ZachHandley / **ZMCPTools** Public

Code | Issues 2 | Pull requests | Discussions | Actions | Projects | Security and quality

... | **2** Branches | **0** Tags | Go to file | Go to file | Code

claude fix hooks modifying settings.local.json 8def46a · 9 months ago

.github	Update CODEOWNERS, rec. globa…	9 months ago
.treesummary	Fix MCP token limit issue in navigat…	9 months ago
docs	[0.2.1] Simple Context Hook to enc…	9 months ago
examples	[0.2.1] Simple Context Hook to enc…	9 months ago
migrations	Refactor knowledge graph tool sch…	9 months ago
src	fix hooks modifying settings.local.json	9 months ago
.claudeignore	.claudeignore and file operations to …	9 months ago
.gitattributes	it works!	9 months ago
.gitignore	Fix MCP token limit issue in navigat…	9 months ago
.gitmessage	Update CODEOWNERS, rec. globa…	9 months ago
CHANGELOG.md	0.2.2 Fix installation __dirname	9 months ago
CLAUDE.md	[0.2.1] Simple Context Hook to enc…	9 months ago
README.md	Add MseeP.ai badge to README.md	9 months ago
TOOL_LIST.md	Pre-commit warning unless told, pr…	9 months ago
drizzle.config.ts	0.2.0 - Plans	9 months ago
package.json	0.2.2 Fix installation __dirname	9 months ago
pnpm-lock.yaml	0.2.0 - Plans	9 months ago
pnpm-workspace.yaml	should fix tree sitter	9 months ago
tsconfig.json	Fix/Upgrade Documentation Scrap…	9 months ago
tsup.config.ts	[0.2.1] Simple Context Hook to enc…	9 months ago
vitest.config.ts	Implement automatic foundation ca…	9 months ago

README

Zmcptools
by ZachHandley
Misc

MseeP.ai Audited

ZMCPTools

License MIT | TypeScript 5.8+ | Node.js 18+ | MCP 1.15.0

🚀 **TypeScript MCP Tools for Claude Code** - Professional multi-agent orchestration platform with 61 enhanced tools, documentation intelligence, and advanced automation capabilities.

⚠️ Important Setup Note

Before spawning agents, run this command once to enable proper agent permissions:

```
claude --dangerously-skip-permissions
```

Agents run on daemon threads and need this permission to execute properly.

✨ Key Features

🎯 Multi-Agent Orchestration

- **Architect-Led Coordination**: AI architect automatically spawns and coordinates specialized agent teams
- **Intelligent Dependencies**: Agents work in proper order (Backend → Frontend → Testing → Documentation)
- **Real-Time Communication**: Agents collaborate through dedicated chat rooms with message broadcasting
- **Foundation Session Caching**: 85-90% cost reduction through automatic shared context management
- **Professional Task Management**: Create, assign, track, and monitor complex development workflows

🎨 TypeScript-First Architecture

- **Type-Safe MCP Server**: Built with Zod schemas and strict TypeScript for reliability
- **Modern CLI Interface**: Commander.js-powered CLI with structured command hierarchy
- **Development Ready**: One-command setup with hot-reload development via tsx
- **Binary Distribution**: Global access via `claude-mcp-tools` and `claude-mcp-server` commands
- **Professional Build System**: tsup-based compilation with dual CLI/server binaries

🌐 Advanced Browser Automation

- **Playwright Integration**: Professional web automation with session management
- **AI-Powered DOM Analysis**: Intelligent page structure analysis and navigation
- **Screenshot Analysis**: AI-driven visual page analysis with region focusing
- **Smart Session Management**: Auto-cleanup, session persistence, and connection pooling
- **Legacy Support**: Comprehensive tool migration with backward compatibility

📚 Documentation Intelligence & Vector Search

- **LanceDB Vector Database**: Local, high-performance semantic search with multiple embedding providers
- **Intelligent Web Scraping**: Multi-page documentation collection with automatic vectorization
- **Advanced Content Processing**: Smart URL filtering, pattern matching, and content extraction
- **Job Management**: Background worker system with status monitoring and job control
- **Documentation Sources**: Track and manage multiple documentation repositories

🧠 Knowledge Graph & Memory Systems

- **Graph-Based Knowledge Storage**: Entity-relationship modeling for cross-agent learning
- **Semantic Search**: Vector-powered knowledge discovery and relationship traversal
- **Shared Memory**: Persistent agent collaboration and insight sharing
- **Project Analysis**: Comprehensive code structure analysis with symbol extraction
- **Smart File Operations**: Pattern-based file operations with fuzzy matching

🚀 Quick Installation

Prerequisites

- **Node.js 18+**: Required for TypeScript runtime and LanceDB native bindings
- **Claude Code CLI**: Anthropic's Claude Code CLI
- **Package Manager**: npm (included), yarn, pnpm, or bun

Production Installation (Recommended)

```
# Install globally first (recommended for WSL/Linux compatibility)
pnpm add -g zmcp-tools
# If requested, approve build scripts for native dependencies
pnpm approve-builds -g

# Then install MCP integration
zmcp-tools install

# Alternative: Direct installation (may have issues with Sharp in WSL)
npx zmcp-tools@latest install     # npm
yarn dlx zmcp-tools@latest install  # yarn
bunx zmcp-tools@latest install      # bun
```

This automatically:

- ✅ Installs MCP server to `~/.mcptools/server/`
- ✅ Configures Claude Code with `claude mcp add --scope local` (current directory only)
- ✅ Sets up project permissions and CLAUDE.md integration
- ✅ Initializes SQLite database for agent coordination
- ✅ Initializes LanceDB vector database for semantic search
- ✅ Creates 61 professional MCP tools ready for use

Development Installation

```
# Clone and setup development environment
git clone https://github.com/zachhandley/ZMCPTools
cd ZMCPTools

# Quick automated setup
pnpm install && pnpm run install:global

# Or manual setup
pnpm install              # Install dependencies
pnpm build               # Compile TypeScript
pnpm link --global       # Create global symlink
zmcp-tools install       # Configure MCP integration
```

Development features:

- ✅ Global `zmcp-tools` command
- ✅ Hot-reload development: `pnpm dev`
- ✅ TypeScript compilation: `pnpm build`
- ✅ Test suite: `pnpm test`
- ✅ Full source code access and modification

MCP Server Configuration

The installer automatically configures the MCP server using `claude mcp add --scope local`. The server runs directly with Node.js:

```
# Automatically executed during installation
claude mcp add --scope local zmcp-tools node ~/.mcptools/server/index.js
```

This provides:

- ✅ Core MCP server with 43 tools (including LanceDB)
- ✅ Multi-agent orchestration capabilities
- ✅ TypeScript type safety and performance
- ✅ SQLite-based data persistence
- ✅ LanceDB vector database for semantic search
- ✅ Advanced file operations and project analysis
- ✅ Documentation intelligence with vector embeddings

- ✅ Foundation session caching for cost optimization

📋 Prerequisites

ZMCPTools TypeScript requires the following:

Required

- **Node.js 18+** - JavaScript runtime and LanceDB native bindings
- **Package Manager** - npm (included), yarn, pnpm, or bun
- **Claude CLI** - Anthropic's Claude Code CLI

Optional

- **TypeScript**: For development (`npm install -g typescript`)
- **TSX**: For development hot-reload (included in devDependencies)

Note: This TypeScript implementation includes native LanceDB vector database with no Python dependencies required.

📝 Example Commands

Here are some common workflows you can achieve with ZMCPTools:

🎨 Brand & Style Analysis

```
"Using the zmcp server, find the styles and generate me a brand guide called
THEWEBSITE_BRAND.md"
```

📚 Documentation Scraping

```
"Scrape https://modelcontextprotocol.io/introduction -- use the selector #content-area
(by ID), don't allow any subdomains, and nothing ignored (though we can ask it to ignore
regex, glob, patterns to *not* get some docs), update it weekly"
```

🤖 Multi-Agent Orchestration

```
"Create a multi-agent orchestration to design and architect a ModelContextProtocol
TypeScript server to do XYZ"
```

🔍 Project Analysis

```
"Analyze the project structure, and then search the .treesummary directory to see what's
there"
```

👥 Agent Management

```
"Spawn an agent to do X, and then monitor its progress"
```

🔎 Documentation Search

```
"Search the documentation for X"
```

🌐 Browser Automation

```
"Create a browser session, navigate to https://example.com, take a screenshot, and
analyze the page structure for accessibility issues"
```

🧠 Knowledge Graph Operations

```
"Store this implementation pattern in the knowledge graph and find related patterns
we've used before"
```

💾 Foundation Session Optimization

```
"Create a multi-agent team with foundation session 'auth-refactor-2024' to refactor
authentication across frontend and backend with 90% cost savings"
```

🔄 Development Workflow

```
"Start a dev server, run the test suite, and spawn an agent to fix any failing tests
while monitoring progress in real-time"
```

📊 Cross-Agent Learning

```
"Analyze recent agent errors, identify patterns, and spawn a debugging agent that learns
from previous failures"
```

🎯 Streamlined Plan System

```
"Create an execution plan for implementing OAuth, then execute it with coordinated
agents following the plan"
```

The Plan system provides 4 streamlined tools for orchestration:

- `create_execution_plan` - Create high-level execution plans from objectives
- `get_execution_plan` - Retrieve plans with progress tracking via linked Tasks

- `execute_with_plan` - Execute plans by creating coordinated Tasks for agents
- `list_execution_plans` - List and monitor execution plans

Plans create Tasks for implementation - Plans are high-level orchestration templates while Tasks are specific work items assigned to agents.

📊 Data Scope

- **Documentation & Websites**: Shared project-wide across all repositories
- **Agents, Tasks, Memory**: Scoped per repository_path for isolation
- **Prompts & Resources**: Available globally for all projects

🎯 Multi-Agent Orchestration

Architect-Led Coordination

ZMCPTools features an AI architect that automatically analyzes objectives and spawns coordinated agent teams with proper dependencies and real-time communication.

🎯 Usage

CLI Commands

```
# Show help and available commands
zmcp-tools --help

# Show system status
zmcp-tools status

# Start the MCP server
zmcp-server

# Agent management
zmcp-tools agent list
zmcp-tools agent spawn -t <type> -r <repository> -d <description>
zmcp-tools agent terminate -i <agent-id>

# Task management
zmcp-tools task list
zmcp-tools task create -t <title> -d <description>

# Memory operations
zmcp-tools memory search -q <query>
zmcp-tools memory store -t <title> -c <content>

# Communication rooms
zmcp-tools room list
zmcp-tools room join -n <name>
```

🛠️ Development Commands

```
# Initial setup (one time - using pnpm)
pnpm install                  # Install dependencies first
pnpm run install:global       # Build, link globally, and configure everything

# Alternative package managers
npm install && npm run build && npm link && zmcp-tools install
yarn install && yarn build && yarn link && zmcp-tools install
bun install && bun run build && bun link && zmcp-tools install

# Development with hot-reload
pnpm dev          # Start MCP server with tsx
pnpm dev:cli      # Start CLI with tsx

# Building and testing
pnpm build        # Compile TypeScript to dist/
pnpm test         # Run Vitest tests
pnpm test:ui      # Run tests with UI
pnpm test:run     # Run tests once

# Code quality
pnpm lint         # ESLint checking
pnpm typecheck    # TypeScript type checking

# Production
pnpm start        # Start compiled MCP server
pnpm start:cli    # Start compiled CLI

# Management
zmcp-tools install    # Install/reinstall MCP server
zmcp-tools uninstall  # Remove MCP server and settings
zmcp-tools status     # Check system status
zmcp-tools help       # Show all commands

# For users who installed via npx
npx zmcp-tools@latest status     # Check status
npx zmcp-tools@latest uninstall  # Remove installation
```

🌟 TypeScript Features:

- **🎯 Type Safety**
  - Full TypeScript implementation with strict mode
  - Zod schemas for runtime validation
  - Compile-time error checking
  - IntelliSense support in IDEs
- **🚀 Performance**
  - Better-sqlite3 for high-performance database operations
  - ES2022 target with modern optimizations
  - Efficient memory management
  - Fast development with tsx hot-reload
- **🧪 Testing**

- Vitest for modern testing experience
- UI mode for interactive test debugging
- Coverage reports with V8 provider
- TypeScript test support out of the box

- 📦 **Module System**
  - ESNext modules for tree-shaking
  - Clean imports and exports
  - Library mode for programmatic use
  - Dual CLI and server binaries

Convenient Aliases

The following aliases are available (add to `~/.zshrc`):

```
alias mcp-tools="zmcp-tools"
alias mcp-server="zmcp-server"
alias mcp-status="zmcp-tools status"
alias mcp-dev="npm run dev"
```

⚙️ Configuration

TypeScript Configuration

```
# Build configuration in tsconfig.json
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ESNext",
    "moduleResolution": "bundler",
    "strict": true,
    "experimentalDecorators": true,
    "outDir": "dist",
    "rootDir": "src"
  }
}

# Development scripts (works with npm/yarn/pnpm/bun)
npm run dev       # Hot-reload development
npm run build     # Production build
npm test          # Run test suite
```

🎯 **TypeScript Features:**

- **Strict Type Checking**: Full type safety with strict mode enabled
- **Modern ES Modules**: ESNext target with bundler resolution
- **Development Tools**: tsx for hot-reload, Vitest for testing
- **Code Quality**: ESLint with TypeScript rules

- **Binary Generation**: Dual binaries for CLI and server
- **Library Mode**: Exportable as TypeScript library

Manual MCP Server Configuration (if needed)

The installer automatically configures the MCP server, but if you need to manually configure it:

```
# For production install (via npx)
# Server is installed at ~/.mcptools/server/index.js
# Configuration is automatic via 'claude mcp add --scope local'

# For development install
claude mcp add zmcp-tools $(pwd)/dist/server/index.js

# Verify installation
claude mcp list

# Test server directly
echo '{"jsonrpc":"2.0","id":1,"method":"tools/list"}' | node ~/.mcptools/server/index.js
```

🛡️ MCP Protocol Compliance

Full MCP 1.15.0 Compatibility

- **JSON-RPC 2.0**: Complete implementation with proper message handling
- **Stdio Transport**: High-performance local process communication
- **Tool Definitions**: 40 tools with comprehensive input schemas and validation
- **Error Handling**: Standardized MCP error codes and proper error propagation
- **Initialization Protocol**: Full handshake with capability negotiation

TypeScript MCP Implementation

Full TypeScript implementation with MCP SDK, proper error handling, and tool management.

MCP Best Practices Implemented

- **Input Validation**: Zod schemas for runtime type safety
- **Proper Error Handling**: MCP-compliant error responses with detailed messages
- **Tool Annotations**: Descriptive schemas with security hints where applicable
- **Resource Management**: SQLite database connections with proper cleanup
- **Transport Security**: Stdio transport for secure local communication

🏗️ Architecture

Modern CLI with TypeScript

- **Commander.js Framework**: Type-safe commands with automatic help generation
- **Console Formatting**: Colored output and structured command display

- **CLI Interface**: Comprehensive command structure for all operations
- **Status Reporting**: Real-time feedback for operations and system health

Dual-Layer Design

Layer 1: Enhanced File Operations

- Hierarchical ignore pattern system (.claudeignore > .gitignore > defaults)
- Fuzzy string matching with configurable similarity thresholds
- Cross-platform screenshot capabilities with native tool integration

Layer 2: Multi-Agent Orchestration

- Project analysis integration for intelligent code understanding
- Documentation intelligence with local vector database storage
- SQLite-based coordination with real-time communication
- Foundation Session pattern for 85-90% token cost reduction

Enhanced Project Integration

Automatic CLAUDE.md Integration:

- Unique delimiters: `<!-- zzZMCPToolszz START/END -->`
- Intelligent replacement of existing sections
- Architect-led workflow guidance
- Multi-agent examples with dependencies
- Documentation-driven development patterns

Automatic Claude Hooks Integration:

- Session start context injection for instant MCP tools awareness
- Knowledge graph and core tools reminders (analyze_project_structure(), search_knowledge_graph(), plan tools)
- One-time per session to avoid context bloat
- Non-destructive settings.json merging

Per-Project Setup:

```
# Automatic integration during project setup
zmcp-tools install --project

# Creates/updates:
# • MCP server registration via 'claude mcp add --scope local'
# • ./.claude/commands/ (Claude commands)
# • ./.claude/hooks/ (session start context injection)
# • ./.claude/settings.json (hook configuration)
# • ./CLAUDE.md (integration guide with architect examples)
```

Data Storage

- **Installation**: Local project directory (`./dist/`)
- **Data Directory**: `~/.mcptools/data/` (SQLite databases)
- **Vector Storage**: `~/.mcptools/lancedb/` (LanceDB vector database)
- **Main Database**: `~/.mcptools/data/claude_mcp_tools.db`
- **All Data**: Agents, tasks, memory, and documentation in SQLite database
- **Vector Data**: Embeddings and vector indices stored in LanceDB
- **Cache**: Foundation session cache in memory/disk with vector index caching

🛠️ Development

```
# Clone and setup development environment
git clone https://github.com/zachhandley/ZMCPTools
cd ZMCPTools
pnpm install

# Quick setup
pnpm run install:global && mcp-tools install # Build, link

# Development mode
pnpm dev          # Run MCP server with hot-reload
pnpm dev:cli      # Run CLI with hot-reload

# Build and test
pnpm build        # Compile TypeScript
pnpm test         # Run test suite

# Test the binaries
node dist/index.js       # MCP server
node dist/cli/index.js   # CLI interface
```

🎨 CLI Examples

Status Display

```
$ zmcp-tools status
```

```
ZMCPTools Status:
✅ TypeScript Build: dist/ directory exists
✅ Data Directory: ~/.mcptools/data/
✅ SQLite Database: claude_mcp_tools.db
✅ LanceDB Vector Database: ~/.mcptools/lancedb/
✅ MCP Server: zmcp-server binary available
✅ Dependencies: @modelcontextprotocol/sdk, @lancedb/lancedb, better-sqlite3
```

Development Workflow

```
$ npm run dev
```

```
Starting TypeScript development server...
✅ TypeScript compilation successful
✅ MCP server starting on stdio
✅ SQLite databases initialized
✅ LanceDB vector database initialized
✅ Agent orchestration ready
✅ Foundation cache system active

Listening for MCP requests...
Press Ctrl+C to stop
```

🔍 Troubleshooting

Installation Issues

```
# Check prerequisites
node --version          # Node.js 18+ required
pnpm --version          # Package manager (or npm/yarn/bun)
claude --version        # Claude CLI required

# Clean installation
rm -rf node_modules dist
pnpm install
pnpm build

# Development installation
git clone https://github.com/zachhandley/ZMCPTools
cd ZMCPTools
pnpm install && pnpm run install:global
```

Verification

```
# Check build output
ls -la dist/
node dist/index.js --help

# Check data directory
ls -la ~/.mcptools/data/

# Test MCP server
claude mcp list
echo '{"jsonrpc":"2.0","id":1,"method":"tools/list"}' | node dist/index.js
```

Server Connection Issues

```
# Test TypeScript compilation
pnpm typecheck
pnpm lint
```

```
# Test MCP server directly
node dist/index.js

# Debug with development server
pnpm dev

# Check MCP configuration
claude mcp list
claude mcp remove zmcp-tools
claude mcp add zmcp-tools $(pwd)/dist/index.js
```

TypeScript Issues

```
# Type checking errors
pnpm typecheck                      # Check TypeScript errors
npx tsc --noEmit --pretty           # Detailed type errors

# Runtime errors
node --inspect dist/index.js        # Debug with Node.js inspector
pnpm dev                            # Hot-reload development

# Database issues
rm -rf ~/.mcptools/data/*.db        # Reset databases
node dist/index.js                   # Reinitialize

# Dependency issues
rm -rf node_modules pnpm-lock.yaml
pnpm install                        # Clean dependency install
```

📈 Performance & Architecture

Production Metrics

- **61 MCP Tools**: Complete tool suite with full type safety and MCP 1.15.0 compliance
- **Database Performance**: SQLite with WAL mode and optimized connection pooling
- **Vector Search**: LanceDB native TypeScript bindings for <100ms semantic search
- **Memory Efficiency**: <75MB baseline with intelligent caching and cleanup
- **Response Time**: <200ms average tool execution, <50ms for cached operations
- **Cost Optimization**: 85-90% reduction through automatic foundation session management

Technical Architecture

TypeScript-First Design:

- Strict TypeScript with Zod schemas for runtime validation
- Modern ES modules with tree-shaking optimization
- Dual binary system (CLI + MCP server)
- Hot-reload development with tsx

Database Layer:

- SQLite with Write-Ahead Logging for performance
- Drizzle ORM for type-safe database operations
- Automatic schema migrations and connection pooling
- LanceDB vector database for semantic search

MCP Compliance:

- Full MCP 1.15.0 protocol implementation
- JSON-RPC 2.0 with proper error handling
- Stdio and HTTP transport support
- Resource and prompt management

🤝 Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/amazing-feature`)
3. Make your changes with tests
4. Test with Claude Code integration
5. Submit a pull request

Development Guidelines

- Follow TypeScript strict mode requirements
- Add comprehensive error handling with MCP compliance
- Include tool annotations for destructive/read-only operations
- Test all changes with the actual MCP server integration

📜 License

MIT License - see LICENSE file for details.

🚀 Supercharge your Claude Code workflows with ZMCPTools - TypeScript-powered multi-agent

Releases

No releases published

Packages

No packages published

Contributors 2

ZachHandley ZachHandley

lwsinclair Lawrence Sinclair

Languages

TypeScript 99.2% **Other** 0.8%