AI Routing Troubleshooting Guide

Overview

This guide provides comprehensive troubleshooting information for the AI Routing System, including common issues, diagnostic procedures, and resolution strategies.

Table of Contents

Quick Diagnosis

Health Check Commands

```
# Quick system health check
curl -f http://localhost:8080/health || echo "System unhealthy"

# Check AI routing service specifically
curl -f http://localhost:8080/api/ai/health || echo "AI routing service unavailable"

# Check provider status
curl -s http://localhost:8080/api/ai/providers | jq '.providers[] | {name: .name,
status: .status, healthScore: .healthScore}'
```

Status Indicators

Status	Meaning	Action Required
🟢 Healthy	All systems operational	None
🟡 Degraded	Some providers down	Monitor, may self-heal
🟠 Warning	Performance issues	Investigate logs
🔴 Critical	Service unavailable	Immediate action needed

Quick Fixes

1. **Service Restart**:

```
# Restart AI routing service
docker-compose restart ai-routing-service

# Check if service is responding
curl -f http://localhost:8080/api/ai/health
```

1. **Provider Reset**:

```
# Reset provider connections
curl -X POST http://localhost:8080/api/ai/providers/reset

# Verify provider status
curl http://localhost:8080/api/ai/providers
```

1. **Cache Clear**:

```
# Clear request cache
curl -X DELETE http://localhost:8080/api/ai/cache

# Verify cache status
curl http://localhost:8080/api/ai/cache/stats
```

Common Issues

1. Request Timeouts

Symptoms:
- Requests taking longer than expected
- Timeout errors in logs
- Client applications hanging

Diagnosis:

```
# Check current timeout settings
curl http://localhost:8080/api/ai/config | jq '.timeout'

# Monitor request latency
curl http://localhost:8080/api/ai/metrics | jq '.latency'

# Check provider response times
curl http://localhost:8080/api/ai/providers | jq '.providers[] | {name: .name,
avgLatency: .avgLatency}'
```

Solutions:

1. **Increase Timeout Values**:

```
// In config/app.config.ts
export const appConfig = {
  routing: {
    timeout: 60000, // Increase from 30s to 60s
    providerTimeout: 45000 // Increase provider-specific timeout
  }
};
```

1. **Optimize Request Parameters**:

```
// Reduce token limits for faster responses
const request = {
  model: 'gpt-4o-mini', // Use faster model
  maxTokens: 200, // Reduce from higher values
  temperature: 0.3 // Lower temperature for faster generation
};
```

1. **Check Provider Health**:

```
# Identify slow providers
curl http://localhost:8080/api/ai/providers | jq '.providers[] | select(.avgLatency > 5000)'
```

2. Authentication Errors

Symptoms:
- 401 Unauthorized responses
- API key validation failures
- Provider authentication issues

Diagnosis:

```
# Check environment variables
echo "OpenAI Key: ${OPENAI_API_KEY:0:10}..."
echo "Anthropic Key: ${ANTHROPIC_API_KEY:0:10}..."
echo "Google Key: ${GOOGLE_API_KEY:0:10}..."

# Test API keys directly
curl -H "Authorization: Bearer $OPENAI_API_KEY" https://api.openai.com/v1/models
```

Solutions:

1. **Verify API Keys**:

```
# Check if keys are properly set
./scripts/env-validation.sh

# Regenerate keys if needed
# Visit provider dashboards to create new keys
```

1. **Update Environment Configuration**:

```
# Reload environment variables
source .env

# Restart services to pick up new config
docker-compose restart
```

1. **Check Key Permissions**:

```
# Test key permissions with minimal request
curl -H "Authorization: Bearer $OPENAI_API_KEY" \
     -H "Content-Type: application/json" \
     -d '{"model":"gpt-4o-mini","messages":
[{"role":"user","content":"test"}],"max_tokens":5}' \
     https://api.openai.com/v1/chat/completions
```

3. Rate Limiting Issues

Symptoms:
- 429 Rate Limit Exceeded errors
- Requests being delayed or dropped
- Uneven provider usage

Diagnosis:

```
# Check current rate limit status
curl http://localhost:8080/api/ai/rate-limits

# Monitor request patterns
curl http://localhost:8080/api/ai/metrics | jq '.requestsPerMinute'

# Check provider-specific limits
curl http://localhost:8080/api/ai/providers | jq '.providers[] | {name: .name, request-
sRemaining: .requestsRemaining}'
```

Solutions:

1. **Implement Exponential Backoff**:

```
async function requestWithBackoff(request: AIRequest, maxRetries = 3): Promise<AIRe-
sponse> {
  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      return await aiRoutingService.processRequest(request);
    } catch (error) {
      if (error.status === 429 && attempt < maxRetries) {
        const delay = Math.min(1000 * Math.pow(2, attempt), 60000);
        await new Promise(resolve => setTimeout(resolve, delay));
        continue;
      }
      throw error;
    }
  }
}
```

1. **Configure Rate Limiting**:

```
// Adjust rate limiting configuration
const rateLimitConfig = {
  openai: { requestsPerMinute: 3500, tokensPerMinute: 90000 },
  anthropic: { requestsPerMinute: 50, tokensPerMinute: 40000 },
  google: { requestsPerMinute: 300, tokensPerMinute: 32000 }
};
```

1. **Load Balance Across Providers**:

```
# Enable load balancing
curl -X PUT http://localhost:8080/api/ai/config \
     -H "Content-Type: application/json" \
     -d '{"loadBalancing": {"enabled": true, "strategy": "least_loaded"}}'
```

4. Quality Score Issues

Symptoms:
- Consistently low quality scores
- User complaints about response quality
- Quality degradation alerts

Diagnosis:

```
# Check recent quality metrics
curl http://localhost:8080/api/ai/quality/metrics | jq '.recentScores'

# Analyze quality trends
curl http://localhost:8080/api/ai/quality/trends?period=24h

# Check provider-specific quality
curl http://localhost:8080/api/ai/quality/by-provider
```

Solutions:

1. **Review Quality Configuration**:

```
// Adjust quality scoring parameters
const qualityConfig = {
  scoringEnabled: true,
  minimumScore: 7.0,
  dimensions: {
    accuracy: { weight: 0.3, threshold: 8.0 },
    relevance: { weight: 0.25, threshold: 7.5 },
    completeness: { weight: 0.2, threshold: 7.0 },
    clarity: { weight: 0.15, threshold: 7.0 },
    creativity: { weight: 0.1, threshold: 6.0 }
  }
};
```

1. **Implement Quality Improvements**:

```
# Get improvement recommendations
curl http://localhost:8080/api/ai/quality/recommendations

# Apply recommended optimizations
curl -X POST http://localhost:8080/api/ai/quality/apply-recommendations \
     -H "Content-Type: application/json" \
     -d '{"recommendationIds": ["rec-1", "rec-2"]}'
```

1. **Monitor Quality Patterns**:

```
# Set up quality monitoring
curl -X POST http://localhost:8080/api/ai/quality/monitor \
     -H "Content-Type: application/json" \
     -d '{"threshold": 7.0, "alertOnDrop": true, "windowSize": "1h"}'
```

Provider-Specific Issues

OpenAI Issues

Common Problems:

1. **Model Not Available**:

```
# Check available models
curl -H "Authorization: Bearer $OPENAI_API_KEY" \
     https://api.openai.com/v1/models | jq '.data[].id'

# Update model configuration
curl -X PUT http://localhost:8080/api/ai/providers/openai/models \
     -H "Content-Type: application/json" \
     -d '{"models": ["gpt-4o", "gpt-4o-mini", "gpt-3.5-turbo"]}'
```

1. **Function Calling Issues**:

```
// Validate function schema
const functionSchema = {
  name: 'get_weather',
  description: 'Get weather information',
  parameters: {
    type: 'object',
    properties: {
      location: { type: 'string', description: 'City name' }
    },
    required: ['location']
  }
};

// Test function calling
const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o',
  messages: [{ role: 'user', content: 'Weather in Paris?' }],
  functions: [functionSchema],
  functionCall: 'auto'
});
```

1. **Vision Processing Errors**:

```
# Check image format and size
file image.jpg
ls -lh image.jpg

# Ensure proper base64 encoding
base64 -i image.jpg | head -c 50
```

Anthropic (Claude) Issues

Common Problems:

1. **Context Length Errors**:

```
// Implement context management
function manageClaudeContext(messages: Message[]): Message[] {
  const maxTokens = 180000; // Claude-3 context limit
  let totalTokens = 0;
  const managedMessages: Message[] = [];

  // Process messages in reverse order (keep most recent)
  for (let i = messages.length - 1; i >= 0; i--) {
    const messageTokens = estimateTokens(messages[i].content);

    if (totalTokens + messageTokens > maxTokens) {
      break;
    }

    totalTokens += messageTokens;
    managedMessages.unshift(messages[i]);
  }

  return managedMessages;
}
```

1. **Tool Integration Problems**:

```
// Debug tool execution
async function debugToolExecution(toolCall: ToolCall): Promise<any> {
  console.log('Executing tool:', toolCall.name);
  console.log('Input schema:', JSON.stringify(toolCall.input, null, 2));

  try {
    const result = await executeToolCall(toolCall);
    console.log('Tool result:', JSON.stringify(result, null, 2));
    return result;
  } catch (error) {
    console.error('Tool execution failed:', error.message);
    console.error('Stack trace:', error.stack);
    throw error;
  }
}
```

1. **Content Filtering Issues**:

```
# Check content safety scores
curl -X POST http://localhost:8080/api/ai/safety/check \
     -H "Content-Type: application/json" \
     -d '{"content": "Your content here", "provider": "anthropic"}'
```

Google (Gemini) Issues

Common Problems:

1. **Multimodal Processing Failures**:

```
// Debug multimodal requests
function debugMultimodalRequest(request: any): void {
  console.log('Content parts:', request.messages[0].content.length);

  request.messages[0].content.forEach((part, index) => {
    console.log(`Part ${index}:`, part.type);
    if (part.type === 'image_url') {
      console.log('Image URL length:', part.image_url.url.length);
    }
  });
}
```

1. **Safety Settings Issues**:

```
// Configure Gemini safety settings
const safetySettings = [
  {
    category: 'HARM_CATEGORY_HARASSMENT',
    threshold: 'BLOCK_MEDIUM_AND_ABOVE'
  },
  {
    category: 'HARM_CATEGORY_HATE_SPEECH',
    threshold: 'BLOCK_MEDIUM_AND_ABOVE'
  },
  {
    category: 'HARM_CATEGORY_SEXUALLY_EXPLICIT',
    threshold: 'BLOCK_MEDIUM_AND_ABOVE'
  },
  {
    category: 'HARM_CATEGORY_DANGEROUS_CONTENT',
    threshold: 'BLOCK_MEDIUM_AND_ABOVE'
  }
];
```

Performance Issues

High Latency

Diagnosis Steps:

1. **Identify Bottlenecks**:

```
# Check service response times
curl -w "@curl-format.txt" -o /dev/null -s http://localhost:8080/api/ai/health

# Monitor database performance
curl http://localhost:8080/api/ai/db-stats

# Check cache performance
curl http://localhost:8080/api/ai/cache/stats
```

1. **Profile Request Processing**:

```
# Enable profiling
curl -X POST http://localhost:8080/api/ai/profiling/enable

# Generate load and analyze
ab -n 100 -c 10 http://localhost:8080/api/ai/simple-request

# Get profiling results
curl http://localhost:8080/api/ai/profiling/results
```

Solutions:

1. **Enable Caching**:

```
// Configure intelligent caching
const cacheConfig = {
  enabled: true,
  ttl: 3600, // 1 hour
  maxSize: 1000,
  strategy: 'lru',
  compression: true
};
```

1. **Optimize Database Queries**:

```
-- Add indexes for common queries
CREATE INDEX idx_ai_requests_user_id ON ai_requests(user_id);
CREATE INDEX idx_ai_requests_created_at ON ai_requests(created_at);
CREATE INDEX idx_ai_requests_provider ON ai_requests(provider);
```

1. **Implement Connection Pooling**:

```
const poolConfig = {
  min: 2,
  max: 10,
  acquireTimeoutMillis: 30000,
  createTimeoutMillis: 30000,
  destroyTimeoutMillis: 5000,
  idleTimeoutMillis: 30000,
  reapIntervalMillis: 1000
};
```

Memory Issues

Diagnosis:

```
# Monitor memory usage
docker stats ai-routing-service

# Check for memory leaks
curl http://localhost:8080/api/ai/memory-stats

# Analyze heap usage
node --inspect=0.0.0.0:9229 /app/dist/index.js
```

Solutions:

1. **Implement Memory Management**:

```
// Clear unused objects
setInterval(() => {
  if (global.gc) {
    global.gc();
  }
}, 60000); // Every minute

// Monitor memory usage
setInterval(() => {
  const memUsage = process.memoryUsage();
  console.log('Memory usage:', {
    rss: Math.round(memUsage.rss / 1024 / 1024) + 'MB',
    heapUsed: Math.round(memUsage.heapUsed / 1024 / 1024) + 'MB',
    heapTotal: Math.round(memUsage.heapTotal / 1024 / 1024) + 'MB'
  });
}, 30000);
```

1. **Optimize Cache Size**:

```
# Reduce cache size if memory constrained
curl -X PUT http://localhost:8080/api/ai/cache/config \
     -H "Content-Type: application/json" \
     -d '{"maxSize": 500, "ttl": 1800}'
```

Error Codes Reference

HTTP Status Codes

Code	Meaning	Typical Causes	Solutions
400	Bad Request	Invalid input parameters	Validate request format
401	Unauthorized	Missing/invalid API key	Check authentication
403	Forbidden	Insufficient permissions	Review user roles
404	Not Found	Invalid endpoint/resource	Check API documentation
408	Request Timeout	Request took too long	Increase timeout limits
429	Rate Limited	Too many requests	Implement backoff strategy
500	Internal Error	Server-side issue	Check logs, restart service
502	Bad Gateway	Provider API issue	Check provider status
503	Service Unavailable	Service overloaded	Scale up resources

Custom Error Codes

Code	Description	Resolution
`AI_PROVIDER_UNAVAILABLE`	All providers are down	Check provider health
`AI_QUALITY_THRESHOLD_NOT_MET`	Response quality too low	Adjust quality settings
`AI_COST_BUDGET_EXCEEDED`	Cost limits reached	Increase budget or optimize
`AI_CONTEXT_TOO_LONG`	Input context exceeds limits	Truncate or summarize
`AI_MODEL_NOT_SUPPORTED`	Requested model unavailable	Use different model

Diagnostic Tools

Health Check Script

```
#!/bin/bash
# health-check.sh

echo "🔍 AI Routing System Health Check"
echo "================================="

# Check service availability
echo -n "Service Health: "
if curl -f -s http://localhost:8080/health > /dev/null; then
    echo "✅ Healthy"
else
    echo "❌ Unhealthy"
fi

# Check providers
echo -n "Provider Status: "
PROVIDERS=$(curl -s http://localhost:8080/api/ai/providers | jq -r '.providers[] | se-
lect(.status != "healthy") | .name' | wc -l)
if [ "$PROVIDERS" -eq 0 ]; then
    echo "✅ All providers healthy"
else
    echo "⚠️  $PROVIDERS providers unhealthy"
fi

# Check recent errors
echo -n "Recent Errors: "
ERRORS=$(curl -s http://localhost:8080/api/ai/metrics | jq -r '.errors.last1hour')
if [ "$ERRORS" -lt 10 ]; then
    echo "✅ Low error rate ($ERRORS)"
else
    echo "⚠️  High error rate ($ERRORS)"
fi

# Check performance
echo -n "Average Latency: "
LATENCY=$(curl -s http://localhost:8080/api/ai/metrics | jq -r '.latency.average')
echo "${LATENCY}ms"

echo ""
echo "For detailed diagnostics, run: curl http://localhost:8080/api/ai/diagnostics"
```

Performance Monitor

```
#!/bin/bash
# performance-monitor.sh

echo "📊 Performance Monitoring"
echo "========================"

while true; do
    clear
    echo "$(date): AI Routing Performance"
    echo "------------------------------"

    # Get metrics
    METRICS=$(curl -s http://localhost:8080/api/ai/metrics)

    echo "Requests/min: $(echo $METRICS | jq -r '.requestsPerMinute')"
    echo "Average Latency: $(echo $METRICS | jq -r '.latency.average')ms"
    echo "Success Rate: $(echo $METRICS | jq -r '.successRate')%"
    echo "Active Connections: $(echo $METRICS | jq -r '.activeConnections')"
    echo "Memory Usage: $(echo $METRICS | jq -r '.memoryUsage')MB"

    # Provider status
    echo ""
    echo "Provider Status:"
    curl -s http://localhost:8080/api/ai/providers | jq -r
'.providers[] | "  \(.name): \(.status) (\(.healthScore))"'

    sleep 5
done
```

Log Analysis Tool

```
#!/bin/bash
# analyze-logs.sh

echo "📋 Log Analysis"
echo "==============="

# Recent errors
echo "Recent Errors:"
docker logs ai-routing-service --since=1h 2>&1 | grep -i error | tail -10

echo ""
echo "Response Time Analysis:"
docker logs ai-routing-service --since=1h 2>&1 | grep "response_time" | \
    awk '{print $NF}' | sort -n | awk '
    BEGIN { sum = 0; count = 0; }
    { sum += $1; count++; values[count] = $1; }
    END {
        if (count > 0) {
            mean = sum / count;
            median = (count % 2) ? values[(count + 1) / 2] : (values[count / 2] +
values[count / 2 + 1]) / 2;
            print "Mean: " mean "ms";
            print "Median: " median "ms";
            print "Min: " values[1] "ms";
            print "Max: " values[count] "ms";
        }
    }'

echo ""
echo "Top Error Types:"
docker logs ai-routing-service --since=1h 2>&1 | grep -i error | \
    sed 's/.*error_type":"([^"]*).*/\1/' | sort | uniq -c | sort -nr | head -5
```

Monitoring and Alerts

Alert Configuration

```
// Configure alerts for critical issues
const alertConfig = {
  latency: {
    threshold: 5000, // 5 seconds
    severity: 'warning',
    action: 'email_team'
  },
  errorRate: {
    threshold: 0.05, // 5%
    severity: 'critical',
    action: 'page_oncall'
  },
  providerFailure: {
    threshold: 1, // Any provider failure
    severity: 'high',
    action: 'slack_alert'
  },
  budgetExceeded: {
    threshold: 0.9, // 90% of budget
    severity: 'medium',
    action: 'email_admin'
  }
};
```

Monitoring Dashboard Setup

```
# Set up Grafana dashboard
curl -X POST http://localhost:3000/api/dashboards/db \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer $GRAFANA_API_KEY" \
     -d @monitoring-dashboard.json

# Configure Prometheus targets
echo "
- job_name: 'ai-routing-service'
  static_configs:
    - targets: ['localhost:8080']
  metrics_path: /metrics
  scrape_interval: 15s
" >> /etc/prometheus/prometheus.yml
```

Advanced Troubleshooting

Debug Mode Activation

```
# Enable debug logging
curl -X POST http://localhost:8080/api/ai/debug/enable

# Set specific log levels
curl -X POST http://localhost:8080/api/ai/debug/log-level \
     -H "Content-Type: application/json" \
     -d '{"level": "debug", "component": "routing"}'

# Enable request tracing
curl -X POST http://localhost:8080/api/ai/debug/trace \
     -H "Content-Type: application/json" \
     -d '{"enabled": true, "sampleRate": 0.1}'
```

Performance Profiling

```
# Start CPU profiling
curl -X POST http://localhost:8080/api/ai/profile/cpu/start

# Run load test
ab -n 1000 -c 50 http://localhost:8080/api/ai/test-endpoint

# Stop and get CPU profile
curl -X POST http://localhost:8080/api/ai/profile/cpu/stop > cpu-profile.json

# Start memory profiling
curl -X POST http://localhost:8080/api/ai/profile/memory/start

# Take heap snapshot
curl -X POST http://localhost:8080/api/ai/profile/memory/snapshot > heap-snapshot.json
```

Database Troubleshooting

```
-- Check for long-running queries
SELECT query, state, query_start, now() - query_start as runtime
FROM pg_stat_activity
WHERE state = 'active' AND now() - query_start > interval '30 seconds';

-- Analyze query performance
EXPLAIN ANALYZE SELECT * FROM ai_requests
WHERE user_id = 'test-user' AND created_at > NOW() - INTERVAL '1 day';

-- Check index usage
SELECT schemaname, tablename, attname, n_distinct, correlation
FROM pg_stats
WHERE tablename = 'ai_requests';
```

Support Resources

Getting Help

1. **Documentation**: Comprehensive guides and API references
2. **Community Forums**: Developer community discussions

3. **Issue Tracker**: Report bugs and feature requests
4. **Support Email**: Direct technical support
5. **Emergency Hotline**: Critical issue escalation

Useful Commands

```
# Generate diagnostic report
curl http://localhost:8080/api/ai/diagnostics > diagnostic-report.json

# Export configuration
curl http://localhost:8080/api/ai/config/export > current-config.json

# Download recent logs
curl http://localhost:8080/api/ai/logs?hours=1 > recent-logs.txt

# Performance test
./scripts/performance-test.sh

# System information
curl http://localhost:8080/api/ai/system-info
```

Emergency Procedures

1. **Service Down**:

```
# Quick restart
docker-compose restart ai-routing-service

# Full system restart
docker-compose down && docker-compose up -d

# Rollback to previous version
docker-compose -f docker-compose.prod.yml up -d --scale ai-routing-service=0
docker-compose -f docker-compose.prod.yml up -d --scale ai-routing-service=1
```

1. **Database Issues**:

```
# Check database connection
psql -h localhost -U ai_platform -d ai_platform -c "SELECT 1"

# Restart database
docker-compose restart postgres

# Run database health check
./scripts/db-health-check.sh
```

1. **Provider Outages**:

```
# Disable problematic provider
curl -X POST http://localhost:8080/api/ai/providers/openai/disable

# Enable fallback mode
curl -X POST http://localhost:8080/api/ai/fallback/enable

# Monitor recovery
watch -n 10 'curl -s http://localhost:8080/api/ai/providers | jq ".providers[] |
{name: .name, status: .status}"'
```

For additional support or to report issues not covered in this guide, contact the development team or create an issue in the project repository.

Last updated: August 2025
Version: 3.15.0