Multi-AI Diagnostics System

Overview

The Multi-AI Diagnostics System enhances the Sports Bar TV Controller with advanced AI consultation capabilities. It queries multiple AI models (OpenAI, Anthropic Claude, Google Gemini, xAI Grok) in parallel, compares their responses, synthesizes consensus answers, and flags disagreements for both interactive chatbot queries and automated diagnostics.

Features

1. Multi-AI Consultation

- **Parallel Queries**: Queries all available AI models simultaneously for faster responses
- **Consensus Synthesis**: Combines responses from multiple AIs into a unified answer
- **Disagreement Detection**: Identifies when AIs provide conflicting recommendations
- **Voting System**: Uses majority voting for critical decisions
- **Confidence Scoring**: Calculates confidence levels for recommendations

2. Integration Points

A. AI Chatbot (Interactive)

- Location: `/diagnostics` → AI Assistant tab
- Features:
- Side-by-side AI responses
- Consensus summary at top
- Disagreement highlights
- Voting visualization
- Confidence meters
- Individual AI response cards

B. Automated Diagnostics

- **Deep Diagnostics** (`scripts/diagnostics/deep-diagnostics.js`)
- Runs Sunday 5 AM
- Consults AI for medium+ severity issues
- Logs all AI consultations to database
- **Self-Healing** (`scripts/diagnostics/self-healing.js`)
- Consults AI before taking critical actions
- Requires majority consensus for automatic fixes
- Minimum confidence threshold enforcement
- Manual review flagging for low confidence

Configuration

Environment Variables

Add these to your `.env` file:

```
# OpenAI Configuration
OPENAI_API_KEY=sk-...
OPENAI_MODEL=gpt-4-turbo-preview

# Anthropic (Claude) Configuration
ANTHROPIC_API_KEY=sk-ant-...
ANTHROPIC_MODEL=claude-3-5-sonnet-20241022

# Google (Gemini) Configuration
GOOGLE_API_KEY=...
GOOGLE_MODEL=gemini-1.5-pro

# xAI (Grok) Configuration
GROK_API_KEY=xai-...
GROK_MODEL=grok-beta

# Multi-AI Settings
USE_MULTI_AI=true                    # Enable/disable multi-AI consultation
REQUIRE_AI_CONSENSUS=true            # Require majority vote for auto-fixes
MIN_AI_CONFIDENCE=0.6                # Minimum confidence threshold (0-1)
AI_CONSULTATION_THRESHOLD=medium     # Severity level to trigger AI (low/medium/high/
critical)
```

Getting API Keys

1. **OpenAI**: https://platform.openai.com/api-keys
2. **Anthropic**: https://console.anthropic.com/
3. **Google AI**: https://makersuite.google.com/app/apikey
4. **xAI (Grok)**: https://console.x.ai/

Architecture

Core Components

1. Multi-AI Consultant (`src/lib/multi-ai-consultant.ts`)

Main service that orchestrates AI consultations:
- Manages API clients for all providers
- Executes parallel queries
- Analyzes and synthesizes responses
- Detects disagreements
- Performs voting

Key Methods:

```
consultAll(query, systemPrompt, context): Promise<MultiAIResult>
getAvailableProviders(): string[]
isAvailable(): boolean
```

2. Multi-AI Response Component (`src/components/MultiAIResponse.tsx`)

React component for displaying multi-AI results:
- Consensus tab with synthesized answer
- Individual responses tab with expandable cards
- Analysis tab with statistics and metrics
- Voting visualization
- Disagreement flags

3. Enhanced Diagnostics Chat API (`src/app/api/chat/diagnostics/route.ts`)

Updated to use multi-AI consultation:
- Detects available AI providers
- Falls back to rule-based responses if no AI configured
- Returns structured multi-AI results

4. Enhanced Diagnostics Scripts

- **deep-diagnostics.js**: Consults AI for critical issues
- **self-healing.js**: Uses AI consensus for fix decisions

Data Flow

```
User Query / Automated Check
         ↓
Multi-AI Consultant
         ↓
    ┌────┴────┬────────┬────────┐
    ↓         ↓        ↓        ↓
  OpenAI  Anthropic  Google   Grok
    ↓         ↓        ↓        ↓
    └────┬────┴────────┴────────┘
         ↓
  Response Analysis
    - Consensus Synthesis
    - Disagreement Detection
    - Voting
    - Confidence Scoring
         ↓
  Structured Result
         ↓
  UI Display / Action Decision
```

Usage Examples

Interactive Chatbot

1. Navigate to `/diagnostics`
2. Click “AI Assistant” tab
3. Ask a question: “Should I restart the system?”
4. View multi-AI response with:
   - Consensus summary
   - Individual AI opinions
   - Voting results
   - Disagreement flags (if any)

Automated Diagnostics

The system automatically consults AI when:
- Deep diagnostics detects medium+ severity issues
- Self-healing considers critical actions
- Confidence threshold is met

Example log output:

```
🤖 Consulting AI models for critical issues...
Found 2 issue(s) requiring AI consultation

Consulting AI about: High CPU Usage
✅ Multi-AI consultation completed:
   - Providers: 4/4
   - Agreement: majority
   - Confidence: 85%
   🔧 Majority recommends: Restart System
```

Response Structure

MultiAIResult

```
{
  query: string,
  timestamp: Date,
  responses: AIResponse[],      // Individual AI responses
  consensus: ConsensusResult,   // Synthesized answer
  disagreements: DisagreementFlag[],
  voting: VotingResult,
  summary: {
    totalProviders: number,
    successfulResponses: number,
    failedResponses: number,
    averageConfidence: number,
    processingTime: number
  }
}
```

Agreement Levels

- **unanimous**: All AIs agree (>80% similarity)
- **majority**: Most AIs agree (>60% similarity)
- **split**: Mixed opinions (>40% similarity)
- **no-consensus**: Significant disagreement (<40% similarity)

Database Schema

AIConsultation Table

```sql
CREATE TABLE AIConsultation (
  id TEXT PRIMARY KEY,
  timestamp DATETIME,
  issueId TEXT,
  query TEXT,
  result TEXT,              -- JSON of full MultiAIResult
  providers TEXT,           -- JSON array of providers used
  consensusLevel TEXT,      -- unanimous/majority/split/no-consensus
  confidence REAL,
  hasDisagreements INTEGER,
  votingWinner TEXT,
  processingTime INTEGER
)
```

Enhanced Issue Table

```sql
ALTER TABLE Issue ADD COLUMN aiRecommendation TEXT;
ALTER TABLE Issue ADD COLUMN aiConfidence REAL;
ALTER TABLE Issue ADD COLUMN needsReview INTEGER;
```

Enhanced Fix Table

```sql
ALTER TABLE Fix ADD COLUMN aiRecommended INTEGER;
ALTER TABLE Fix ADD COLUMN aiConfidence REAL;
```

Safety Features

1. Confidence Thresholds

- Minimum confidence required for automatic actions
- Default: 0.6 (60%)
- Configurable via `MIN_AI_CONFIDENCE`

2. Consensus Requirements

- Can require majority vote for critical actions
- Enabled via `REQUIRE_AI_CONSENSUS=true`
- Prevents action if no majority agreement

3. Manual Review Flagging

- Low confidence recommendations flagged for review
- Disagreements highlighted for human decision
- Issues marked with `needsReview` flag

4. Fallback Mechanisms

- Rule-based responses if no AI configured
- Graceful degradation on API failures
- Individual AI failures don’t block others

Performance

Response Times

- Parallel queries: ~2-5 seconds (all AIs)
- Sequential would be: ~8-20 seconds
- Caching: Not implemented (real-time analysis)

Cost Optimization

- Only consult AI for critical issues
- Configurable severity threshold
- Batch similar queries when possible

Rate Limiting

- Respects individual provider limits
- Automatic retry with exponential backoff
- Error handling for quota exceeded

Monitoring

Metrics Tracked

- AI consultation count
- Provider success rates
- Average confidence scores
- Disagreement frequency
- Processing times
- Cost per consultation (if tracked)

Logs

All AI consultations logged to:
- Database: `AIConsultation` table
- Console: Detailed output during diagnostics
- Files: Standard application logs

Troubleshooting

No AI Responses

Problem: Chatbot shows “No AI configured” message
Solution:
1. Check `.env` file has at least one API key
2. Verify API key format is correct
3. Test API key with provider’s test endpoint
4. Check server logs for authentication errors

Low Confidence Scores

Problem: AI recommendations have low confidence
Solution:
1. Provide more context in queries

2. Check if issue description is clear
3. Verify system metrics are being collected
4. Consider lowering `MIN_AI_CONFIDENCE` threshold

Disagreements

Problem: AIs frequently disagree
Solution:
1. Review issue descriptions for ambiguity
2. Check if system context is complete
3. Consider if issue truly requires human judgment
4. Adjust `REQUIRE_AI_CONSENSUS` setting

API Failures

Problem: Some AI providers fail
Solution:
1. Check API key validity
2. Verify network connectivity
3. Check provider status pages
4. Review rate limits and quotas
5. Check server logs for specific errors

Best Practices

1. Query Formulation

- Be specific and detailed
- Include relevant context
- Ask clear, actionable questions
- Provide system metrics when available

2. Threshold Configuration

- Start with default thresholds
- Adjust based on your risk tolerance
- Higher thresholds = more manual review
- Lower thresholds = more automation

3. Provider Selection

- Use at least 2 providers for voting
- Mix different AI architectures
- Consider cost vs. quality tradeoffs
- Test each provider’s strengths

4. Review Process

- Regularly review flagged issues
- Monitor disagreement patterns
- Adjust thresholds based on outcomes
- Document manual decisions

Future Enhancements

Planned Features

1. **Learning from Outcomes**: Track which AI recommendations worked
2. **Provider Weighting**: Weight votes based on historical accuracy
3. **Cost Tracking**: Monitor API usage and costs
4. **Custom Prompts**: Allow customization of system prompts
5. **Response Caching**: Cache similar queries to reduce costs
6. **A/B Testing**: Compare AI recommendations vs. rule-based
7. **Sentiment Analysis**: Detect urgency in AI responses
8. **Multi-Language**: Support non-English diagnostics

Experimental Features

- **Chain-of-Thought**: Ask AIs to explain reasoning
- **Self-Critique**: Have AIs review each other's answers
- **Ensemble Methods**: Advanced consensus algorithms
- **Fine-Tuning**: Train models on historical diagnostics

API Reference

getMultiAIConsultant()

Returns singleton instance of MultiAIConsultant

```
const consultant = getMultiAIConsultant();
```

consultAll(query, systemPrompt?, context?)

Queries all available AI models

```
const result = await consultant.consultAll(
  "Should I restart the system?",
  SYSTEM_PROMPT,
  { activeIssues, systemMetrics }
);
```

getAvailableProviders()

Returns list of configured providers

```
const providers = consultant.getAvailableProviders();
// ['OpenAI', 'Anthropic', 'Google']
```

isAvailable()

Checks if any AI provider is configured

```
if (consultant.isAvailable()) {
  // Use multi-AI
} else {
  // Fallback to rules
}
```

Support

For issues or questions:
1. Check server logs: `/home/ubuntu/Sports-Bar-TV-Controller/logs/`
2. Review diagnostics: `/diagnostics` page
3. Check database: `AIConsultation` table
4. GitHub Issues: [Repository URL]

License

Same as Sports Bar TV Controller project

Credits

Developed as part of the Sports Bar TV Controller diagnostics enhancement project.