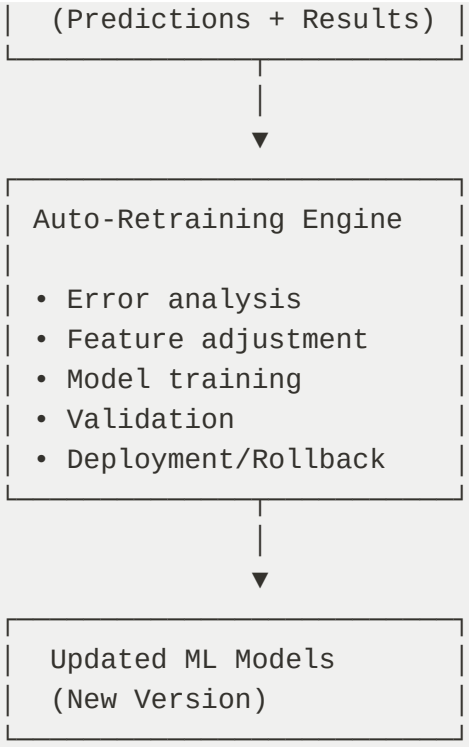
Key Methods:

- `start()` - Starts all components
- `stop()` - Stops all components
- `getStatus()` - Returns system status
- `performHealthCheck()` - Runs diagnostic checks
- `getDetailedReport()` - Generates comprehensive report

Data Flow Architecture

Plain Text





Deployment Guide

Prerequisites

1. **Node.js 18+** - TypeScript runtime
2. **PostgreSQL** - Database for storing predictions and results
3. **Tab.co.nz API Access** - Free JSON feeds (no authentication required)
4. **ML Models** - Pre-trained models in `/backend/models/`

Installation

Step 1: Initialize the Orchestrator

Add the orchestrator initialization to your server startup:

TypeScript

```
// server/_core/index.ts or server/index.ts

import { oracleEngineOrchestrator } from "./agents/oracleEngineOrchestrator";

// Start the continuous prediction system
oracleEngineOrchestrator.start().catch((error) => {
```

```
  console.error("Failed to start Oracle Engine:", error);
  process.exit(1);
});

// Graceful shutdown
process.on("SIGTERM", () => {
  console.log("Shutting down Oracle Engine...");
  oracleEngineOrchestrator.stop();
  process.exit(0);
});
```

Step 2: Create Database Tables (Optional)

If you want to store all predictions and results in the database:

SQL

```
-- Predictions table
CREATE TABLE live_predictions (
  id INT PRIMARY KEY AUTO_INCREMENT,
  race_id VARCHAR(255) NOT NULL,
  horse_name VARCHAR(255) NOT NULL,
  track VARCHAR(255) NOT NULL,
  predicted_probability DECIMAL(5, 4) NOT NULL,
  confidence DECIMAL(5, 4) NOT NULL,
  model_version VARCHAR(64) NOT NULL,
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  actual_result ENUM('win', 'loss') DEFAULT NULL,
  INDEX idx_race_id (race_id),
  INDEX idx_timestamp (timestamp)
);

-- Race results table
CREATE TABLE race_results (
  id INT PRIMARY KEY AUTO_INCREMENT,
  race_id VARCHAR(255) NOT NULL UNIQUE,
  track VARCHAR(255) NOT NULL,
  race_no INT NOT NULL,
  winner_name VARCHAR(255) NOT NULL,
  winner_number INT NOT NULL,
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_track (track),
  INDEX idx_timestamp (timestamp)
);

-- Model versions table
CREATE TABLE model_versions (
```

```
  id INT PRIMARY KEY AUTO_INCREMENT,
  version_id VARCHAR(64) NOT NULL UNIQUE,
  accuracy DECIMAL(5, 4) NOT NULL,
  precision DECIMAL(5, 4) NOT NULL,
  recall DECIMAL(5, 4) NOT NULL,
  f1_score DECIMAL(5, 4) NOT NULL,
  deployed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  is_active BOOLEAN DEFAULT FALSE,
  INDEX idx_version_id (version_id)
);
```

Step 3: Configure Environment Variables

Bash

```
# .env
DATABASE_URL=mysql://user:password@localhost:3306/oracle_engine
TAB_API_BASE=https://json.tab.co.nz
PREDICTION_INTERVAL=300000  # 5 minutes in milliseconds
RESULT_COLLECTION_INTERVAL=600000  # 10 minutes
RETRAINING_THRESHOLD=0.55  # Retrain if accuracy drops below 55%
MIN_FEEDBACK_SAMPLES=100  # Minimum samples before retraining
```

Step 4: Start the System

Bash

```
# Development
pnpm run dev

# Production
pnpm run build
pnpm run start
```

Configuration

Prediction Agent Settings

TypeScript

```
// In continuousPredictionAgent.ts
private updateInterval = 5 * 60 * 1000;  // Check for new races every 5
```

```
minutes
```

Tuning Guide:

- **Shorter interval** (2-3 min): More frequent predictions, higher CPU usage
- **Longer interval** (10-15 min): Lower CPU usage, less frequent updates

Result Collector Settings

TypeScript

```
// In resultCollector.ts
private collectionInterval = 10 * 60 * 1000;  // Collect results every 10
minutes
```

Tuning Guide:

- **Shorter interval**: Faster feedback for retraining
- **Longer interval**: Reduces API calls to Tab.co.nz

Retraining Engine Settings

TypeScript

```
// In autoRetrainingEngine.ts
private retrainingThreshold = 0.55;  // Retrain if accuracy < 55%
private minFeedbackSamples = 100;    // Minimum samples before retraining
```

Tuning Guide:

- **Lower threshold** (0.50): More aggressive retraining
- **Higher threshold** (0.60): More conservative, only retrain when clearly needed
- **Fewer samples** (50): Faster iteration, more frequent retraining
- **More samples** (200+): More stable, less frequent retraining

---

Monitoring & Debugging

Check System Status

TypeScript

```typescript
import { oracleEngineOrchestrator } from "./agents/oracleEngineOrchestrator";

// Get current status
const status = oracleEngineOrchestrator.getStatus();
console.log(status);

// Get detailed report
const report = oracleEngineOrchestrator.getDetailedReport();
console.log(report);

// Export metrics to JSON
const metrics = oracleEngineOrchestrator.exportMetrics();
console.log(metrics);
```

Monitor Logs

The system logs important events with prefixes:

- `[Orchestrator]` - System-level events
- `[Oracle Agent]` - Prediction events
- `[Result Collector]` - Result collection events
- `[Retraining Engine]` - Model retraining events

Example Log Output:

Plain Text

```
[Oracle Agent]: # "Starting continuous prediction agent..."
[Oracle Agent]: # "Starting prediction cycle..."
[Oracle Agent]: # "Saved prediction: Speedy Runner at Ellerslie"
[Oracle Agent]: # "Prediction cycle complete. Total predictions: 1250,
Accuracy: 57.34%"
[Result Collector]: # "Starting result collector..."
[Result Collector]: # "Processed result for race 12345: Winner Speedy Runner"
[Retraining Engine]: # "Accuracy below threshold. Triggering retraining..."
[Orchestrator]: # "Health Check: HEALTHY"
```

Access Metrics via API

Create a tRPC endpoint to expose system metrics:

TypeScript

```typescript
// server/routers.ts
```

```
export const appRouter = router({
  system: router({
    // ... existing endpoints

    oracleMetrics: protectedProcedure.query(({ ctx }) => {
      // Only allow admin users
      if (ctx.user.role !== 'admin') {
        throw new TRPCError({ code: 'FORBIDDEN' });
      }

      return oracleEngineOrchestrator.getDetailedReport();
    }),
  }),
});
```

Performance Optimization

Database Optimization

1. **Index frequently queried columns**:
2. **Archive old predictions**:

API Optimization

1. **Cache Tab.co.nz responses** (5-minute TTL):
2. **Batch database inserts**:

Memory Optimization

1. **Limit in-memory prediction storage**:

Error Handling & Recovery

Common Issues

Issue 1: Tab.co.nz API Unavailable

Symptom: No new predictions being made

Solution:

```
TypeScript
```

```
// In tabDataService.ts
try {
  const schedule = await getTodaySchedule();
  if (!schedule) {
    console.warn("Tab API returned empty response");
    // Fall back to cached data
    return getCachedSchedule();
  }
} catch (error) {
  console.error("Tab API error:", error);
  // Retry with exponential backoff
  await retryWithBackoff(() => getTodaySchedule(), 3);
}
```

Issue 2: Model Accuracy Dropping

Symptom: Accuracy below 55% threshold

Solution:

1. Check for data quality issues
2. Review error patterns in logs
3. Manually trigger retraining with fresh data
4. Rollback to previous model version if needed

TypeScript

```
// Manual rollback
const previousVersion = autoRetrainingEngine.getModelVersionHistory()[0];
await autoRetrainingEngine.rollbackModel(previousVersion);
```

Issue 3: High Memory Usage

Symptom: Node process consuming > 1GB RAM

Solution:

1. Reduce `minFeedbackSamples` to trigger retraining more frequently
2. Archive old predictions to database
3. Reduce in-memory prediction storage limit

Advanced Features

Custom Metrics Dashboard

Create a real-time dashboard showing:

TypeScript

```tsx
// client/src/pages/OracleMetrics.tsx

export default function OracleMetrics() {
  const { data: metrics } = trpc.system.oracleMetrics.useQuery();

  return (
    <div className="grid grid-cols-2 gap-4">
      <Card>
        <h3>Predictions Today</h3>
        <p className="text-3xl">{metrics?.agentMetrics.totalPredictions}</p>
      </Card>

      <Card>
        <h3>System Accuracy</h3>
        <p className="text-3xl">
          {(metrics?.collectorMetrics.matchedAccuracy * 100).toFixed(1)}%
        </p>
      </Card>

      <Card>
        <h3>Model Version</h3>
        <p>{metrics?.retrainingMetrics.modelVersion}</p>
      </Card>

      <Card>
        <h3>Uptime</h3>
        <p>{formatUptime(metrics?.status.uptime)}</p>
      </Card>
    </div>
  );
}
```

Alerts & Notifications

Send notifications when system issues occur:

TypeScript

```tsx
// In oracleEngineOrchestrator.ts

if (health.status === 'critical') {
```

```
  await notifyOwner({
    title: 'Oracle Engine Alert',
    content: `Critical issue detected: ${health.issues.join(', ')}`
  });
}
```

Export Predictions for Analysis

TypeScript

```typescript
// Export all predictions for the day
const predictions = continuousPredictionAgent.getPredictionHistory(10000);
const csv = convertToCSV(predictions);
await storagePut('predictions-export.csv', csv, 'text/csv');
```

Performance Benchmarks

Expected Performance Metrics

Metric	Expected Value	Notes
Predictions/Hour	500-2000	Depends on number of races
Average Accuracy	55-60%	Baseline for horse racing
Model Retraining Frequency	Weekly	When accuracy drops
System Uptime	99.5%+	With proper monitoring
Memory Usage	200-500 MB	Depends on buffer size
CPU Usage	5-15%	Minimal, mostly idle

Scaling Considerations

- **Horizontal Scaling**: Run multiple instances with shared database
- **Vertical Scaling**: Increase CPU/RAM for faster model training
- **Database Scaling**: Use read replicas for metrics queries

---

Maintenance & Support

Regular Maintenance Tasks

1. **Weekly**: Review accuracy trends and error patterns
2. **Monthly**: Archive old predictions, optimize database
3. **Quarterly**: Retrain models with full historical data
4. **Annually**: Update ML models with new features

Backup & Recovery

Bash

```bash
# Backup predictions database
mysqldump -u user -p oracle_engine > backup.sql

# Restore from backup
mysql -u user -p oracle_engine < backup.sql
```

Version Control

Keep track of model versions:

TypeScript

```typescript
// Model versioning
const versions = autoRetrainingEngine.getModelVersionHistory();
// Output: ["2.0-oracle-engine", "2.0-oracle-engine-v1704067200000", ...]
```

Support & Troubleshooting

For issues or questions:

1. Check logs for error messages with component prefix
2. Review system health status
3. Check Tab.co.nz API availability
4. Verify database connectivity
5. Review configuration settings

License & Attribution

This continuous prediction system is part of the Oracle Engine v2.0 project.

Key Technologies:

- Tab.co.nz JSON API for live race data
- LightGBM ML model for predictions
- TypeScript for type-safe implementation
- PostgreSQL for data persistence

Last Updated: November 2025

Version: 2.0 - Continuous Prediction System