🤖 AI Auto-Fix System Documentation

Table of Contents

Overview

The AI Auto-Fix System is an intelligent testing framework that automatically analyzes test failures and generates fixes using multiple AI providers. It features persistent memory, cross-session learning, and support for both local and cloud-based AI models.

Key Features

- 🧠 **Multi-Provider Support**: Choose between Claude, Qwen, and other AI models
- 💾 **Persistent Memory**: Learn from previous fixes and build institutional knowledge
- 🔄 **Test-Fix-Retest Loop**: Automatically verify fixes and continue until all tests pass
- 📊 **Success Tracking**: Monitor fix success rates and identify patterns
- 🏠 **Local & Cloud Options**: Privacy-focused local models or powerful cloud AI
- 🔧 **Extensible Architecture**: Easy to add new AI providers

Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                     AI Auto-Fix System                      │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  ┌─────────────┐    ┌──────────────┐    ┌─────────────┐     │
│  │    Test     │───▶│   Detect     │───▶│   Analyze   │     │
│  │   Runner    │    │   Issues     │    │   with AI   │     │
│  └─────────────┘    └──────────────┘    └─────────────┘     │
│                                                │            │
│  ┌─────────────┐    ┌──────────────┐    ┌─────────────┐     │
│  │   Verify    │◀───│    Apply     │◀───│  Generate   │     │
│  │    Fix      │    │     Fix      │    │    Fix      │     │
│  └─────────────┘    └──────────────┘    └─────────────┘     │
│                                                             │
├─────────────────────────────────────────────────────────────┤
```

```
│                  AI Provider Layer                         │
├────────────────────────────────────────────────────────────┤
│                                                            │
│  ┌─────────────┐                 ┌─────────────┐           │
│  │    Qwen     │                 │   Claude    │           │
│  │ (Local AI)  │                 │ (Cloud AI)  │           │
│  └─────────────┘                 └─────────────┘           │
├────────────────────────────────────────────────────────────┤
│               Persistent Memory System                     │
├────────────────────────────────────────────────────────────┤
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐        │
│  │ Fix History │  │  Patterns   │  │ Knowledge   │        │
│  │  Database   │  │ & Insights  │  │    Base     │        │
│  └─────────────┘  └─────────────┘  └─────────────┘        │
└────────────────────────────────────────────────────────────┘
```

Available AI Providers

1. DeepSeek Coder (Default)

- **Type**: Local AI model via Ollama
- **Strengths**:
  - Specialized for code debugging and analysis
  - Strong problem-solving capabilities
  - Complete privacy (no data leaves your machine)
  - No API costs after initial setup
  - Excellent at understanding error patterns
- **Best for**: Code issues, debugging, systematic problem analysis

2. Qwen 2.5 Coder

- **Type**: Local AI model via Ollama
- **Strengths**:
  - Code-specialized for programming tasks
  - Fast response times
  - Complete privacy (no data leaves your machine)
  - No API costs after initial setup
- **Best for**: Quick fixes, code generation, offline environments

3. Claude 3.5 Sonnet

- **Type**: Cloud AI service by Anthropic
- **Strengths**:
  - Advanced reasoning capabilities
  - Deep context understanding
  - Complex problem analysis
  - Comprehensive solution generation
- **Best for**: Complex issues, detailed analysis, when internet is available

4. Extensible for Future Providers

The system is designed to easily support additional AI providers like:

- GPT models via OpenAI API
- Gemini via Google AI
- Other local models via Ollama
- Custom AI endpoints

Quick Start Guide

1. Basic Auto-Fix (Default DeepSeek)

```
./test.sh --auto-fix
```

2. Choose Specific AI Provider

```
# Use Claude for advanced analysis
./test.sh --auto-fix --fixer=claude

# Use Qwen for fast fixes
./test.sh --auto-fix --fixer=qwen

# Use DeepSeek explicitly (default)
./test.sh --auto-fix --fixer=deepseek
```

3. Check Available Fixers

```
./ai_fixers.sh list
```

4. View Fixer Information

```
./ai_fixers.sh info deepseek
./ai_fixers.sh info qwen
./ai_fixers.sh info claude
```

Setup Instructions

Setting Up DeepSeek Coder (Local, Default)

Prerequisites

- Linux/macOS system with 8GB+ RAM
- Internet connection for initial model download

Installation Steps

1. **Install Ollama**:

```
curl -fsSL https://ollama.ai/install.sh | sh
```

2. **Install DeepSeek Model**:

```
# For systems with 24GB+ RAM (recommended)
ollama pull deepseek-coder:33b

# For systems with 8GB+ RAM (default)
ollama pull deepseek-coder:6.7b

# For systems with 4GB+ RAM
ollama pull deepseek-coder:1.3b
```

3. **Verify Installation**:

```
ollama list
./ai_fixers.sh list
```

System Requirements

- **Minimum**: 4GB RAM, 3GB disk space (1.3b model)
- **Recommended**: 8GB RAM, 8GB disk space (6.7b model)
- **Optimal**: 24GB RAM, 20GB disk space (33b model)

Setting Up Qwen 2.5 Coder (Alternative Local Option)

Installation Steps

1. **Install Ollama** (if not already installed):

```
curl -fsSL https://ollama.ai/install.sh | sh
```

2. **Install Qwen Model**:

```
# For systems with 16GB+ RAM (recommended)
ollama pull qwen2.5-coder:32b

# For systems with 8GB+ RAM
ollama pull qwen2.5-coder:7b
```

3. **Verify Installation**:

```
ollama list
./ai_fixers.sh list
```

System Requirements

- **Minimum**: 8GB RAM, 5GB disk space
- **Recommended**: 16GB RAM, 10GB disk space
- **Optimal**: 32GB RAM, 20GB disk space

Setting Up Claude (Cloud)

Prerequisites

- Internet connection
- Anthropic account with API access
- Python 3.6+ with requests package

Installation Steps

1. **Create Anthropic Account**:
   - Visit: https://console.anthropic.com/
   - Sign up or log in
   - Navigate to "API Keys" section
2. **Generate API Key**:
   - Click "Create Key"
   - Copy the generated key
   - Keep it secure (treat like a password)
3. **Set Environment Variable**:

```
# Temporary (current session only)
export ANTHROPIC_API_KEY="your-api-key-here"

# Permanent (add to shell profile)
echo 'export ANTHROPIC_API_KEY="your-api-key-here"' >> ~/.bashrc
source ~/.bashrc
```

4. **Install Python Dependencies**:

```
pip3 install requests
```

5. **Verify Setup**:

```
./ai_fixers.sh list
```

Cost Considerations

- Claude API usage incurs costs based on tokens processed
- Typical fix analysis: $0.01 - $0.10 per issue
- Monitor usage at: https://console.anthropic.com/

Usage Examples

Example 1: Basic Testing with Auto-Fix

```
# Run tests with automatic fixing enabled
./test.sh --auto-fix

# Expected output:
[INFO] Auto-fix mode: ENABLED
[INFO] AI Fixer: deepseek
🤖 AI-powered intelligent auto-fix enabled (using: deepseek)
🧠 Memory: 15 fixes tried, 87.2% success rate
🔍 deepseek analyzing: deepseek-r1:7b
✅ deepseek successfully fixed: deepseek-r1:7b
🎉 ALL MODELS CONFIRMED WORKING!
```

Example 2: Using Different AI Providers

```
# Use Claude for complex analysis
./test.sh --auto-fix --fixer=claude

# Use Qwen for fast local fixing
./test.sh --auto-fix --fixer=qwen

# Use DeepSeek for debugging (default)
./test.sh --auto-fix --fixer=deepseek

# Use specific date for test results
./test.sh --auto-fix --fixer=claude --date=2025-01-15
```

Example 3: Managing AI Fixers

```
# List all available fixers
./ai_fixers.sh list

# Get detailed information about a fixer
./ai_fixers.sh info qwen

# Test a fixer directly on an issue file
./ai_fixers.sh fix Tests/2025-01-15/model-name/ai_issue.json deepseek
```

Example 4: Memory System Management

```
# View memory statistics
./memory.sh stats

# Export insights report
./memory.sh export my_insights.json

# View recent fix activity
./memory.sh recent 7

# Show successful strategies
./memory.sh successful
```

Memory & Learning System

Overview

The AI Auto-Fix System features a sophisticated memory system that learns from every fix attempt, building institutional knowledge over time.

Components

1. Fix History Database

- **Storage**: SQLite database (`AIMemory/ai_memory.db`)
- **Records**: Every fix attempt with full context
- **Fields**:
  - Timestamp and execution time
  - Model name and issue type
  - AI provider used (deepseek, qwen, claude, etc.)
  - Analysis and fix commands
  - Success/failure status
  - Verification results

2. Pattern Recognition

- **Issue Signatures**: Unique fingerprints for problem types

- **Model Characteristics**: Learning model-specific quirks
- **Success Patterns**: Tracking what works best
- **Failure Analysis**: Understanding why fixes fail

3. Historical Context

When analyzing new issues, the system provides AI fixers with:

- **Similar Issues**: Exact matches and related problems
- **Model History**: Past performance for the specific model
- **Successful Strategies**: Proven solutions for similar issues
- **Failure Patterns**: What to avoid based on past attempts

4. Database Schema Migration

The system has been updated to use a generic `ai_analysis` column instead of the provider-specific `claude_analysis` to support all AI providers:

- **Backward Compatibility**: Automatically migrates from old schema
- **Provider Agnostic**: Works with any AI provider (DeepSeek, Qwen, Claude, etc.)
- **Error Handling**: Robust JSON parsing with fallback for malformed data

Learning Progression

- **First Run**: Basic rule-based fixes only
- **After 5 fixes**: AI has initial patterns to reference
- **After 20 fixes**: AI can identify recurring issues and optimal solutions
- **After 50+ fixes**: AI becomes highly effective with deep institutional knowledge

Memory Commands

```
# View statistics
./memory.sh stats

# Model-specific insights
./memory.sh model deepseek-r1:7b

# Export complete knowledge base
./memory.sh export insights.json

# View recent activity
./memory.sh recent 7

# Show successful strategies
./memory.sh successful

# Identify patterns
./memory.sh patterns
```

```
# Clean up old records
./memory.sh cleanup 30
```

Advanced Configuration

Environment Variables

```
# Claude API configuration
export ANTHROPIC_API_KEY="your-key-here"

# Auto-fix settings
export AUTO_FIX=true
export FIXER_TYPE=deepseek

# Debug mode
export CLAUDE_DEBUG=1
```

Custom Model Configuration

For Qwen, you can specify different models:

```
# Use specific Qwen model
./Scripts/AutoFixers/qwen_autofix.py issue.json qwen2.5-coder:7b
```

Memory System Configuration

The memory system can be customized by modifying `Scripts/ai_memory.py`:

- Database location
- Retention policies
- Pattern matching algorithms
- Knowledge base structure

Adding New AI Providers

1. **Create Provider File**:

   ```
   # Scripts/AutoFixers/your_provider_autofix.py
   class YourProviderAutoFixer:
       def analyze_issue(self, issue_data):
           # Implement analysis logic
           pass

       def apply_fix(self, fix_data):
           # Implement fix application
           pass
   ```

```
    def verify_fix(self, issue_data, fix_data):
        # Implement verification
        pass
```

2. **Update Manager**:

```
# Add to Scripts/AutoFixers/autofix_manager.py
elif self.fixer_type == "your_provider":
    from your_provider_autofix import YourProviderAutoFixer
    return YourProviderAutoFixer()
```

3. **Add to Detection**:

```
# Update list_available_fixers() method
your_provider_available = check_your_provider_availability()
fixers.append({
    "name": "your_provider",
    "display_name": "Your Provider",
    # ... other details
})
```

Troubleshooting

Common Issues and Solutions

Qwen Issues

Problem: "Qwen model not available"

```
# Check Ollama installation
ollama --version

# Install Ollama if missing
curl -fsSL https://ollama.ai/install.sh | sh

# Check available models
ollama list

# Install Qwen model
ollama pull qwen2.5-coder:7b
```

Problem: "Out of memory" errors

```
# Use smaller model
ollama pull qwen2.5-coder:7b

# Check system resources
free -h
```

Claude Issues

Problem: "API call failed"

```
# Check API key
echo $ANTHROPIC_API_KEY

# Verify key is valid
curl -H "x-api-key: $ANTHROPIC_API_KEY"
https://api.anthropic.com/v1/messages

# Check account credits at console.anthropic.com
```

Problem: "requests module not found"

```
# Install requests
pip3 install requests

# Or use conda
conda install requests
```

DeepSeek Issues

Problem: "DeepSeek model not available"

```
# Check Ollama installation
ollama --version

# Install Ollama if missing
curl -fsSL https://ollama.ai/install.sh | sh

# Check available models
ollama list

# Install DeepSeek model
ollama pull deepseek-coder:6.7b
```

Problem: "JSON parsing errors" or "Invalid control character"

```
# This indicates special characters in test data
# The system now handles this automatically with Python JSON encoding
# If issues persist, check for corrupted test files
```

Problem: "Database schema error: no such column"

```
# The system automatically migrates old database schemas
# If migration fails, backup and recreate:
mv AIMemory/ai_memory.db AIMemory/ai_memory.db.backup
# System will create new database on next run
```

General Issues

Problem: "No fixers available"

```
# Check fixer status
./ai_fixers.sh list

# Set up at least one fixer following setup guides
```

Problem: "Permission denied"

```
# Make scripts executable
chmod +x Scripts/test.sh
chmod +x ai_fixers.sh
chmod +x memory.sh
```

Problem: "Bash arithmetic error" or function return value issues

```
# This was fixed in recent updates - ensure you have latest version
# Error occurred when log output mixed with return values
# Now all logs are redirected to stderr properly
```

Debug Mode

Enable debug mode for detailed logging:

```
export CLAUDE_DEBUG=1
./test.sh --auto-fix
```

Log Files

Check these locations for detailed logs:

- `Tests/YYYY-MM-DD/*/ai_issue.json` - AI analysis data
- `AIMemory/ai_memory.db` - Fix history database
- `AIMemory/knowledge_base.json` - Accumulated knowledge

API Reference

AutoFixManager Class

```python
from Scripts.AutoFixers.autofix_manager import AutoFixManager

# Initialize with specific fixer
manager = AutoFixManager("deepseek")  # or "qwen" or "claude"

# Analyze an issue
fix_data = manager.analyze_issue(issue_data)

# Apply generated fix
success = manager.apply_fix(fix_data)

# Verify fix worked
verified = manager.verify_fix(issue_data, fix_data)

# Get fixer information
info = manager.get_fixer_info()

# List available fixers
fixers = AutoFixManager.list_available_fixers()
```

Memory System API

```python
from Scripts.ai_memory import AIMemory

# Initialize memory system
memory = AIMemory()

# Record a fix attempt
fix_id = memory.record_fix_attempt(
    issue_data, ai_response, fix_success,
    verification_success, execution_time,
    notes, fixer_type
)

# Query similar issues
similar = memory.query_similar_issues(issue_data)

# Get model insights
```

```
insights = memory.get_model_insights("model-name")

# Build context for AI
context = memory.build_context_for_ai(issue_data)

# Get statistics
stats = memory.get_memory_stats()

# Export insights
memory.export_insights("output.json")
```

Issue Data Format

```
{
    "model": "model-name",
    "issue_type": "TIMEOUT|MODEL_NOT_AVAILABLE|UNEXPECTED_RESPONSE",
    "description": "Human-readable description",
    "error_output": "Raw error text",
    "test_prompt": "Prompt that was sent to model",
    "expected_pattern": "Regex pattern for expected response",
    "actual_response": "What the model actually returned",
    "test_environment": {
        "system_info": "Additional context"
    }
}
```

Fix Data Format

```
{
    "analysis": "Root cause analysis from AI",
    "fix_type":
"model_config|prompt_adjustment|system_fix|model_reinstall",
    "fix_commands": ["array", "of", "bash", "commands"],
    "verification_steps": ["how", "to", "verify", "fix"],
    "confidence": 0.95,
    "expected_outcome": "What should happen after fix"
}
```

📚 This documentation covers the complete AI Auto-Fix System. For additional help, check the troubleshooting section or examine the source code in `Scripts/AutoFixers/`.