Online Learning Module

Overview

The Online Learning Module enables continuous improvement of the multi-agent system through real-time model updates based on outcome events. It implements a micro-batch learning approach that processes agent execution outcomes and updates predictive models without requiring full retraining.

Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                    Feedback Pipeline                            │
│  ┌──────────────┐    ┌──────────────┐    ┌──────────────┐       │
│  │    Agents    │───▶│ ActionDispat │───▶│    Kafka     │       │
│  │   Execute    │    │     cher     │    │   Producer   │       │
│  └──────────────┘    └──────────────┘    └──────┬───────┘       │
└─────────────────────────────────────────────────┼───────────────┘
                                                  │
                                                  │
                              Topic: agent-outcomes
                                                  │
                                                  │
┌─────────────────────────────────────────────────┼───────────────┐
│             Real-Time Model Updater             │               │
│  ┌──────────────────────────────────────────────▼────────────┐  │
│  │          Kafka Consumer                                   │  │
│  │  ┌─────────────────────────────────────────────────────┐  │  │
│  │  │         Micro-Batch Processing                      │  │  │
│  │  │                                                     │  │  │
│  │  │  1. Collect events (batch_size=10, timeout=30s)     │  │  │
│  │  │  2. Extract features from outcomes                  │  │  │
│  │  │  3. Update models incrementally                     │  │  │
│  │  │  4. Calculate performance metrics                   │  │  │
│  │  │  5. Persist updated models                          │  │  │
│  │  └─────────────────────────────────────────────────────┘  │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │              Model Repository                             │  │
│  │  ┌──────────────────────┐  ┌──────────────────────┐       │  │
│  │  │ Agent Selection      │  │ Quality Prediction   │       │  │
│  │  │  Model               │  │  Model               │       │  │
│  │  │                      │  │                      │       │  │
│  │  │ - Thompson Sampling  │  │ - Performance Score  │       │  │
│  │  │ - Success Rates      │  │ - Latency Estimate   │       │  │
│  │  │ - Task Patterns      │  │                      │       │  │
│  │  └──────────────────────┘  └──────────────────────┘       │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │            REST API for Predictions                       │  │
│  │                                                           │  │
│  │  POST /api/learning/predict/agent-selection               │  │
│  │  GET  /api/learning/metrics                               │  │
│  │  GET  /api/learning/agents/performance                    │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
```

Key Components

1. AgentPerformanceModel

Tracks agent performance patterns and learns optimal selections using a multi-armed bandit approach with Thompson Sampling.

Features:
- Success/failure tracking per agent
- Latency and quality score monitoring
- Task-specific performance patterns
- Context-aware recommendations
- Thompson Sampling for exploration/exploitation balance

Usage:

```python
from core.online_learning import AgentPerformanceModel

model = AgentPerformanceModel()

# Update with outcome event
model.update(outcome_event)

# Get predictions
recommendations = model.predict_best_agent(
    task_type="analysis",
    context={"complexity": "high"},
    top_k=3
)
```

2. RealTimeModelUpdater

Subscribes to Kafka outcome events and performs micro-batch updates on ML models.

Configuration:

```python
from core.online_learning import RealTimeModelUpdater

updater = RealTimeModelUpdater(
    kafka_servers="localhost:9092",
    kafka_topic="agent-outcomes",
    batch_size=10,              # Events per batch
    batch_timeout_seconds=30,   # Max wait time
    model_storage_path="./models",
    enable_auto_save=True,
    save_interval_seconds=300   # Save every 5 minutes
)

# Start the updater
updater.start()
```

Environment Variables:

```
# Kafka Configuration
KAFKA_BOOTSTRAP_SERVERS=localhost:9092
KAFKA_OUTCOME_TOPIC=agent-outcomes
ENABLE_KAFKA=true

# Learning Configuration
MODEL_BATCH_SIZE=10
MODEL_BATCH_TIMEOUT=30
MODEL_STORAGE_PATH=./models
MODEL_SAVE_INTERVAL=300
```

3. Learning Metrics

The system tracks comprehensive metrics:

```
{
    "total_updates": 150,
    "successful_updates": 148,
    "failed_updates": 2,
    "avg_update_time_ms": 45.3,
    "samples_processed": 1500,
    "current_model_score": 0.87,
    "improvement_rate": 0.05
}
```

4. Model Versioning

Models are versioned and persisted with full metadata:

```
@dataclass
class ModelVersion:
    version_id: str
    model_type: ModelType
    created_at: str
    parameters: Dict[str, Any]
    metrics: Dict[str, float]
    training_samples: int
    parent_version: Optional[str]
```

API Endpoints

Get Learning Metrics

```
GET /api/learning/metrics

Response:
{
    "total_updates": 150,
    "successful_updates": 148,
    "failed_updates": 2,
    "avg_update_time_ms": 45.3,
    "samples_processed": 1500,
    "current_model_score": 0.87,
    "improvement_rate": 0.05
}
```

Get Agent Performance

```
GET /api/learning/agents/performance

Response:
[
    {
        "agent_name": "ReActAgent",
        "success_rate": 0.92,
        "avg_latency_ms": 1200.5,
        "total_executions": 450
    },
    {
        "agent_name": "DebateAgent",
        "success_rate": 0.88,
        "avg_latency_ms": 2500.3,
        "total_executions": 320
    }
]
```

Predict Best Agent

```
POST /api/learning/predict/agent-selection

Request:
{
    "task_type": "compliance_analysis",
    "context": {
        "complexity": "high",
        "domain": "financial"
    },
    "top_k": 3
}

Response:
{
    "recommendations": [
        {
            "agent_name": "ReActAgent",
            "score": 0.89,
            "rank": 1
        },
        {
            "agent_name": "EvaluatorAgent",
            "score": 0.85,
            "rank": 2
        },
        {
            "agent_name": "GovernorAgent",
            "score": 0.82,
            "rank": 3
        }
    ],
    "model_score": 0.87,
    "timestamp": "2025-10-09T12:34:56"
}
```

Get Model Status

```
GET /api/learning/status

Response:
{
    "status": "running",
    "running": true,
    "kafka_topic": "agent-outcomes",
    "batch_size": 10,
    "models_loaded": ["agent_selection"]
}
```

Learning Strategies

1. Thompson Sampling (Default)

Used for agent selection to balance exploration and exploitation:

```
# Prior belief: Beta(α, β)
alpha = successes + 1
beta = failures + 1

# Sample from posterior
score = np.random.beta(alpha, beta)
```

Advantages:
- Naturally balances exploration/exploitation
- No hyperparameter tuning needed
- Works well with sparse data
- Provides probabilistic guarantees

2. Moving Average

Simple moving average for metrics:

```
new_avg = (old_avg * n + new_value) / (n + 1)
```

3. Exponential Moving Average

Gives more weight to recent observations:

```
new_avg = alpha * new_value + (1 - alpha) * old_avg
```

Integration Guide

Step 1: Initialize Kafka

Ensure Kafka is running and configured:

```bash
# Start Kafka
docker run -d \
  --name kafka \
  -p 9092:9092 \
  -e KAFKA_LISTENERS=PLAINTEXT://0.0.0.0:9092 \
  confluentinc/cp-kafka:latest
```

Step 2: Enable Feedback Pipeline

Configure the feedback pipeline in your orchestrator:

```python
from core.feedback_pipeline import get_feedback_pipeline
from config.kafka_config import kafka_config

pipeline = get_feedback_pipeline(
    kafka_servers=kafka_config.KAFKA_BOOTSTRAP_SERVERS,
    kafka_topic=kafka_config.KAFKA_OUTCOME_TOPIC
)
```

Step 3: Start Model Updater

Initialize and start the model updater:

```python
from core.online_learning import get_model_updater

updater = get_model_updater(
    kafka_servers="localhost:9092",
    force_new=True
)

updater.start()
```

Step 4: Use Predictions

Query the model for agent recommendations:

```python
# Via Python API
predictions = updater.predict(
    ModelType.AGENT_SELECTION,
    task_type="analysis",
    top_k=3
)

# Via REST API
import requests

response = requests.post(
    "http://localhost:8000/api/learning/predict/agent-selection",
    json={
        "task_type": "analysis",
        "top_k": 3
    }
)
recommendations = response.json()["recommendations"]
```

Database Schema

The online learning module uses additional database tables:

```sql
-- Model versions
CREATE TABLE model_versions (
    id VARCHAR(36) PRIMARY KEY,
    model_type VARCHAR(100),
    version_number INTEGER,
    status ENUM('training', 'active', 'deprecated', 'archived'),
    parameters JSON,
    metrics JSON,
    model_file_path VARCHAR(500),
    created_at TIMESTAMP
);

-- Learning events
CREATE TABLE learning_events (
    id VARCHAR(36) PRIMARY KEY,
    model_version_id VARCHAR(36),
    event_type VARCHAR(100),
    batch_size INTEGER,
    samples_processed INTEGER,
    metrics_before JSON,
    metrics_after JSON,
    improvement FLOAT,
    duration_ms FLOAT,
    created_at TIMESTAMP
);
```

Performance Considerations

Batch Size Selection

- **Small batches (5-10)**: Faster updates, more responsive to changes
- **Medium batches (10-50)**: Balanced approach
- **Large batches (50+)**: More stable updates, better for production

Memory Management

The system uses bounded buffers:

```python
# Agent stats use deques with maxlen
latencies = deque(maxlen=100)
quality_scores = deque(maxlen=100)
```

Persistence Strategy

- **Auto-save**: Models saved every 5 minutes (configurable)
- **Manual save**: Trigger via API endpoint
- **On shutdown**: Models automatically persisted

Monitoring

Key Metrics to Track

1. **Update Performance**
   - Updates per minute
   - Average update time
   - Success rate

2. **Model Quality**
   - Current model score
   - Improvement rate over time
   - Prediction accuracy

3. **Agent Performance**
   - Success rates by agent
   - Average latencies
   - Task-specific patterns

Alerting Thresholds

```python
# Recommended alerts
if metrics.failed_updates / metrics.total_updates > 0.05:
    alert("High failure rate in model updates")

if metrics.improvement_rate < -0.1:
    alert("Model performance degrading")

if metrics.avg_update_time_ms > 1000:
    alert("Slow model updates")
```

Testing

Run the test suite:

```bash
# Unit tests
pytest tests/test_online_learning.py -v

# Integration tests (requires Kafka)
pytest tests/test_online_learning.py -v --kafka

# Performance tests
pytest tests/test_online_learning.py -v --benchmark
```

Troubleshooting

Issue: Model not updating

Check:
1. Kafka connection: `kafka-topics --list`
2. Consumer group status: `kafka-consumer-groups --describe`
3. Event format: Verify OutcomeEvent schema

Issue: High latency

Solutions:
1. Increase batch size
2. Reduce save frequency
3. Use asynchronous processing

Issue: Poor prediction quality

Solutions:
1. Increase data collection period
2. Check for data quality issues
3. Adjust Thompson Sampling priors
4. Review context extraction logic

Future Enhancements

1. **Additional Models**
   - Quality prediction model
   - Latency prediction model
   - Cost optimization model

2. **Advanced Learning**
   - Neural networks for complex patterns
   - Transfer learning between tenants
   - Multi-objective optimization

3. **A/B Testing**
   - Compare model versions
   - Gradual rollout of new models
   - Statistical significance testing

4. **Federated Learning**
   - Cross-tenant learning (privacy-preserving)
   - Distributed model training
   - Model aggregation strategies

References

- Thompson Sampling: Tutorial (https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf)
- Multi-Armed Bandits: Book (https://tor-lattimore.com/downloads/book/book.pdf)
- Online Learning: Wikipedia (https://en.wikipedia.org/wiki/Online_machine_learning)