Code Mode without the Code

Presenting: mcpGraph - a no code "Code Mode" solution — Fast, secure, reliable, observable MCP tool aggregation and orchestration

MCP
Dev Summit
North America

Bob Dickinson · Friday, April 3, 2026

Meet Bob Dickinson

Background

- First-line-of-code founder with several exits
- CTO, most recently OneLogin & Censys

Currently

- Building Agent frameworks & MCP security tooling at **TeamSpark.ai**
- Fractional CTO and advisor
- Open Source: Maintainer of MCP Registry and MCP Inspector

The Origin of Code Mode: CodeAct

> *Executable Code Actions Elicit Better LLM Agents* — Jun 2024, published on the Apple Machine Learning blog

Agents are better at writing code to execute algorithms than they are at executing algorithms (especially when involving elements like looping, conditional logic, and data transformation).

LLMs are better at one of these than the other…

How many R's are in the word "strawberry"?

Write a Python script to count the occurrences of a given character in a string

The "Code Mode" Revolution with MCP

Cloudflare

Code Mode: the better way to use MCP

Sep 26, 2025

Anthropic

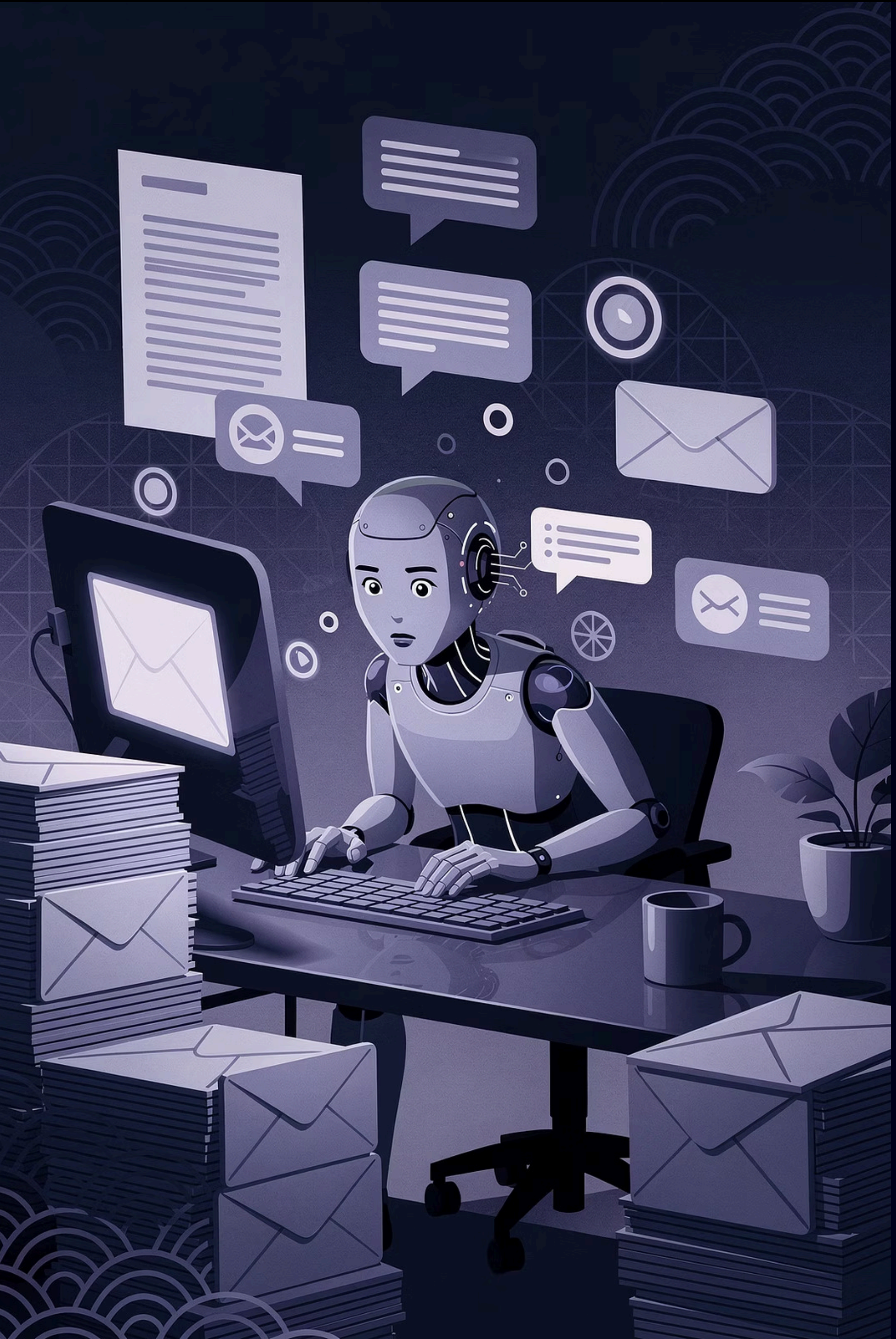
Code execution with MCP: Building more efficient agents

Nov 4, 2025

Docker

Dynamic MCPs with Docker: Stop Hardcoding Your Agents' World

Nov 6, 2025



Traditional LLM Tool Calling is Problematic for Workflows

Example

"Read transcript from Google Docs and add to Salesforce prospect"

Dynamic Orchestration

Agents aren't good at consistently picking the right tools, and calling them in the right order and the right way (transforming outputs and mapping to inputs).

Context Overload

Tool definitions and intermediate results flood the context window — costing time, money, and accuracy.

The Two Faces of Context Overload

Tool Catalog Bloat

Every tool from every MCP server, including all tool descriptions, parameters, and metadata, is inserted into the LLM's context.

This problem is largely solved through semantic tool inclusion strategies

Tool Calling Bloat

Each actual tool call, including all tool inputs and outputs, are inserted into the LLM's context.

This is a persistent challenge, can be addressed to some extent with sub-agents.

Why We Care about Context Bloat

💸 Cost: Time & Money

More tokens = slower responses and higher API costs

🧠 Distracted Agents

Irrelevant context reduces accuracy and response quality

Code Mode to the Rescue

What is Code Mode?

Agents create higher-level tools in code that orchestrate other MCP tools — building wrappers that replace sprawling tool calls with a single, clean interface.

The Result

A single `moveTranscriptToProspect` tool — reliably calls the right tools in the right sequence, produces concise status, no tool pollution, no intermediate context. It also runs faster and cheaper.

From the Anthropic Paper

"Code execution with MCP improves context efficiency"

98.7%

token reduction in their example

The Problem with Code Mode

Hint: It's right there in the name ;)

LLM Generates and Executes Code

Letting the agent generate and the directly run code makes YOLO mode look safe by comparison (also bypasses tool permissions).

Sandboxing Isn't Enough

Even with sandboxing and hardened tools, you are still running arbitrary, unreviewed code against powerful MCP tool sets.

Inspection and Detection is Challenging

Difficult to observe and detect problems through casual review or existing tooling. Code is intentionally outside of controls, spread across many files. Artifacts can be ephemeral.

Code is Still Code

- Who maintains it?
- Is it in a language we support, using tooling we support?
- Does it comply with coding standards, is it in version control?
- Has it been reviewed, approved, and tested for bugs?
- Does it pass static/dynamic security analysis and security review?
- How does it perform, will it scale?

What About a "No Code" Code Mode?

Orchestrates MCP

MCP tool calls with data transformation and conditional logic

Predictable and Complete

100% reliable, fast, and solves all code mode examples and more

Observable

Inspectable and analyzable by humans, LLMs, or tools

Simple

Easily authored by humans or LLMs alike

Alternative: Enterprise Workflow Solutions

For certain use cases, established enterprise workflow systems like **n8n**, **Zapier**, and **Workato** offer robust solutions. These platforms empower users to build sophisticated automation flows without writing code, and can be integrated into agent workflows.

Advanced Orchestration

Create complex sequences, including conditional logic and data transformations, to call and manage MCP tools.

Enterprise-Grade Benefits

These platforms offer enhanced security, traceability, audit trails, and dedicated support, crucial for large organizations.

Considerations

Typically closed-source and require paid subscriptions, which may not align with all project budgets or open-source initiatives.

Our No Code Solution Prerequisites

Principles

- **Open Source**
- **No embedded programming language**
- **Composable by Agents out-of-the-box** — simple, familiar syntax; no training or fine-tuning required

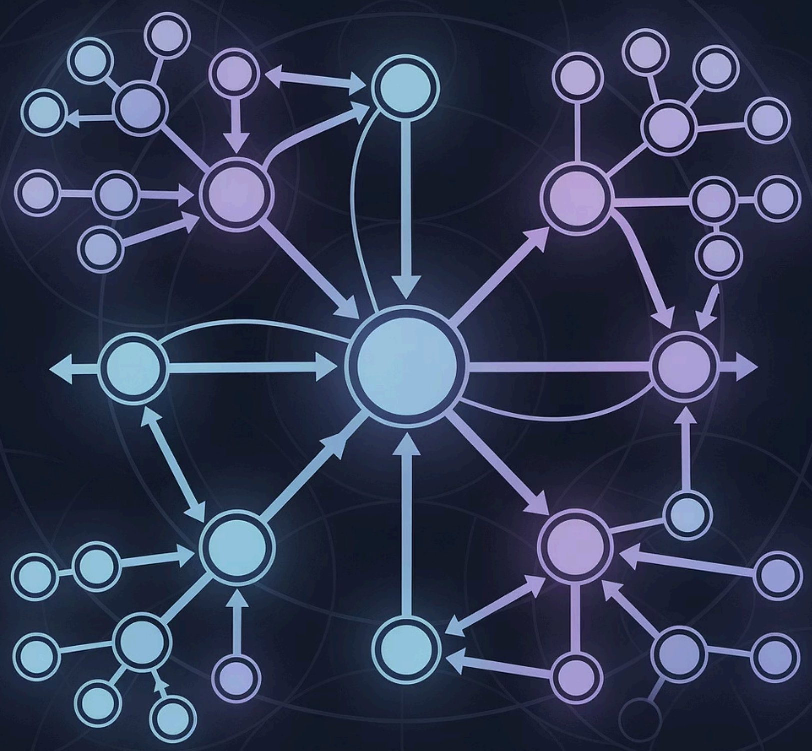
Building Blocks

JSONata

Data transformation — filtering, formatting, extraction

JSON-Logic

Conditional logic and branching



Introducing mcpGraph

Domain specific language (DSL) for MCP tool aggregation and orchestration, with tools implemented as directed graphs

Declarative Configuration

Define MCP servers, tools, and execution graphs in YAML

Data Transformation

JSONata expressions to transform data between nodes

Conditional Routing

JSON Logic for conditional branching

Observable

Every tool call, transformation, and decision is traceable

mcpGraph — Get Started

NPM Packages

```
npm install -g mcpgraph mcpgraph-ux
```

- npmjs.com/package/mcpgraph
- npmjs.com/package/mcpgraph-ux

GitHub

- github.com/TeamSparkAI/mcpGraph
- github.com/TeamSparkAI/mcpGraph-ux

Open source and freely available under the MIT license.

mcpGraph YAML Structure

A single YAML file defines an MCP server, the tools it uses, and the tools it exports.

```yaml
server:
  # MCP server metadata

mcpServers:
  # MCP server configurations (servers used by tools)

tools:
  # Exported MCP tool definitions, with execution graphs
```

Sample mcpGraph: Server Config

A minimal YAML definition of the server metadata:

```yaml
version: "1.0"

# MCP Server Metadata
server:
  name: "fileUtils"
  version: "1.0.0"
  description: "File utilities"
```

Sample mcpGraph: Imported MCP Servers

```yaml
# MCP Servers used by the graph
mcpServers:
  filesystem:
    command: "npx"
    args:
      - "-y"
      - "@modelcontextprotocol/server-filesystem"
      - "/Users/bob/Documents/GitHub/mcpGraph/tests/files"
```

Sample mcpGraph: Exported Tools

```yaml
tools:
 - name: "count_files"
   description: "Counts the number of files in a directory"
   inputSchema:
     type: "object"
     properties:
       directory:
         type: "string"
         description: "The directory path to count files in"
     required:
     - directory
   outputSchema:
     type: "object"
     properties:
       count:
         type: "number"
         description: "The number of files in the directory"
   nodes: ## Node graph goes here
```

mcpGraph Node Types

Tools implemented as a **directed graph (non-acyclic, with guardrails)**

entry

Process exported MCP tool call with input parameters

mcp

Call an MCP tool

transform

Filter, summarize, extract, reformat data

switch

Conditionally route to next node

exit

Return data as output parameters

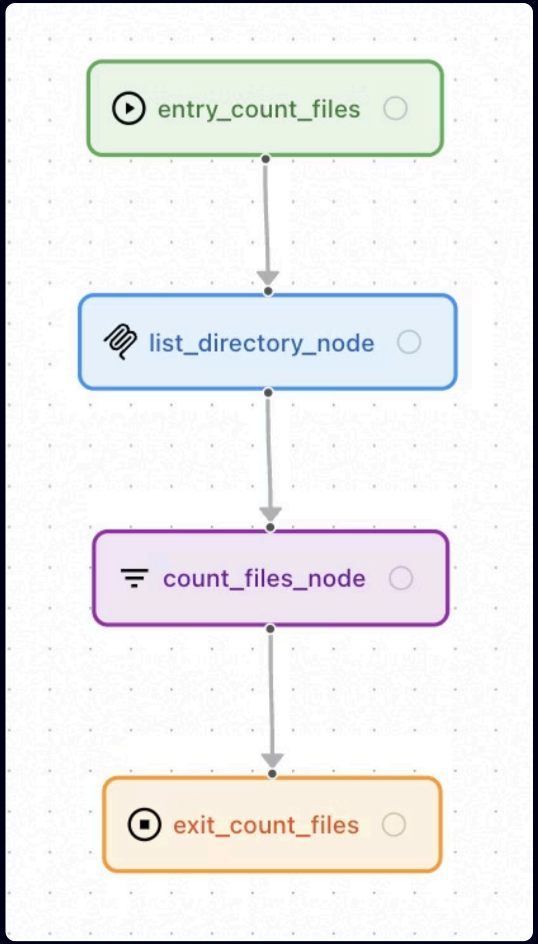
Sample mcpGraph: Graph Nodes

```yaml
nodes:
- id: "entry_count_files"
  type: "entry"
  next: "list_directory_node"

- id: "list_directory_node"
  type: "mcp"
  server: "filesystem"
  tool: "list_directory"
  args:
    path: "$.entry_count_files.directory"
  next: "count_files_node"

- id: "count_files_node"
  type: "transform"
  transform:
    expr: |
      { "count": $count($split($.list_directory_node, "\n")) }
  next: "exit_count_files"

- id: "exit_count_files"
  type: "exit"
```



Running mcpGraph

mcpGraph is itself an MCP server. Install and configure as an MCP server in your client config:

```
npm install -g mcpgraph

{
  "mcpServers": {
    "mcpgraph": {
      "command": "mcpgraph",
      "args": [
        "-g",
        "/path/to/your/graph.yaml"
      ]
    }
  }
}
```

mcpGraph UX

Graph Runner

Execute graphs interactively

Visualization

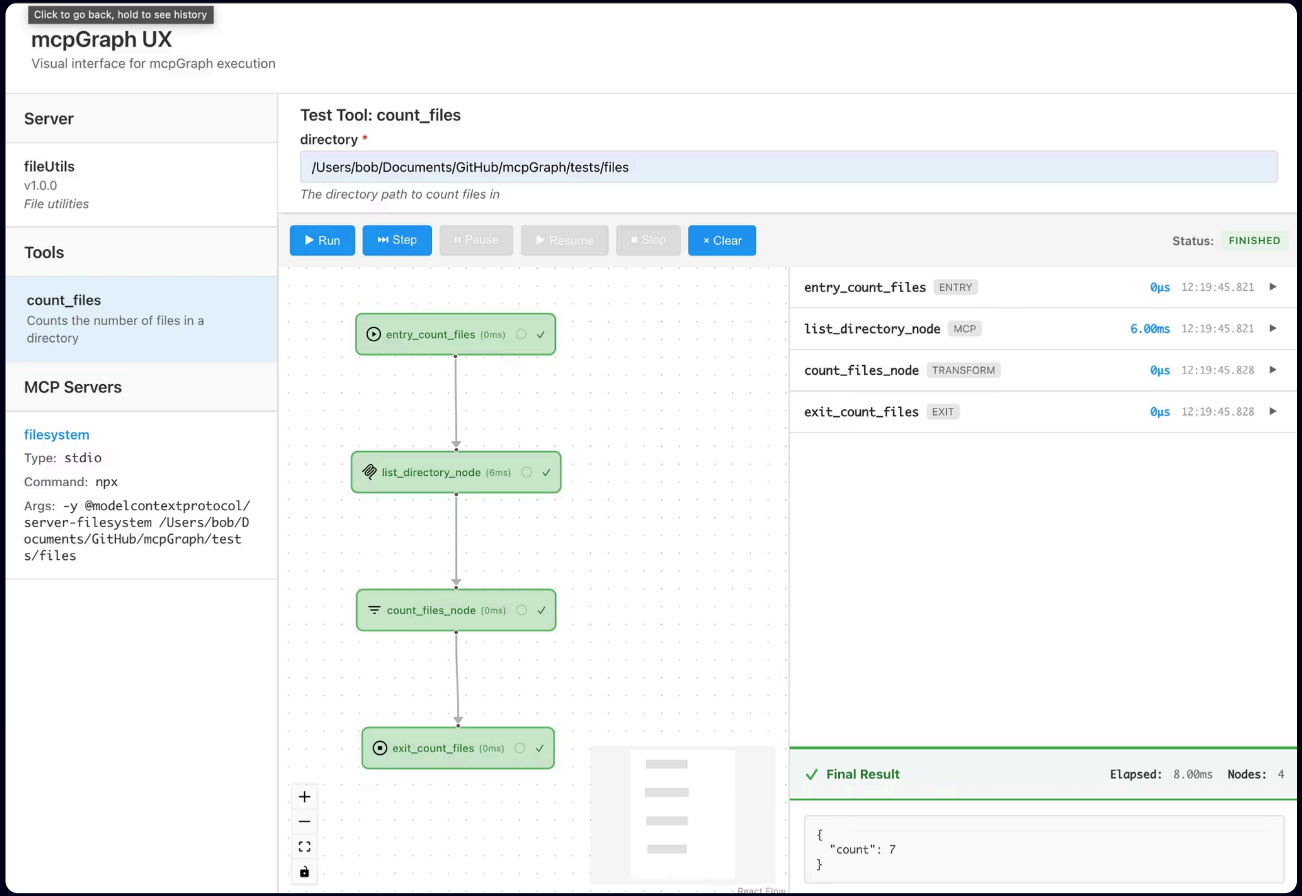
See the graph structure live

Detailed Tracing

Full observability of every step

Debugging Controls

Step, breakpoint, and more



mcpGraph: Virtual MCP Server with Workflows

Beyond integrating and orchestrating existing tools, mcpGraph provides robust features for customizing and combining tool functionalities to address specific needs.

Tool Composition

Combine diverse tools from multiple MCP servers to create powerful new tool sets for specific use cases.

Tool Reshaping

Modify existing tool names, descriptions, and other metadata to better suit new contexts and use cases.

Input/Output Remapping

Dynamically adjust how data flows into and out of tools using flexible JSONata expressions.

Example: Bespoke Filesystem Server

Develop focused versions of tool sets, such as a filesystem server with read-only tools, all with structured output.

Conclusion

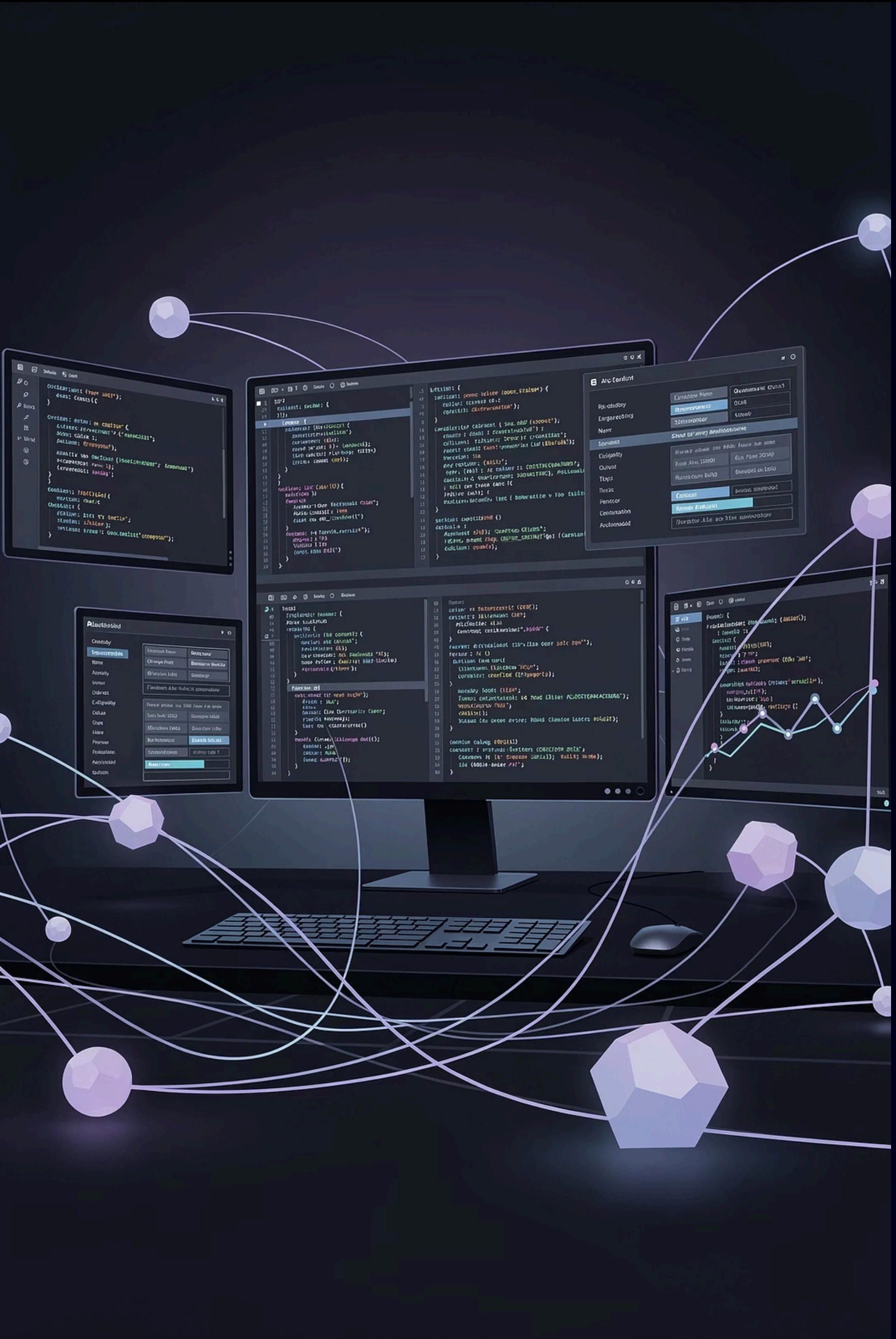
Code Mode without the Code.

✅ All the Benefits of Code Mode

- Build powerful and predictable workflows
- Provide the exact context needed
- Higher performance and minimizes token use

🚫 Without the Code

- No arbitrary, unreviewed code
- Easily observed, visualized, tested, and audited



But wait, how do agents actually build Code Mode solutions?

How does the agent…

- Understand available MCP servers and their configuration
- Know how to structure the code and where to place artifacts
- Test the new MCP server they've built
- Install the new MCP server into their own environment

Agents building with mcpGraph

Dev Environment

- **SKILL.md** - detailed documentation and examples of mcpGraph
- **mcpgraph** MCP server - to run mcpGraphs as MCP servers

Results

- With very significant and detailed prompting (including details on MCP server configuration) agents could usually build working mcpGraphs
- Agents struggled with tooling (esp how to test)
- Significant variability depending on complexity of task, quality of MCP servers

mcpGraphToolkit

What if we gave the agent an mcpGraph development environment?

Interact with graph

Inspect, add, update, remove tools

Understand available MCP servers/tools

Test building blocks

MCP tool, JSONata, JSON Logic

Test graph tools

With logging, detailed errors

Deploy graph tools

Run deployed graph tools

Agents building with mcpGraphToolkit

Dev Environment

- **SKILL.md** - detailed documentation of mcpGraphToolkit usage
- **mcpgraphtoolkit** - MCP server to test, build, deploy, and run mcpGraph tools

Note: Refined with several DevRel cycles (test, collect feedback, update)

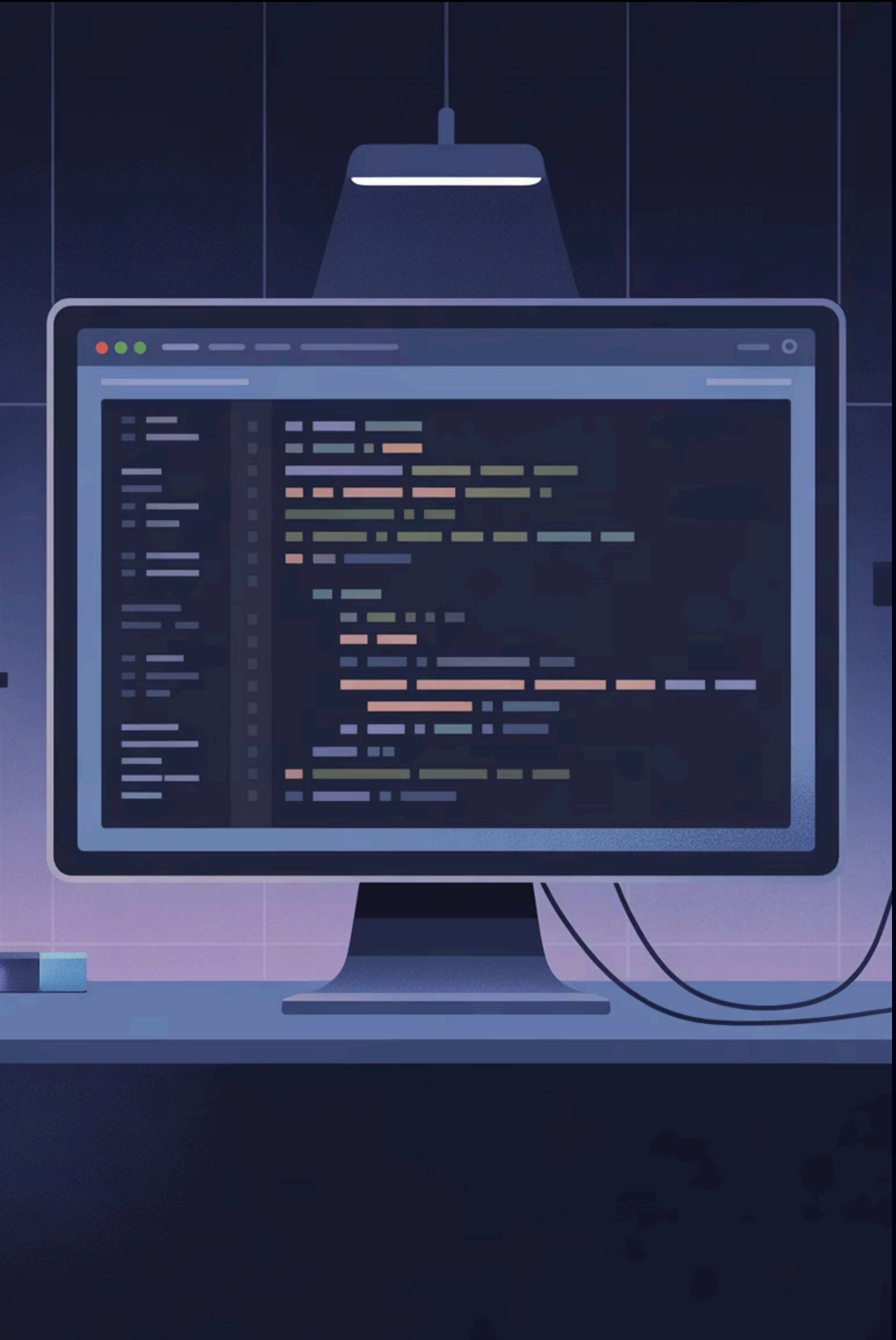
Results

100%

Success rate

Few wrong turns

- Agent can build and run tools on demand
- Tools are then available for direct use going forward



In Practice - Claude Code

The mcpGraph repo contains a fully configured Claude Code test workspace

mcpgraphtoolkit **SKILL.md** pre-installed and pre-approved

mcpgraphtoolkit MCP server pre-installed and pre-approved

Starting graph with **fetch** and **filesystem** MCP servers available

Run Claude Code in `/testClaude`

> **Prompt Example:**
>
> *Create an mcpGraph tool to download a URL to a local file in the downloads directory of this project and return the filename and size*

Conclusion

> **mcpGraph brings the benefits of Code Mode to every agent — without the code.**

mcpGraph

Define and run MCP workflows reliably and securely using a simple, open YAML DSL

mcpGraph UX

Visualize, validate, test, and debug mcpGraphs with an intuitive visual interface

mcpGraphToolkit

A complete development environment for LLMs to create, validate, and deploy mcpGraphs