


# Zen MCP: Many Workflows. One Context.

[zen\\_web.webm](https://zen_web.webm)

 **Claude** OR **Gemini CLI** + [Gemini / OpenAI / Grok / OpenRouter / DIAL / Ollama / Any Model] = Your Ultimate AI Development Team

The ultimate development partners for your favorite Coding Agent (**Claude** OR **Gemini CLI**) - a Model Context Protocol server that gives you access to multiple AI models for enhanced code analysis, problem-solving, and collaborative development.

**Features true AI orchestration with conversations that continue across workflows** - Give Claude a complex *workflow* and let it orchestrate between models automatically. Claude stays in control, performs the actual work, but gets perspectives from the best AI for each subtask. With tools like [planner](#) for breaking down complex projects, [analyze](#) for understanding codebases, [codereview](#) for audits, [refactor](#) for improving code structure, [debug](#) for solving complex problems, and [precommit](#) for validating changes, Claude can switch between different tools *and* models mid-conversation, with context carrying forward seamlessly.

**Example Workflow - Claude Code:** 1. Perform a codereview using gemini pro and o3 and use planner to generate a detailed plan, implement the fixes and do a final precommit check by continuing from the previous codereview 2. This triggers a [codereview](#) workflow where Claude walks the code, looking for all kinds of issues 3. After multiple passes, collects relevant code and makes note of issues along the way 4. Maintains a confidence level between exploring, low, medium, high and certain to track how confidently it's been able to find and identify issues 5. Generates a detailed list of critical -> low issues 6. Shares the relevant files, findings, etc with **Gemini Pro** to perform a deep dive for a second [codereview](#) 7. Comes back with a response and next does the same with o3, adding to the prompt if a new discovery comes to light 8. When done, Claude takes in all the feedback and combines a single list of all critical -> low issues, including good patterns in your code. The final list includes new findings or revisions in case Claude misunderstood or missed something crucial and one of the other models pointed this out 9. It then uses the [planner](#) workflow to break the work down into simpler steps if a major refactor is required 10. Claude then performs the actual work of fixing highlighted issues 11. When done, Claude returns to Gemini Pro for a [precommit](#) review

All within a single conversation thread! Gemini Pro in step 11 *knows* what was recommended by O3 in step 7! Taking that context and review into consideration to aid with its final pre-commit review.

**Think of it as Claude Code *for* Claude Code.** This MCP isn't magic. It's just **super-glue**.

**Remember:** Claude stays in full control — but **YOU** call the shots. Zen is designed to have Claude engage other models only when needed — and to follow through with meaningful back-and-forth. **You're** the one who crafts the powerful prompt that makes Claude bring in Gemini, Flash, O3 — or fly solo. You're the guide. The prompter. The puppeteer. #### You are the AI - **Actually Intelligent**.

Because these AI models [clearly aren't when they get chatty](#) →

# Quick Navigation

- **Getting Started**
  - [Quickstart](#) - Get running in 5 minutes
  - [Supported Models](#) - All available AI models
  - [Available Tools](#) - Overview of all tools
  - [AI-to-AI Conversations](#) - Multi-turn conversations
- **Tools Reference**
  - [chat](#) - Collaborative thinking
  - [thinkdeep](#) - Extended reasoning
  - [challenge](#) - Prevents **You're absolutely right!** responses
  - [planner](#) - Interactive step-by-step planning
  - [consensus](#) - Multi-model consensus analysis
  - [codereview](#) - Code review
  - [precommit](#) - Pre-commit validation
  - [debug](#) - Debugging help
  - [analyze](#) - File analysis
  - [refactor](#) - Code refactoring with decomposition focus
  - [tracer](#) - Call-flow mapping and dependency tracing
  - [testgen](#) - Test generation with edge cases
  - [secaudit](#) - Security audit with OWASP analysis
  - [docgen](#) - Documentation generation with complexity analysis
- **Advanced Usage**
  - [Advanced Features](#) - AI-to-AI conversations, large prompts, web search
  - [Complete Advanced Guide](#) - Model configuration, thinking modes, workflows, tool parameters
- **Setup & Support**
  - [WSL Setup Guide](#) - Windows Subsystem for Linux configuration
  - [Troubleshooting Guide](#) - Common issues and debugging steps
  - [License](#) - Apache 2.0

## Supported Models

Zen MCP Server supports a wide range of AI models from multiple providers. Claude automatically selects the best model for each task in `auto` mode, or you can specify models directly.

### Native API Providers

**Google Gemini:** - `gemini-2.5-pro` (aliases: `pro`, `gemini-pro`, `gemini`) - 1M context, deep analysis & extended thinking - `gemini-2.5-flash` (aliases: `flash`, `gemini-flash`) - 1M context, fast responses

**OpenAI:** - `gpt-5` (alias: `gpt5`) - 400K context, 128K output, advanced reasoning with `reasoning_effort` parameter - `gpt-5-mini` (aliases: `gpt5-mini`, `gpt5mini`) - 400K context, 128K output, balanced performance/cost - `gpt-5-nano` (aliases: `gpt5-nano`, `gpt5nano`) - 400K context, 128K output, fastest/cheapest variant - `o3` - 200K context, strong reasoning (no temperature support) - `o3-mini` (aliases: `o3mini`) - 200K context, balanced performance - `o3-pro` (alias: `o3-pro`) - 200K context, professional-grade (EXPENSIVE - use sparingly) - `o4-mini` (aliases: `mini`, `o4mini`) - 200K context, optimized for shorter contexts - `gpt-4.1-2025-04-14` (alias: `gpt4.1`) - 1M context, advanced reasoning

**X.AI:** - grok-4 (aliases: grok, grok4) - 256K context, frontier multimodal reasoning with vision support - grok-4-heavy (aliases: grok4-heavy, grok-heavy) - 256K context, multi-agent reasoning - grok-3 (alias: grok3) - 131K context, advanced reasoning model - grok-3-fast (aliases: grok3-fast, grokfast) - 131K context, higher performance variant

## OpenRouter & Custom Providers

**Via OpenRouter:** Access to Claude Opus/Sonnet/Haiku, additional GPT models, Mistral, Llama, and more. Configure in [conf/custom\\_models.json](#).

**Local Models:** Support for Ollama, vLLM, LM Studio, or any OpenAI-compatible API endpoint. Common local models: - llama3.2 (aliases: local-llama, local) - Via Ollama/vLLM - Custom models configured in [conf/custom\\_models.json](#)

## Model Capabilities

**Vision Support:** GPT-5 family, O3/O4 models, Grok-4 family, Gemini models **Extended**

**Thinking:** Gemini models (thinking modes), GPT-5 family (reasoning\_effort) **No Temperature:**

O3/O4 models use fixed temperature for consistent reasoning

## Why This Server?

Claude is brilliant, but sometimes you need: - **Guided workflows** - Developer-centric processes that enforce systematic investigation, preventing rushed analysis by ensuring Claude examines code thoroughly at each phase ([debug](#), [precommit](#), [refactor](#), [analyze](#), [codereview](#)) - **Multiple AI perspectives** - Let Claude orchestrate between different models to get the best analysis - **Automatic model selection** - Claude picks the right model for each task (or you can specify) - **A senior developer partner** to validate and extend ideas ([chat](#)) - **A second opinion** on complex architectural decisions - augment Claude's thinking with perspectives from Gemini Pro, O3, or [dozens of other models via custom endpoints](#) ([thinkdeep](#)) - **Get multiple expert opinions** - Have different AI models debate your ideas (some supporting, some critical) to help you make better decisions ([consensus](#)) - **Professional code reviews** with actionable feedback across entire repositories ([codereview](#)) - **Pre-commit validation** with deep analysis using the best model for the job ([precommit](#)) - **Expert debugging** - O3 for logical issues, Gemini for architectural problems ([debug](#)) - **Extended context windows beyond Claude's limits** - Delegate analysis to Gemini (1M tokens) or O3 (200K tokens) for entire codebases, large datasets, or comprehensive documentation - **Model-specific strengths** - Extended thinking with Gemini Pro, fast iteration with Flash, strong reasoning with O3, advanced capabilities with GPT-5 (reasoning\_effort), local privacy with Ollama - **Local model support** - Run models like Llama 3.2 locally via Ollama, vLLM, or LM Studio for privacy and cost control - **Dynamic collaboration** - Models can request additional context and follow-up replies from Claude mid-analysis - **Smart file handling** - Automatically expands directories, manages token limits based on model capacity - **Vision support** - Analyze images, diagrams, screenshots, and visual content with vision-capable models - [Bypass MCP's token limits](#) - Work around MCP's 25K limit automatically - [Context revival across sessions](#) - Continue conversations even after Claude's context resets, with other models maintaining full history

## Pro Tip: Context Revival

**This is an extremely powerful feature that cannot be highlighted enough:**

The most amazing side-effect of this *conversation continuation* system is that even AFTER Claude's context resets or compacts, since the continuation info is kept within MCP's memory, you can ask it to *continue* discussing the plan with o3, and it will suddenly revive Claude because O3 would know what was being talked about and relay this back in a way that re-ignites Claude's understanding. All this without wasting context on asking Claude to ingest lengthy documents / code again and re-prompting it to communicate with another model. Zen manages that internally. The model's response revives Claude with better context around the discussion than an automatic summary ever can.

 [Read the complete technical deep-dive on how this revolutionary system works](#)

This server orchestrates multiple AI models as your development team, with Claude automatically selecting the best model for each task or allowing you to choose specific models for different strengths.

*Let me now brainstorm with Gemini about these ob*

- **gemini:think\_deeper** (MCP)(current\_analysis: "I've  
The main performance bo  
processing\*\*: Uses Stri  
`json.index(after: curr  
but still has string c  
`expectedTypes.contains  
`String(repeating: opti  
operations\*\*: String ex  
state machine\*\*: Large  
character by character,  
handles malformed JSON  
areas:\n- Main sanitiza  
index advancement\n- Ex  
optimization strategies  
"Swift JSON sanitizer  
maintaining correctness  
["performance","memory"

└ Extended Analysis by Gemini:

Excellent analysis, Claude. You've correctly i  
on the character-by-character processing, stri

... +159 lines (ctrl+r to expand)

\* Thinking...

Wow, Gemini provided incredible insights! This i  
initial analysis. The key insight is that we're  
just need to process ASCII control characters fo

## Prompt Used:

Study the code properly, think deeply about what this does and then see if there's any room for improvement in terms of performance optimizations, brainstorm with gemini on this to get feedback and then confirm any change by first adding a unit test with `measure` and measuring current code and then implementing the optimization and measuring again to ensure it improved, then share results. Check with gemini in between as you make tweaks.

The final implementation resulted in a 26% improvement in JSON parsing performance for the selected library, reducing processing time through targeted, collaborative optimizations guided by Gemini's analysis and Claude's refinement.

## Quickstart (5 minutes)

### Prerequisites

- Python 3.10+ (3.12 recommended)
- Git
- **Windows users:** WSL2 is required for Claude Code CLI

### 1. Get API Keys (at least one required)

**Option A: OpenRouter (Access multiple models with one API) - OpenRouter:** Visit [OpenRouter](#) for access to multiple models through one API. [Setup Guide](#) - Control model access and spending limits directly in your OpenRouter dashboard - Configure model aliases in [conf/custom\\_models.json](#)

**Option B: Native APIs - Gemini:** Visit [Google AI Studio](#) and generate an API key. For best results with Gemini 2.5 Pro, use a paid API key as the free tier has limited access to the latest models. - **OpenAI:** Visit [OpenAI Platform](#) to get an API key for O3 and GPT-5 model access. - **X.AI:** Visit [X.AI Console](#) to get an API key for GROK model access. - **DIAL:** Visit [DIAL Platform](#) to get an API key for accessing multiple models through their unified API. DIAL is an open-source AI orchestration platform that provides vendor-agnostic access to models from major providers, open-source community, and self-hosted deployments. [API Documentation](#)

**Option C: Custom API Endpoints (Local models like Ollama, vLLM) [Please see the setup guide](#).** With a custom API you can use: - **Ollama:** Run models like Llama 3.2 locally for free inference - **vLLM:** Self-hosted inference server for high-throughput inference - **LM Studio:** Local model hosting with OpenAI-compatible API interface - **Text Generation WebUI:** Popular local interface for running models - **Any OpenAI-compatible API:** Custom endpoints for your own infrastructure

**Note:** Using multiple provider options may create ambiguity about which provider / model to use if there is an overlap. If all APIs are configured, native APIs will take priority when there is a clash in model name, such as for `gemini` and `o3`. Configure your model aliases and give them unique names in [conf/custom\\_models.json](#)

## 2. Choose Your Installation Method

### Option A: Quick Install with uvx

**Prerequisites:** Install [uv](#) first (required for uvx)

Claude Desktop Configuration

Add this to your `claude_desktop_config.json`:

```
{
  "mcpServers": {
    "zen": {
      "command": "sh",
      "args": [
        "-c",
        "exec $(which uvx || echo uvx) --from git+https://github.com/BeehiveInnovations/zen-mcp-server.git zen-mcp-server"
      ],
      "env": {
        "PATH": "/usr/local/bin:/usr/bin:/bin:/opt/homebrew/bin:~/.local/bin",
        "OPENAI_API_KEY": "your_api_key_here"
      }
    }
  }
}
```

Claude Code CLI Configuration

Create a `.mcp.json` file in your project root for [project-scoped configuration](#):

```
{
  "mcpServers": {
    "zen": {
      "command": "sh",
      "args": [
        "-c",
        "exec $(which uvx || echo uvx) --from git+https://github.com/BeehiveInnovations/zen-mcp-server.git zen-mcp-server"
      ],
      "env": {
        "PATH": "/usr/local/bin:/usr/bin:/bin:/opt/homebrew/bin:~/.local/bin",
        "OPENAI_API_KEY": "your_api_key_here"
      }
    }
  }
}
```



```
}  
}
```

## Gemini CLI Configuration

Edit ~/.gemini/settings.json and add:

```
{  
  "mcpServers": {  
    "zen": {  
      "command": "sh",  
      "args": [  
        "-c",  
        "exec $(which uvx || echo uvx) --from git+https://  
        github.com/BeehiveInnovations/zen-mcp-server.git zen-mcp-  
        server"  
      ],  
      "env": {  
        "PATH": "/usr/local/bin:/usr/bin:/bin:/opt/homebrew/  
        bin:~/.local/bin",  
        "OPENAI_API_KEY": "your_api_key_here"  
      }  
    }  
  }  
}
```

**Note:** While Zen MCP Server connects successfully to Gemini CLI, tool invocation is not working correctly yet. See [Gemini CLI Setup](#) for updates.

**What this does:** - **Zero setup required** - uvx handles everything automatically - **Always up-to-date** - Pulls latest version on each run - **No local dependencies** - Works without Python environment setup - **Instant availability** - Ready to use immediately

## Option B: Traditional Clone and Set Up

```
# Clone to your preferred location  
git clone https://github.com/BeehiveInnovations/zen-mcp-  
server.git  
cd zen-mcp-server  
  
# One-command setup installs Zen in Claude  
./run-server.sh  
  
# Or for Windows users using PowerShell:  
./run-server.ps1  
  
# To view MCP configuration for Claude  
./run-server.sh -c  
  
# PowerShell:  
./run-server.ps1 -Config
```



```
# See help for more
./run-server.sh --help
```

```
# PowerShell:
./run-server.ps1 -Help
```

**What this does:** - Sets up everything automatically - Python environment, dependencies, configuration - **Configures Claude integrations** - Adds to Claude Code CLI and guides Desktop setup - **Ready to use immediately** - No manual configuration needed - **Also works with Gemini CLI** - See [Gemini CLI Setup](#) for configuration

**After updates:** Always run `./run-server.sh` again after `git pull` to ensure everything stays current.

**Windows users:** Using WSL? See the [WSL Setup Guide](#) for detailed instructions.

### 3. Add Your API Keys

```
# Edit .env to add your API keys (if not already set in
environment)
nano .env

# The file will contain, at least one should be set:
# GEMINI_API_KEY=your-gemini-api-key-here # For Gemini models
# OPENAI_API_KEY=your-openai-api-key-here # For O3 and GPT-5
models
# OPENROUTER_API_KEY=your-openrouter-key # For OpenRouter (see
docs/custom_models.md)
# DIAL_API_KEY=your-dial-api-key-here # For DIAL platform

# For DIAL (optional configuration):
# DIAL_API_HOST=https://core.dialx.ai # Default DIAL host
(optional)
# DIAL_API_VERSION=2024-12-01-preview # API version
(optional)
# DIAL_ALLOWED_MODELS=o3,gemini-2.5-pro # Restrict to specific
models (optional)

# For local models (Ollama, vLLM, etc.):
# CUSTOM_API_URL=http://localhost:11434/v1 # Ollama example
# CUSTOM_API_KEY= # Empty
for Ollama
# CUSTOM_MODEL_NAME=llama3.2 # Default
model

# Note: At least one API key OR custom URL is required
```

**No restart needed:** The server reads the `.env` file each time Claude calls a tool, so changes take effect immediately.

**Next:** Now run `claude` from your project folder using the terminal for it to connect to the newly added mcp server. If you were already running a `claude` code session, please exit and start a new session.

## If Setting up for Claude Desktop

**Need the exact configuration?** Run `./run-server.sh -c` to display the platform-specific setup instructions with correct paths.

1. **Open Claude Desktop config:** Settings → Developer → Edit Config
2. **Copy the configuration** shown by `./run-server.sh -c` into your `claude_desktop_config.json`
3. **Restart Claude Desktop** for changes to take effect

## 4. Start Using It!

Just ask Claude naturally: - “Think deeper about this architecture design with zen” → Claude picks best model + `thinkdeep` - “Using zen perform a code review of this code for security issues” → Claude might pick Gemini Pro + `codereview` - “Use zen and debug why this test is failing, the bug might be in `my_class.swift`” → Claude might pick O3 + `debug` - “With zen, analyze these files to understand the data flow” → Claude picks appropriate model + `analyze` - “Use flash to suggest how to format this code based on the specs mentioned in `policy.md`” → Uses Gemini Flash specifically - “Think deeply about this and get o3 to debug this logic error I found in the `checkOrders()` function” → Uses O3 specifically - “Use gpt-5 with high reasoning effort to analyze this algorithm’s time complexity” → Uses GPT-5 with `reasoning_effort` parameter - “Brainstorm scaling strategies with pro. Study the code, pick your preferred strategy and debate with pro to settle on two best approaches” → Uses Gemini Pro specifically - “Use local-llama to localize and add missing translations to this project” → Uses local Llama 3.2 via custom URL - “First use local-llama for a quick local analysis, then use opus for a thorough security review” → Uses both providers in sequence

## Available Tools

These aren’t just tools—they’re how you get Claude to think like a real developer. Instead of rushing to reply with surface-level takes or shallow-insight, these workflows make Claude pause, dig into your code, and reason through problems step by step.

It’s the difference between a rushed guess and a focused second pair of eyes that actually understands your code. Try them and feel the difference.

**Quick Tool Selection Guide:** - **Need a thinking partner?** → `chat` (brainstorm ideas, get second opinions, validate approaches) - **Need deeper thinking?** → `thinkdeep` (extends analysis, finds edge cases) - **Want to prevent “You’re absolutely right!” responses?** → `challenge` (challenges assumptions, encourages thoughtful re-evaluation) - **Need to break down complex projects?** → `planner` (step-by-step planning, project structure, breaking down complex ideas) - **Need multiple perspectives?** → `consensus` (get diverse expert opinions on proposals and decisions) - **Code needs review?** → `codereview` (bugs, security, performance issues) - **Pre-commit validation?** → `precommit` (validate git changes before committing) - **Something’s broken?** → `debug` (systematic investigation, step-by-step root cause analysis) - **Want to understand code?** → `analyze` (architecture, patterns, dependencies) - **Code needs refactoring?** → `refactor` (intelligent refactoring with decomposition focus) - **Need call-flow analysis?** → `tracer` (generates prompts for execution tracing and dependency mapping) - **Need comprehensive tests?** → `testgen` (generates test suites with edge cases) - **Security**

**concerns?** → `secaudit` (OWASP analysis, compliance evaluation, vulnerability assessment) - **Code needs documentation?** → `docgen` (generates comprehensive documentation with complexity analysis) - **Which models are available?** → `listmodels` (shows all configured providers and models) - **Server info?** → `version` (version and configuration details)

**Auto Mode:** When `DEFAULT_MODEL=auto`, Claude automatically picks the best model for each task. You can override with: “Use flash for quick analysis” or “Use o3 to debug this”.

**Model Selection Examples:** - Complex reasoning/debugging → Claude picks **GPT-5** (flagship reasoning) - Multi-agent analysis → Claude picks **Grok-4 Heavy** (collaborative reasoning) - Critical thinking tasks → Claude picks **O3** (deep logical analysis) - Advanced code analysis → Claude picks **Grok-4** (thinking capability) - Quick responses/chat → Claude picks **GPT-5 Nano** (ultra-fast, low-latency) - Balanced tasks → Claude picks **GPT-5 Mini** (optimal cost/performance) - Legacy fallbacks → Claude picks **Flash** or **O4-mini** as needed

**Model Prioritization Strategy:** Claude uses intelligent tool categorization to select optimal models:

- 🧠 **Extended Reasoning** (analysis, debugging, security audits): GPT-5 → Grok-4 Heavy → O3 → Grok-4 → Grok-3
- ⚡ **Fast Response** (chat, quick tasks): GPT-5 Nano → O4-mini → GPT-5 Mini → O3-mini → Grok-3-fast
- ⚖️ **Balanced** (general tasks): GPT-5 Mini → O4-mini → O3-mini → Grok-4 → Grok-3

This prioritization is based on OpenAI’s guidance for GPT-5 model selection and Grok-4’s multi-agent capabilities, optimizing for reasoning depth, speed, and cost-efficiency.

**Pro Tip:** Thinking modes (for Gemini models) control depth vs token cost. Use “minimal” or “low” for quick tasks, “high” or “max” for complex problems. [Learn more](#)

**Tools Overview:** 1. [chat](#) - Collaborative thinking and development conversations 2. [thinkdeep](#) - Extended reasoning and problem-solving 3. [challenge](#) - Critical challenge prompt, prevents **You’re absolutely right!** 4. [planner](#) - Interactive sequential planning for complex projects 5. [consensus](#) - Multi-model consensus analysis with stance steering 6. [codereview](#) - Professional code review with severity levels 7. [precommit](#) - Validate git changes before committing 8. [debug](#) - Systematic investigation and debugging 9. [analyze](#) - General-purpose file and code analysis 10. [refactor](#) - Code refactoring with decomposition focus 11. [tracer](#) - Static code analysis prompt generator for call-flow mapping 12. [testgen](#) - Comprehensive test generation with edge case coverage 13. [secaudit](#) - Comprehensive security audit with OWASP Top 10 analysis 14. [docgen](#) - Comprehensive documentation generation with complexity analysis 15. [listmodels](#) - Display all available AI models organized by provider 16. [version](#) - Get server version and configuration

## 1. chat - General Development Chat & Collaborative Thinking

Your thinking partner for brainstorming, getting second opinions, and validating approaches. Perfect for technology comparisons, architecture discussions, and collaborative problem-solving.

Chat with zen about the best approach for user authentication in my React app

 [Read More](#) - Detailed features, examples, and best practices

## 2. thinkdeep - Extended Reasoning Partner

Get a second opinion to augment Claude's own extended thinking. Uses specialized thinking models to challenge assumptions, identify edge cases, and provide alternative perspectives.

The button won't animate when clicked, it seems something else is intercepting the clicks. Use thinkdeep with gemini pro after gathering related code and handing it the files and find out what the root cause is

 [Read More](#) - Enhanced analysis capabilities and critical evaluation process

## 3. challenge - Critical Challenge Prompt

Encourages thoughtful reassessment of statements instead of automatic agreement, especially when you're wrong. Wraps your input with instructions for critical thinking and honest analysis.

challenge isn't adding this function to the base class a bad idea?

Normally, your favorite coding agent will enthusiastically reply with **“You’re absolutely right!”**—then proceed to completely reverse the *correct* strategy, without ever explaining why you’re wrong.

Example: Without vs With Zen

**Without Zen:**



※ Tip: Press Esc twice to edit your previ

> Give me two good reasons why I should ma

- I'll give you two compelling reasons why

### **1. Higher Revenue Per User**

Apple users spend 2-3x more on apps and despite having fewer downloads. Apple's

### **2. Lower Development & Support Costs**

Developing for iOS only means dealing wi  
You can ship faster, iterate quicker, an

> But nowadays Apple has started charging

- You make a valid point about Apple's dev

### **1. Near-Zero Entry Cost**

Google Play only charges a one-time \$25 with minimal upfront investment, keeping

### **2. Massive Global Market Access**

Android commands 70%+ global market shar

**With Zen:**



> Give me two good reasons why I should ma

- Here are two strong financial reasons to

1. **Higher Revenue Per User:** Apple users (ARPU) on iOS is typically 2-4x higher t

2. **Faster Development ROI:** Building for can reach market faster, validate your

> But nowadays Apple has started charging


- **zen:challenge (MCP)**(prompt: "But nowaday  
└ {

```
    "status": "challenge_created",  
    "original_statement": "But nowadays  
... +8 lines (ctrl+r to expand)
```

- I need to challenge this claim. Apple's to \$500.

Given the correct information, here are

1. **Market Scale:** Android has ~70% global markets where growth potential is highes

 [Read More](#) - Challenge an approach or validate ideas with confidence

## 4. planner - Interactive Step-by-Step Planning

Break down complex projects or ideas into manageable, structured plans through step-by-step thinking. Perfect for adding new features to an existing system, scaling up system design, migration strategies, and architectural planning with branching and revision capabilities.

### Pro Tip

Claude supports sub-tasks where it will spawn and run separate background tasks. You can ask Claude to run Zen's planner with two separate ideas. Then when it's done, use Zen's consensus tool to pass the entire plan and get expert perspective from two powerful AI models on which one to work on first! Like performing **AB** testing in one-go without the wait!

```
Create two separate sub-tasks: in one, using planner tool show me
how to add natural language support
to my cooking app. In the other sub-task, use planner to plan how
to add support for voice notes to my cooking app.
Once done, start a consensus by sharing both plans to o3 and
flash to give me the final verdict. Which one do
I implement first?
```

 [Read More](#) - Step-by-step planning methodology and multi-session continuation

## 5. consensus - Multi-Model Perspective Gathering

Get diverse expert opinions from multiple AI models on technical proposals and decisions. Supports stance steering (for/against/neutral) and structured decision-making.

```
Get a consensus with flash taking a supportive stance and gemini
pro being critical to evaluate whether we should
migrate from REST to GraphQL for our API. I need a definitive
answer.
```

 [Read More](#) - Multi-model orchestration and decision analysis


## 6. codereview - Professional Code Review

Comprehensive code analysis with prioritized feedback and severity levels. This workflow tool guides Claude through systematic investigation steps with forced pauses between each step to ensure thorough code examination, issue identification, and quality assessment before providing expert analysis.

```
Perform a codereview with gemini pro especially the auth.py as I
feel some of the code is bypassing security checks
and there may be more potential vulnerabilities. Find and share
related code."
```

**Tips:** \* To avoid extra API costs, add do not use another model to run the entire codereview workflow locally. \* If you want to **always** consult an external model (recommended for code reviews) regardless of Claude's confidence level in identifying issues, use must

codereview using <model name>. This ensures the model is consulted even if Claude is *certain* about identified issues.

 [Read More](#) - Professional review workflow with step-by-step analysis

## 7. precommit - Pre-Commit Validation

Comprehensive review of staged/unstaged git changes across multiple repositories. This workflow tool guides Claude through systematic investigation of git changes, repository status, and file modifications across multiple steps before providing expert validation to ensure changes meet requirements and prevent regressions.

Perform a thorough precommit with o3, we want to only highlight critical issues, no blockers, no regressions. I need 100% confidence this covers all the edge cases listed in FIX\_ISSUE\_ABC.md

### Real Example:

- I asked claude (sonnet-4) to refactor some code to fix a known bug; gave it exact instructions
- The fix looked correct — and at a glance, so did the changes.
- More importantly, all existing unit tests passed (540+ unit tests) without a hitch. Zero regressions. Claude seemed confident and eager to commit. Everything looked perfect.

I then ran:

Run a precommit with o3 confirm our changes are sound and diffs are valid. Confirm this won't cause breakage or regressions and codesmells are out

This passed the tracked and untracked changes / diffs and the related code to o3 for a thorough review via Zen's precommit tool.

Output:

#### ● Precommit Analysis Results - CRITICAL ISSUE FOUND

The O3 model has identified one HIGH-severity regression and several medium/low issues in our refactor. Here's the summary:

 HIGH SEVERITY - MUST FIX BEFORE COMMIT

...

The reported issue was in fact a *very subtle bug* that slipped through the quick glance — and a unit test for this exact case apparently was missing (out of 540 existing tests!) - explains the zero reported regressions. The fix was ultimately simple, but the fact Claude (and by extension, I) overlooked this, was a stark reminder: no number of eyeballs is ever enough. Fixed the issue, ran precommit with o3 again and got:

### RECOMMENDATION: PROCEED WITH COMMIT

Nice! This is just one instance - take a look at [another example here](#).

**Tips:** \* To avoid extra API costs, add do not use another model to run the entire precommit workflow locally. \* If you want to **always** consult an external model (recommended for pre-commit analysis) regardless of Claude's confidence level in identifying issues, use must precommit using <model name>. This ensures the model is consulted even if Claude is *certain* about identified issues.

 [Read More](#) - Multi-repository validation and change analysis

## 8. debug - Expert Debugging Assistant

Systematic investigation-guided debugging that walks Claude through step-by-step root cause analysis. This workflow tool enforces a structured investigation process where Claude performs methodical code examination, evidence collection, and hypothesis formation across multiple steps before receiving expert analysis from the selected AI model. When Claude's confidence reaches **100% certainty** during the investigative workflow, expert analysis via another model is skipped to save on tokens and cost, and Claude proceeds directly to fixing the issue.


See logs under /Users/me/project/diagnostics.log and related code under the sync folder.

Logs show that sync works but sometimes it gets stuck and there are no errors displayed to the user. Using zen's debug tool with gemini pro, find out why this is happening and what the root cause is and its fix

**Tips:** \* To avoid extra API costs, add do not use another model to run the entire debugging workflow locally. This is recommended in most cases, as Claude typically identifies the root cause with high confidence by the end. \* If you want to **always** consult an external model regardless of Claude's confidence level, use must debug using <model name>. This ensures the model is consulted even if Claude is *certain* about the issue.

When in doubt, you can always follow up with a new prompt and ask Claude to share its findings with another model:


Use continuation with thinkdeep, share details with o4-mini to find out what the best fix is for this

 [Read More](#) - Step-by-step investigation methodology with workflow enforcement

## 9. analyze - Smart File Analysis

General-purpose code understanding and exploration. This workflow tool guides Claude through systematic investigation of code structure, patterns, and architectural decisions across multiple steps, gathering comprehensive insights before providing expert analysis for architecture assessment, pattern detection, and strategic improvement recommendations.

Use gemini to analyze main.py to understand how it works


 [Read More](#) - Comprehensive analysis workflow with step-by-step investigation

## 10. refactor - Intelligent Code Refactoring

Comprehensive refactoring analysis with top-down decomposition strategy. This workflow tool enforces systematic investigation of code smells, decomposition opportunities, and modernization

possibilities across multiple steps, ensuring thorough analysis before providing expert refactoring recommendations with precise implementation guidance.

Use gemini pro to decompose my\_crazy\_big\_class.m into smaller extensions

 [Read More](#) - Workflow-driven refactoring with progressive analysis

## 11. tracer - Static Code Analysis Prompt Generator

Creates detailed analysis prompts for call-flow mapping and dependency tracing. Generates structured analysis requests for precision execution flow or dependency mapping.


Use zen tracer to analyze how UserManager.authenticate is used and why

 [Read More](#) - Prompt generation and analysis modes

## 12. testgen - Comprehensive Test Generation

Generates thorough test suites with edge case coverage based on existing code and test framework. This workflow tool guides Claude through systematic investigation of code functionality, critical paths, edge cases, and integration points across multiple steps before generating comprehensive tests with realistic failure mode analysis.


Use zen to generate tests for User.login() method

 [Read More](#) - Workflow-based test generation with comprehensive coverage

## 13. secaudit - Comprehensive Security Audit

Systematic OWASP-based security assessment with compliance evaluation. This workflow tool guides Claude through methodical security investigation steps with forced pauses between each step to ensure thorough vulnerability assessment, security pattern analysis, and compliance verification before providing expert analysis.

Perform a secaudit with o3 on this e-commerce web application focusing on payment processing security and PCI DSS compliance

 [Read More](#) - OWASP Top 10 analysis with compliance framework support

## 14. docgen - Comprehensive Documentation Generation

Generates thorough documentation with complexity analysis and gotcha identification. This workflow tool guides Claude through systematic investigation of code structure, function complexity, and documentation needs across multiple steps before generating comprehensive documentation that includes algorithmic complexity, call flow information, and unexpected behaviors that developers should know about.

```
# Includes complexity Big-O notation, documents dependencies /  
code-flow, fixes existing stale docs  
Use docgen to documentation the UserManager class
```

```
# Includes complexity Big-O notation, documents dependencies /
```

code-flow

Use docgen to add complexity analysis to all the new swift functions I added but don't update existing code

 [Read More](#) - Workflow-based documentation generation with gotcha detection

## 15. listmodels - List Available Models

Display all available AI models organized by provider, showing capabilities, context windows, and configuration status.


Use zen to list available models

 [Read More](#) - Model capabilities and configuration details

## 16. version - Server Information

Get server version, configuration details, and system status for debugging and troubleshooting.

What version of zen do I have

 [Read More](#) - Server diagnostics and configuration verification

For detailed tool parameters and configuration options, see the [Advanced Usage Guide](#).

## Prompt Support

Zen supports powerful structured prompts in Claude Code for quick access to tools and models:

### Tool Prompts

- /zen:chat ask local-llama what 2 + 2 is - Use chat tool with auto-selected model
- /zen:thinkdeep use o3 and tell me why the code isn't working in sorting.swift - Use thinkdeep tool with auto-selected model
- /zen:planner break down the microservices migration project into manageable steps - Use planner tool with auto-selected model
- /zen:consensus use o3:for and flash:against and tell me if adding feature X is a good idea for the project. Pass them a summary of what it does. - Use consensus tool with default configuration
- /zen:codereview review for security module ABC - Use codereview tool with auto-selected model
- /zen:debug table view is not scrolling properly, very jittery, I suspect the code is in my\_controller.m - Use debug tool with auto-selected model
- /zen:analyze examine these files and tell me what if I'm using the CoreAudio framework properly - Use analyze tool with auto-selected model
- /zen:docgen generate comprehensive documentation for the UserManager class with complexity analysis - Use docgen tool with auto-selected model

## Continuation Prompts

- `/zen:chat` continue and ask gemini pro if framework B is better - Continue previous conversation using chat tool

## Advanced Examples

- `/zen:thinkdeeper` check if the algorithm in `@sort.py` is performant and if there are alternatives we could explore
- `/zen:planner` create a step-by-step plan for migrating our authentication system to OAuth2, including dependencies and rollback strategies
- `/zen:consensus` debate whether we should migrate to GraphQL for our API
- `/zen:precommit` confirm these changes match our requirements in `COOL_FEATURE.md`
- `/zen:testgen` write me tests for class ABC
- `/zen:docgen` document the payment processing module with gotchas and complexity analysis
- `/zen:refactor` propose a decomposition strategy, make a plan and save it in `FIXES.md`

## Syntax Format

The prompt format is: `/zen: [tool] [your_message]`

- `[tool]` - Any available tool name (chat, thinkdeep, planner, consensus, codereview, debug, analyze, docgen, etc.)
- `[your_message]` - Your request, question, or instructions for the tool

**Note:** All prompts will show as “(MCP) [tool]” in Claude Code to indicate they’re provided by the MCP server.

## Advanced Features

### AI-to-AI Conversation Threading

This server enables **true AI collaboration** between Claude and multiple AI models, where they can coordinate and build on each other’s insights across tools and conversations.

 [Read More](#) - Multi-model coordination, conversation threading, and collaborative workflows

## Configuration


Configure the Zen MCP Server through environment variables in your `.env` file. Supports multiple AI providers, model restrictions, conversation settings, and advanced options.

```
# Quick start - Auto mode (recommended)
DEFAULT_MODEL=auto
GEMINI_API_KEY=your-gemini-key
OPENAI_API_KEY=your-openai-key
```



DIAL\_API\_KEY=your-dial-key # Optional: Access to multiple models via DIAL

**Key Configuration Options:** - **API Keys:** Native APIs (Gemini, OpenAI, X.AI), OpenRouter, DIAL, or Custom endpoints (Ollama, vLLM) - **Model Selection:** Auto mode or specific model defaults - **Usage Restrictions:** Control which models can be used for cost control - **Conversation Settings:** Timeout, turn limits, memory configuration - **Thinking Modes:** Token allocation for extended reasoning - **Logging:** Debug levels and operational visibility

 [Read More](#) - Complete configuration reference with examples

## Testing

For information on running tests, see the [Testing Guide](#).

## Contributing

We welcome contributions! Please see our comprehensive guides: - [Contributing Guide](#) - Code standards, PR process, and requirements - [Adding a New Provider](#) - Step-by-step guide for adding AI providers

## License

Apache 2.0 License - see LICENSE file for details.

## Acknowledgments

Built with the power of **Multi-Model AI** collaboration 🧠 - **Actual Intelligence** by real Humans - [MCP \(Model Context Protocol\)](#) by Anthropic - [Claude Code](#) - Your AI coding assistant & orchestrator - [Gemini 2.5 Pro & 2.0 Flash](#) - Extended thinking & fast analysis - [OpenAI O3](#) - Strong reasoning & general intelligence

Star History

History

