📚 AI Model Testing Framework Documentation

Welcome to the comprehensive documentation for the AI Model Testing Framework with multi-provider auto-fix capabilities.

Overview

The AI Model Testing Framework is an intelligent testing system that automatically tests AI models, detects issues, and applies fixes using multiple AI providers. It features persistent memory, cross-session learning, and support for both local and cloud-based AI models.

📖 Documentation Structure

Core System Documentation

1. **AI Auto-Fix System** - Complete guide to the multi-provider auto-fix system
    - Available AI providers (DeepSeek, Qwen, Claude)
    - Setup and configuration
    - Usage examples and troubleshooting
2. **Memory & Learning System** - Persistent memory and learning capabilities
    - Database schema and architecture
    - Memory operations and CLI interface
    - Analytics and data export
3. **Test Framework** - Core testing framework documentation
    - Test-fix-retest loop architecture
    - Configuration and customization
    - Advanced features and integration

🚀 Quick Start

1. Basic Setup

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

# Install default AI model (DeepSeek Coder)
ollama pull deepseek-coder:6.7b

# Verify setup
./ai_fixers.sh list
```

2. Run Tests with Auto-Fix

```
# Run with default fixer (DeepSeek)
./test.sh --auto-fix

# Use specific AI provider
./test.sh --auto-fix --fixer=claude
./test.sh --auto-fix --fixer=qwen
./test.sh --auto-fix --fixer=deepseek
```

3. Memory Management

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

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

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

🤖 AI Providers

Provider	Type	Strengths	Best For
DeepSeek Coder (Default)	Local	Code debugging, error analysis	Systematic problem solving
Qwen 2.5 Coder	Local	Fast processing, code generation	Quick fixes, offline work
Claude 3.5 Sonnet	Cloud	Advanced reasoning, complex analysis	Complex issues, detailed analysis

🔧 Key Features

- **Multi-Provider Support**: Choose between DeepSeek, Qwen, and Claude
- **Persistent Memory**: Learn from previous fixes and build institutional knowledge
- **Test-Fix-Retest Loop**: Automatically verify fixes and continue until success
- **Success Tracking**: Monitor fix success rates and identify patterns
- **Privacy Options**: Local models for privacy-sensitive environments
- **Extensible Architecture**: Easy to add new AI providers

📁 Directory Structure

```
Builder/
├── Scripts/
│   ├── AutoFixers/           # AI provider implementations
│   │   ├── autofix_manager.py    # Multi-provider manager
│   │   ├── deepseek_autofix.py   # DeepSeek integration
│   │   ├── qwen_autofix.py       # Qwen integration
│   │   └── claude_autofix.py     # Claude integration
│   ├── ai_memory.py          # Memory system
│   ├── memory_cli.sh         # Memory CLI interface
│   └── test.sh               # Main test framework
├── Documentation/            # This documentation
├── AIMemory/                # Persistent storage
├── Tests/                   # Test results
├── ai_fixers.sh            # AI fixer management
└── memory.sh               # Memory system shortcut
```

🔄 Workflow

1. **Discovery**: Detect available models and system capabilities
2. **Testing**: Execute tests on all models with timeout and pattern matching
3. **Analysis**: Identify failed models and categorize issue types
4. **Fixing**: Apply AI-powered fixes with historical context
5. **Verification**: Re-test fixed models and confirm resolution
6. **Learning**: Store results in persistent memory for future use

🛠 Configuration

Environment Variables

```
# Auto-fix configuration
export AUTO_FIX=true
export FIXER_TYPE=deepseek

# Test configuration
export MAX_ITERATIONS=5
export TIMEOUT_DURATION=30

# Claude API (if using Claude)
export ANTHROPIC_API_KEY="your-key-here"

# Debug mode
export CLAUDE_DEBUG=1
```

Command Line Options

```
# Test framework options
./test.sh [--auto-fix] [--fixer=TYPE] [--date=YYYY-MM-DD] [--help]

# AI fixer management
./ai_fixers.sh [list|info|fix] [arguments]

# Memory system
```

```
./memory.sh [stats|model|export|recent|successful|patterns|cleanup]
[arguments]
```

📊 Memory & Analytics

The system maintains comprehensive statistics and learns from every interaction:

- **Fix Success Rates**: Track effectiveness by AI provider and issue type
- **Model Performance**: Monitor model-specific patterns and behaviors
- **Historical Context**: Provide AI fixers with relevant past experiences
- **Knowledge Accumulation**: Build institutional knowledge over time

🔐 Security & Privacy

Local Providers (DeepSeek, Qwen)

- ✅ All processing happens locally
- ✅ No data sent to external services
- ✅ Complete privacy and control

Cloud Provider (Claude)

- ⚠ Issue data is sent to Anthropic's servers for analysis
- ✅ Encrypted API communications
- ✅ Strong privacy policies from provider

🤝 Contributing

To add support for additional AI providers:

1. Create a new fixer file in `Scripts/AutoFixers/`
2. Implement the standard interface:
   - `analyze_issue(issue_data)`
   - `apply_fix(fix_data)`
   - `verify_fix(issue_data, fix_data)`
3. Update `autofix_manager.py` to include your new fixer
4. Add setup instructions to documentation

📞 Support & Troubleshooting

Common Issues

- **No models available**: Install Ollama and pull models
- **Permission denied**: Make scripts executable with `chmod +x`
- **Memory errors**: Use smaller models or increase system RAM
- **API failures**: Check API keys and internet connectivity
- **Database errors**: System auto-migrates schema from old versions
- **JSON parsing errors**: Fixed with improved special character handling
- **Function return errors**: Fixed in recent updates - ensure latest version

Debug Mode

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

Log Files

- Test results: `Tests/YYYY-MM-DD/*/`
- AI analysis data: `Tests/YYYY-MM-DD/*/ai_issue.json`
- Memory database: `AIMemory/ai_memory.db`
- Knowledge base: `AIMemory/knowledge_base.json`

📈 Performance Tips

1. **Start with DeepSeek**: Good balance of capability and performance
2. **Use Qwen for speed**: Fastest local option for simple issues
3. **Escalate to Claude**: For complex problems that local models can't solve
4. **Monitor memory stats**: Use `./memory.sh stats` to track learning progress

---

🚀 **Ready to experience intelligent AI-powered testing with persistent learning!**

For detailed information on specific components, see the individual documentation files linked above.