GITHUB COPILOT AGENTIC FRAMEWORK

# Configuration hierarchy: how config files shape your agents.

Five cascading layers, from copilot-instructions.md to prompt files. A technical briefing on how GitHub Copilot assembles context, how precedence works, and how to design a configuration stack your whole team inherits.

AUTHOR
Paula Silva

ROLE
Software Global Black Belt

DURATION
45 to 60 minutes

DATE
2026-06-10

AGENDA

# Five parts, one mental model.

| | |
|---|---|
| PART I | Foundation, why configuration matters and the cascade mental model |
| PART II | The five layers, file by file, with real content you can copy today |
| PART III | Resolution, the algorithm GitHub Copilot runs and the precedence rules behind it |
| PART IV | Practice, the token economy, a full project tree, enterprise scenarios, and the do or do not list |
| PART V | Memory, how GitHub Copilot learns the repo on its own and when to graduate it to files |

PART

# The foundation.

Modern AI assistants read a hierarchy of files to assemble the context behind every single response.

WHY CONFIGURATION MATTERS

# Configuration decides what the AI knows, what it can do, and how it behaves.

DIMENSION 01 · KNOWLEDGE

### What the AI knows

Project instructions, code conventions, architecture patterns, and the key commands of your repository.

DIMENSION 02 · PERMISSION

### What the AI can do

Allowed tools, access limits, and the security guardrails that keep agents inside the lines.

DIMENSION 03 · BEHAVIOR

### How the AI behaves

Scoped rules per area, the personality of specialized agents, response tone and format.

WITHOUT THE HIERARCHY

Teams end up with duplicated instructions, conflicting rules, or an AI that ignores critical conventions because the right file was never created.

THE MENTAL MODEL

# Think like CSS: a cascade of specificity.

CSS CASCADE

`body { }`
Global style, applies everywhere

`.container { }`
Component level, narrower scope

`#hero { }`
Most specific wins the conflict

CONFIG CASCADE

`copilot-instructions.md`
Repo-level global context

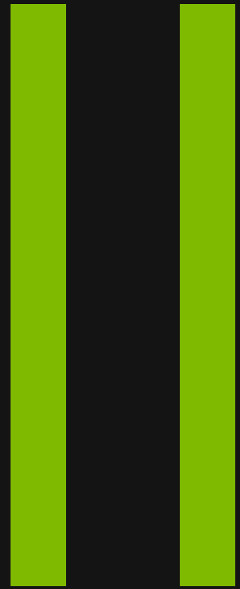
`.instructions.md`
Folder or file pattern scope

`*.agent.md · SKILL.md`
Most specific complements the rest

GOLDEN RULE

A child layer ADDS, it never contradicts. If the global file says "use TypeScript", a scoped file cannot say "use JavaScript". Specificity complements, it never overrides.

PART



# The five layers.

A deep dive into each file type, with real content, real locations, and the activation rule of each one.

LAYER 1 OF 5 · ALWAYS ON

# copilot-instructions.md is the global brain of the repository.

.GITHUB/COPILOT-INSTRUCTIONS.MD

```
# Project: E-Commerce Platform

## Tech Stack
- Next.js 14 with App Router
- TypeScript strict mode
- Prisma ORM with PostgreSQL

## Conventions
- kebab-case for file names
- PascalCase for React components
- All API routes in app/api/
- Tests: Vitest, run with npm test

## Architecture
- Feature-based folder structure
- Server components by default
```

LOCATION

.github/copilot-instructions.md

ACTIVATION

Automatic in every GitHub Copilot session. Always active, no frontmatter needed.

WHY IT MATTERS

The single most impactful file you can create. It sets the tone for every interaction with your project.

LAYER 2 OF 5 · SCOPED BY GLOB

# `.instructions.md` applies rules only where the glob matches.

LOCATION

`.github/instructions/*.instructions.md`

ACTIVATION

The applyTo frontmatter holds a glob pattern. Rules load only when a file in context matches it.

USE IT WHEN

Different parts of the project need different rules: React rules for components, API rules for routes, test rules for specs.

.GITHUB/INSTRUCTIONS/REACT-COMPONENTS.INSTRUCTIONS.MD

```
---
applyTo: "src/components/**/*.tsx"
---

# React Component Rules

- Functional components with hooks
- TypeScript interfaces for all props
- JSDoc comments on public props
- Tailwind for styling, no CSS modules
- Unit tests in __tests__/ folder
- Components must stay under 200 lines
```

LAYER 3 OF 5 · ACTIVE WHEN INVOKED

# *.agent.md defines specialized personas with their own tools.

.GITHUB/AGENTS/SECURITY-REVIEWER.AGENT.MD

```
---
description: "Security code reviewer"
tools: [codeql, semgrep, gh-advanced-security]
---

# Security Reviewer Agent

You are a security code reviewer. Analyze:
- SQL injection vulnerabilities
- XSS attack vectors
- Authentication bypass risks

Always provide:
- Severity (Critical/High/Medium/Low)
- OWASP category reference
- Remediation code example
```

IDENTITY

A name, a description, and a unique behavior the model adopts when the agent is active.

TOOLS

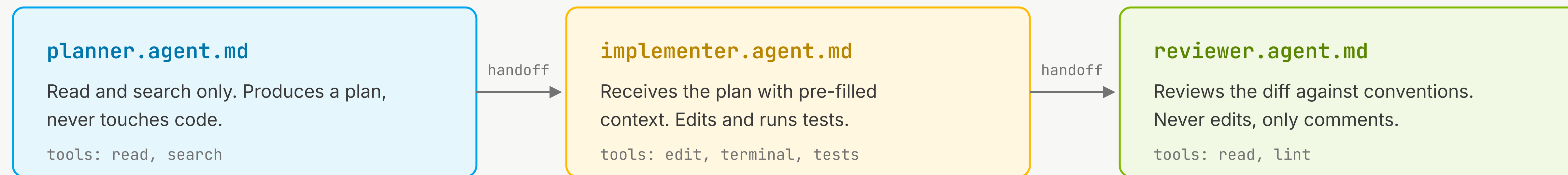
An explicit allowlist of tools the agent may call. Everything else is off the table.

SCOPE

Focus on one area or task type. The active agent narrows the context to its own domain.

LAYER 3 EXTENDED · HANDOFFS

# Agents chain together: plan, implement, review as a pipeline.




When each agent finishes, a handoff button carries the context to the next one. The flow becomes process, not improvisation.

*Custom agents (.agent.md) replace the old chat modes: persona, tool allowlist and chained handoffs. On Business and Enterprise plans, definitions can be shared at the organization level: every repo inherits the same specialists.*

LAYER 4 OF 5 · GLOB OR ALWAYSAPPLY

# SKILL.md packages a complete domain, not just rules.

LOCATION

.github/skills/<name>/SKILL.md

ANATOMY

Rules plus workflows plus templates plus quality checklists. The structure forces completeness.

SKILLS VS INSTRUCTIONS

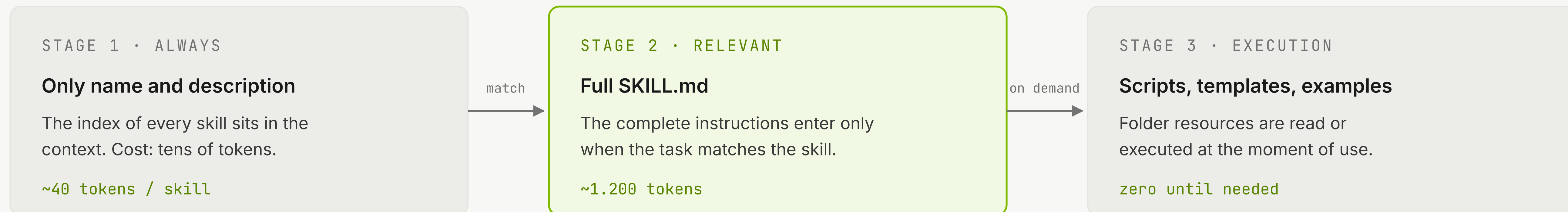
Instructions are simple scoped rules. Skills are complete domain packages. Reach for a skill when the domain is complex.

.GITHUB/SKILLS/API-DESIGN/SKILL.MD

```
---
name: "REST API Design"
globs: ["src/api/**/*.ts", "src/routes/**/*.ts"]
alwaysApply: false
---

# REST API Design Skill

## Rules
- Plural nouns: /users, not /user
- Version APIs: /api/v1/resources
- Cursor-based pagination

## Workflow
1. Define resource schema  2. Route handlers
3. Validation middleware   4. Integration tests

## Quality checklist
- [ ] Proper status codes on all endpoints
- [ ] Input validation on POST and PUT
- [ ] OpenAPI docs generated
```

LAYER 4 EXTENDED · PROGRESSIVE LOADING

# Skills load in three stages. The context only pays for what it uses.




It is the same principle as the three memory tiers: pay tokens for what the task needs, not for what the repository has.

OPEN, PORTABLE STANDARD `VS Code` `GitHub Copilot CLI` `coding agent` `~/.copilot/skills personal`

LAYER 5 OF 5 · ON DEMAND

# .prompt.md turns repeated requests into reusable templates.

.GITHUB/PROMPTS/CREATE-COMPONENT.PROMPT.MD

```
---
mode: "agent"
description: "Create a new React component"
variables:
  - name: "componentName"
  - name: "type"   # page, layout, widget
---

Create a React component called {{componentName}}.
Type: {{type}}

Steps:
1. Create the file in src/components/
2. Add the TypeScript props interface
3. Add a unit test in __tests__/
4. Export from the barrel index.ts
```

AGENT MODE

mode: "agent" lets GitHub Copilot execute actions, create files, run tests. mode: "edit" only suggests changes.

VARIABLES

Double braces mark dynamic parameters. GitHub Copilot prompts for the values before executing.

LIFECYCLE

Ephemeral by design. A prompt file runs once per invocation and does not persist in context.

THE MAP IN ONE VIEW

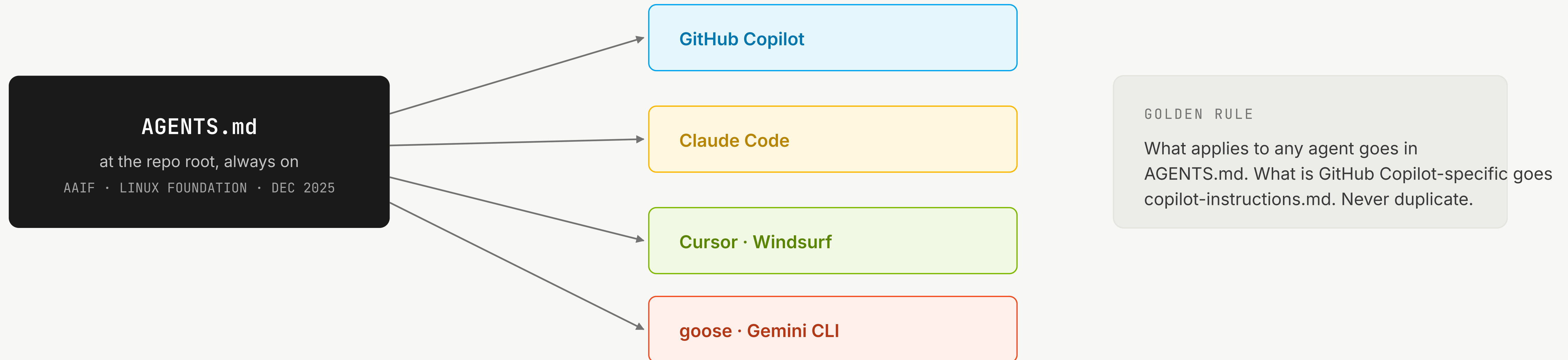
# Seven file types, one composition model.

| FILE | SCOPE | ACTIVATION | PURPOSE | FRONTMATTER |
|---|---|---|---|---|
| copilot-instructions.md | Entire repo | Automatic | Global conventions | none |
| AGENTS.md | Entire repo, multi-agent | Automatic | Open cross-tool standard | none |
| .instructions.md | Glob pattern | On glob match | Rules per area | applyTo |
| *.agent.md | Active agent | When invoked | Persona and tools | description, tools |
| SKILL.md | Glob or always | Glob or alwaysApply | Complete domain | name, globs, alwaysApply |
| .prompt.md | Invocation | On demand | Template with variables | mode, variables |
| .mcp.json | Project | Automatic | External tool connections | JSON, no frontmatter |

*Validate names and frontmatter against the official GitHub Copilot customization documentation. Conventions evolve fast.*

THE MAP IN MOTION · AGENTS.MD

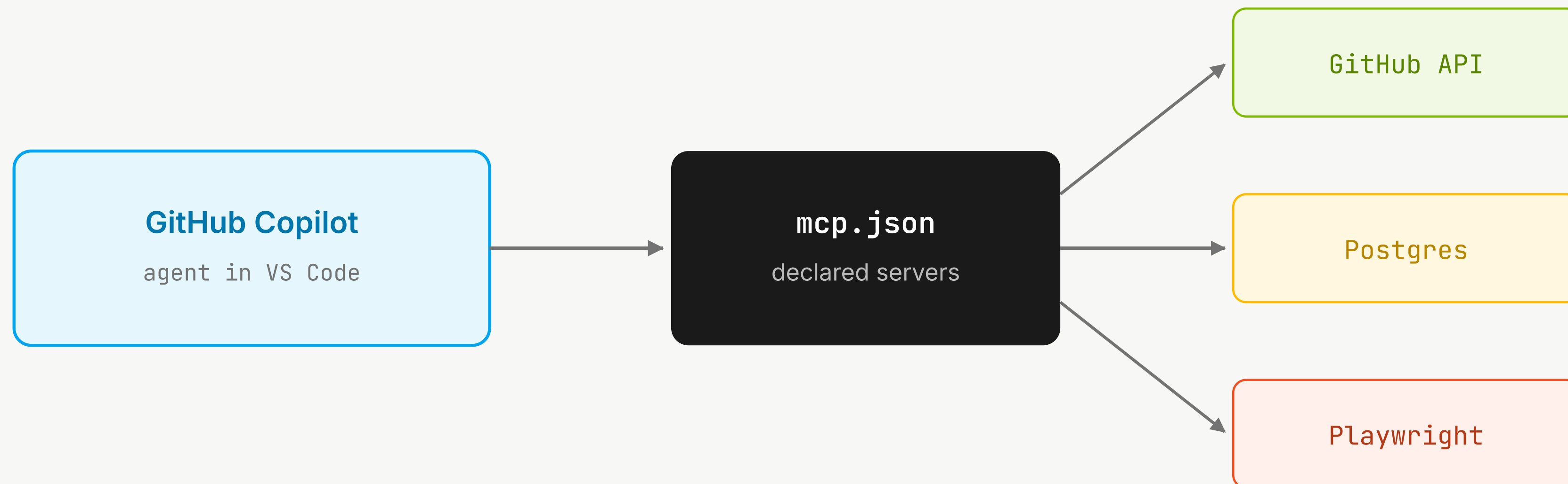
# AGENTS.md: one file, every agent. Now an open standard.




*Donated by OpenAI to the Agentic AI Foundation (Linux Foundation) in Dec 2025, alongside MCP and goose. VS Code reads AGENTS.md as an always-on instruction next to copilot-instructions.md: one is the open cross-tool standard, the other is the GitHub Copilot specific channel.*

THE MAP IN MOTION · MCP

# mcp.json connects agents to the world. And it became neutral infrastructure.




97M SDK downloads / month

10.000+ published servers

spec 2025-11-25

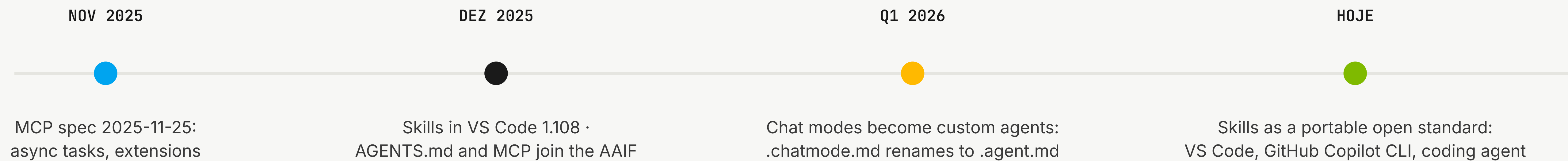
.VSCODE/MCP.JSON

```
// servers que o agente pode usar
{
  "servers": {
    "github": {
      "url": "https://api.githubcopilot.com/mcp/"
    },
    "postgres": {
      "command": "npx",
      "args": ["-y", "@mcp/postgres"]
    }
  }
}
```

*Model Context Protocol: spec 2025-11-25 (async tasks, extensions, elicitation) and governance under the Agentic AI Foundation since Dec 2025. For the hierarchy, the rule is the same as the other layers: declare it in the repo, version it in Git, and the whole team inherits the same connections.*

THE MAP IN MOTION · LAST 6 MONTHS

# What changed since December. Validate before you standardize.




*The operational lesson: file names and frontmatter change in cycles of months. Treat the official customization docs as the source of truth, schedule a quarterly stack review, and rename .chatmode.md to .agent.md in the next cycle.*

PART

# Resolution.

The seven-step algorithm GitHub Copilot runs to assemble context, and the precedence rules that resolve conflicts.

THE RESOLUTION ORDER

# Seven steps, each one ADDS to the accumulated context.

| | | |
|---|---|---|
| 01 | `copilot-instructions.md` | Load the global repo context first, in every session: copilot-instructions.md and, when present, AGENTS.md. |
| 02 | `files in context` | Identify which files are part of the current conversation. |
| 03 | `*.instructions.md` | Apply every instruction file whose glob matches those files. |
| 04 | `*.agent.md` | Load the definition of the active agent, when one is invoked. |
| 05 | `SKILL.md` | Activate skills by glob match or alwaysApply: true. |
| 06 | `*.prompt.md` | Expand the prompt template with its variables, when invoked. |
| 07 | `.mcp.json` | Aggregate the available MCP connections into the final context. |

RESOLUTION · FOUR GOTCHAS

# Four behaviors nobody expects, and every team discovers late.

CODE REVIEW · BASE BRANCH

### Review reads instructions from the base branch, not yours

On a pull request, the code review agent uses the base branch's copilot-instructions.md. New rules in the feature branch only apply after merge. Intentional: a branch cannot rewrite its own review rules.

COMPLETIONS · OUTSIDE THE SYSTEM

### Ghost text reads none of this

Inline suggestions while you type are a separate system: they read neither copilot-instructions.md nor instructions files. The hierarchy governs chat and agents, not autocomplete.

STACKING · NO WINNER

### Instructions stack, they do not compete

Every instructions file whose glob matches enters together, as a union. There is no override and no precedence between them. If two rules conflict, both reach the model. Resolving the conflict is your job, not GitHub Copilot's.

EXCLUDEAGENT · FINE TARGETING

### You can opt one agent out of one rule

The excludeAgent frontmatter opts a specific agent out of an instructions file: the security rule that is perfect in code review and too verbose in chat stays only where it helps.

*Behaviors documented in the official VS Code and GitHub Copilot customization docs. Worth pasting this slide in the team channel: every item here has already cost someone an afternoon of debugging.*

PRECEDENCE RULES

# There is no destructive override. Composition is always additive.

| Rule | Description | Example |
|---|---|---|
| **Specificity wins** | More specific rules complement the generic ones and prevail in practice when both touch the same point. | `.instructions.md > copilot-instructions.md` |
| **Additive, not substitutive** | Layers accumulate, they never replace each other. The final context carries all of them. | `folder rule + global rule = both` |
| **The agent isolates scope** | An active agent narrows the context to its own domain, tools, and identity. | `security agent focuses on vulnerabilities` |
| **The prompt is ephemeral** | Prompt files run on demand and do not persist. One invocation, one expansion. | `template runs once per invocation` |

TEST APPLYTO LIVE

RESOLUTION · BUILD YOUR STACK INTERACTIVE

# Toggle layers on and off. See what the agent receives.

SCENARIO: EDITING SRC/API/PAYMENTS.TS AND ASKING FOR A NEW ENDPOINT

- [x] `copilot-instructions.md` always on · ~800 tokens
- [ ] `AGENTS.md` always on, multi-agent · ~600
- [x] `api.instructions.md` applyTo: src/api/** matches · ~450
- [ ] `security.agent.md` if invoked · ~350
- [x] `api-design/SKILL.md` task match · ~1,200
- [ ] `new-endpoint.prompt.md` if called with / · ~500
- [x] `mcp.json` tool definitions · ~300

CONTEXT ASSEMBLED FOR THIS REQUEST

**2.750**

configuration tokens, before your prompt

READING

Healthy stack: global + scope + domain.

*Typical per-layer estimates, to build order-of-magnitude intuition. The point: configuration is also context paid per request. Curation is worth money.*

THE CONTEXT ROI

# Without a hierarchy, the team pays for the same context a thousand times a day.

**800 tok**

PASTED BY HAND INTO EVERY CHAT: STACK, CONVENTIONS, REPO PATTERNS

**40/dia**

CHATS PER DEV PER DAY, ON TEAMS RUNNING AGENTS FOR REAL

**27M**

TOKENS PER MONTH WASTED BY A TEAM OF 20, WITH NO HIERARCHY

The math: 800 tokens of repeated context, times 40 chats, times 20 devs, times 20 working days. The hierarchy turns those 32 million repeated tokens into versioned files, loaded once, inherited by everyone.

THE CONTEXT ROI · BEFORE AND AFTER

# The hierarchy does not just save tokens. It saves the work of remembering.

WITHOUT A HIERARCHY

- Each dev pastes stack and conventions into the prompt, every time.
- Context dies when the chat closes. Back to zero each session.
- Two devs explain the same thing in two different ways.

WITH A HIERARCHY

- ✓ Context lives in Git. Written once, inherited by all.
- ✓ Selective loading: the agent pays only for the layer the task triggers.
- ✓ One source of truth. Change the file, change it for the whole team.

The real win is not the token, it is consistency. When knowledge lives in the repo, the agent is good on a new dev's first day, not in their third month.

THE CONTEXT ROI · FIVE LEVERS

# Each platform mechanism cuts tokens in a different way.

| | | |
|---|---|---|
| `SKILL.md` | Progressive loading in three stages: only the index sits in context until the task matches. | 40 tok vs 1,200 |
| `*.instructions.md` | applyTo with a glob: the API rule only enters when you edit an API file. | pays by scope |
| `*.agent.md` | The agent narrows context to its own domain and a lean tool allowlist. | focused context |
| GitHub Copilot Spaces | Curation: attach the files that matter instead of the whole repo. Search, do not dump. | curated grounding |
| GitHub GitHub Copilot Memory | Learns the repo on its own and skips re-explanation: less prompt, less instructions maintenance. | zero re-prompt |

Five levers, one principle: the right context, at the right moment, paid once. Curation is not aesthetics, it is the FinOps line nobody looks at.

COMPOSITION IN ACTION

# One project, every layer in its place.

```
my-ecommerce · tree .github

my-ecommerce/
├── .github/
│   ├── copilot-instructions.md   # global: stack + conventions
│   ├── instructions/
│   │   ├── react.instructions.md  # scope: src/components/**
│   │   └── api.instructions.md    # scope: src/app/api/**
│   ├── agents/
│   │   ├── security.agent.md      # persona: security review
│   │   └── db-expert.agent.md     # persona: database expert
│   ├── skills/
│   │   ├── api-design/SKILL.md    # domain: REST patterns
│   │   └── testing/SKILL.md       # domain: test standards
│   └── prompts/
│       └── fix-bug.prompt.md      # template: debug + fix
├── .vscode/
│   └── mcp.json                   # external connections
└── src/
    ├── components/                # matched by react rules
    └── app/api/                   # matched by api rules
```

ONE PURPOSE PER FILE

Each file answers one question. No 500-line catch-all document.

AUTOMATIC COMPOSITION

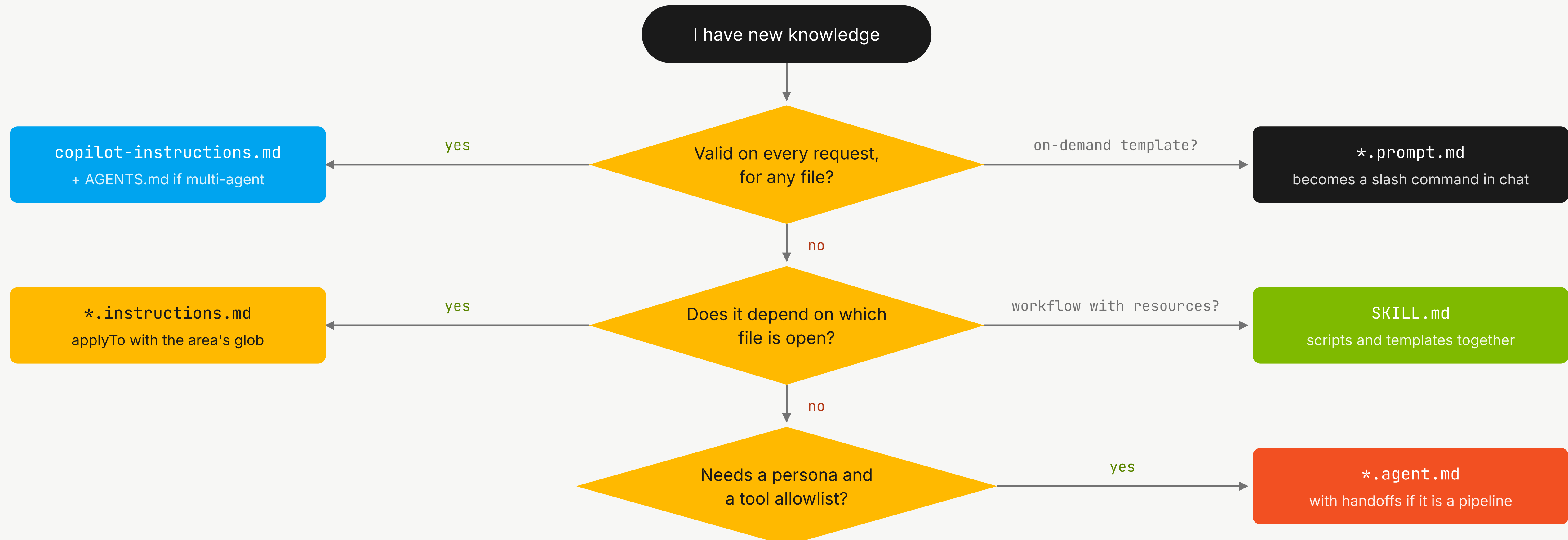
GitHub Copilot assembles the right context for each file you edit. No manual wiring.

VERSIONED WITH THE CODE

The whole stack lives in git. Config changes go through pull requests like any other code.

RESOLUTION · DECISION TREE

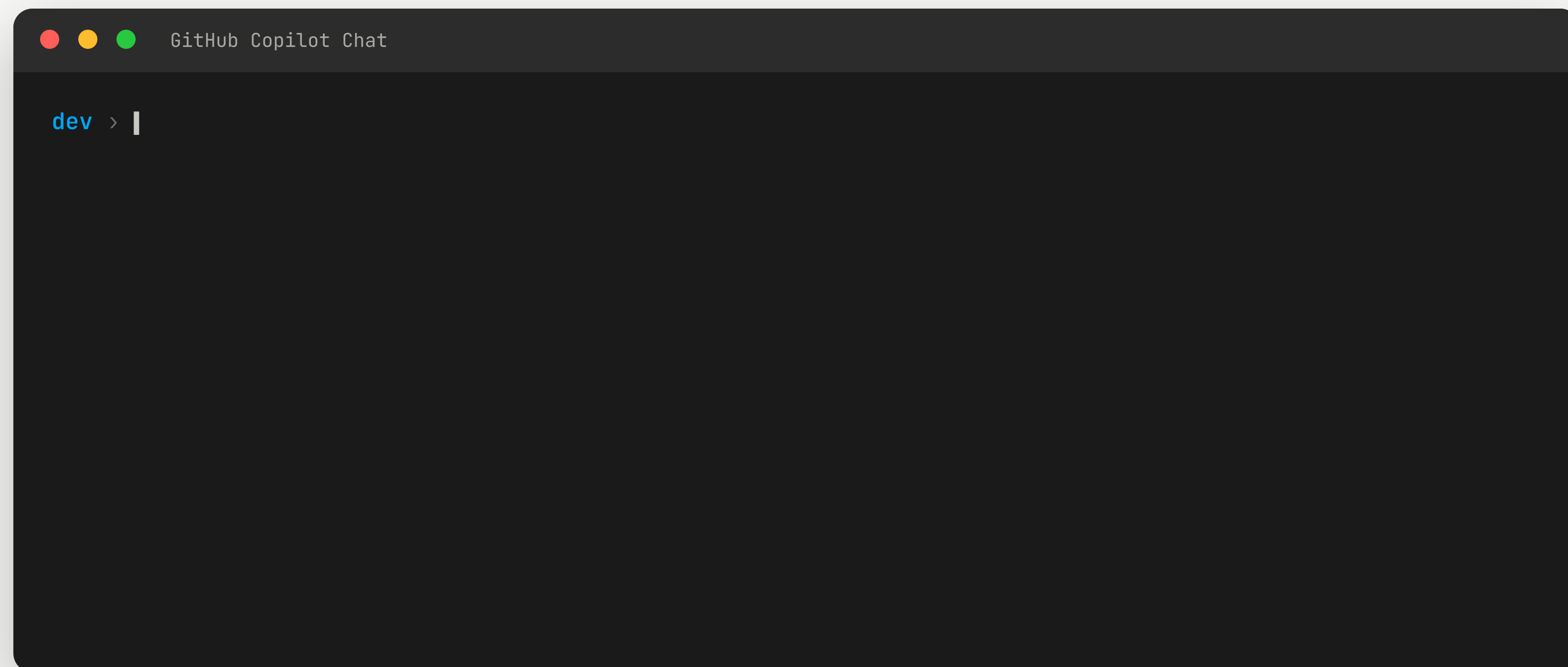
# Which file should you create? Four questions settle it.




*Rule of thumb: always start at the highest level that is still true. Knowledge valid everywhere goes up; knowledge valid for one case goes down. Duplication between levels is the smell that the wrong question was answered.*

THE HIERARCHY AT WORK

# One request, three layers applied, zero manual setup.




GLOBAL LAYER

The stack and conventions came from copilot-instructions.md without anyone pasting them.

SCOPED LAYER

The API rules activated because the target path matched the applyTo glob.

DOMAIN LAYER

The REST design skill brought workflows and checklists, not just rules.

PART

# IV

# In practice.

Three enterprise scenarios, the measured impact, and the do or do not list that keeps the stack healthy.

THREE TEAMS, THREE CONFIGURATIONS

# The same five layers solve three very different problems.

TECH LEAD · FINTECH STARTUP

### Ana standardizes 8 devs

Global instructions define the stack, three scoped instruction files cover components, API and tests, a security agent scans every PR, and a compliance skill encodes the financial domain.

**-40%**
manual code reviews

STAFF ENGINEER · E-COMMERCE ENTERPRISE

### Carlos breaks the monolith

One global architecture file, separate scoped instructions per service (cart, payments, inventory), a migration-expert agent for the legacy code, and shared skills for event sourcing and CQRS.

**18 → 10**
months of migration

PLATFORM ENGINEER · MULTINATIONAL

### Maria scales to 50+ repos

An organizational template every repo inherits, teams add their own scoped rules without conflicts, shared skills ship from a central repository, and standardized prompts cover common operations.

**2w → 2d**
new dev onboarding

*Figures observed in enterprise adoptions of this pattern. Treat them as direction, and validate against your own baseline.*

IN PRACTICE · ANTI-PATTERNS

# Four anti-patterns you will meet before any best practice.

ANTI-PATTERN 01

### The 3,000-line file

The team's entire knowledge dumped into copilot-instructions.md. It pays the cost on every request, buries the critical in the irrelevant, and nobody reviews it. Break it into layers: the decision tree slide is the map.

ANTI-PATTERN 02

### Generated and never curated

/init produces a good draft, but a generated draft documents the obvious and omits the tacit. Without an owner trimming the redundant every quarter, the file becomes noise paid per request.

ANTI-PATTERN 03

### The same rule in three layers

"Use TypeScript strict" in the global file, in the API instructions and in the skill. Since everything stacks as a union, the rule arrives three times, and when the team changes its mind someone forgets one copy. Each rule lives in exactly one level.

ANTI-PATTERN 04

### Secrets and environment in the wrong place

An API token in an instructions file, a production URL in a prompt file. These files go to Git and into the model's context. Secrets live in a vault; environment config lives in mcp.json with inputs, never hardcoded.

*All four come from real stack reviews. The common pattern: treating agent configuration as documentation, when it is code, with a per-execution cost, an owner and a review cycle.*

DO AND DO NOT

# Three habits separate healthy stacks from noisy ones.

DO NOT

Put everything in copilot-instructions.md. A 500-line file becomes noise and the model loses focus.

DO

Split into layers: global, scoped, agents, skills. Each file short, focused, and easy to review.

DO NOT

Write contradictory rules between layers, like global saying CSS Modules and a scoped file saying Tailwind.

DO

Make layers complement each other. If global defines Tailwind, the scoped file adds specific Tailwind rules.

DO NOT

Dump complex domain knowledge into flat instructions with no structure, no workflow, no checklist.

DO

Use SKILL.md for complex domains: rules plus workflows plus templates plus checklists. Structure ensures completeness.

WHY IT IS WORTH THE EFFORT

# Configuration automates what code review used to catch.

**-34%**

**Manual code reviews**

Config automates patterns that used to rely on human review.

**-50%**

**Onboarding time**

New devs produce consistent code from day one, guided by config.

**+29%**

**Code consistency**

Uniform patterns across every repo and team in the organization.

**-42%**

**Pattern bugs**

Automatic rules prevent common mistakes before the commit.

*Ranges observed across enterprise adoptions. Your baseline defines your numbers: measure before and after.*

PART

# The memory layer.

The files are what you write. Memory is what GitHub Copilot learns on its own, and recalls across sessions.

THE MEMORY LAYER · WHAT IT IS

# Stateless forces repetition. GitHub GitHub Copilot Memory remembers across sessions.

TYPE 01 · REPOSITORY FACTS

## What GitHub Copilot discovers from the code

Conventions, architecture decisions, build commands, cross-file dependencies. Created by those with write access, available to the whole team, tied to that repo.

TYPE 02 · USER PREFERENCES

## How you like to work

Communication style, preferred stack, commit conventions. They follow you across repositories, without affecting others. On Business and Enterprise, under admin governance.

Public preview, Jan 2026, on by default for Pro and Pro+ since March. Available in coding agent, code review and CLI. What one agent learns, another uses.

THE MEMORY LAYER · CROSS-AGENT

# What code review learns, the coding agent applies. Without you repeating it.

copilot-memory.log

```
[code-review] learned: ISafeAreaView and ISafeAreaView2 go together
↳ fact saved, cited at line 142, validated against the branch

[coding-agent] 3 days later, in another PR, updates both together
↳ without anyone re-explaining the rule

[copilot-cli] in the terminal, applies the team's commit convention

3 surfaces, 1 shared memory, validated before every use
```

VALIDATED BEFORE USE

Each fact carries citations. GitHub Copilot checks against the current branch before applying. Stale never enters.

EXPIRES IN 28 DAYS

An unused fact disappears on its own. The timer resets on each validated use. No accumulated junk.

SCOPED PER REPO

A fact from one repo never leaks into another. Privacy and security by design.

THE MEMORY LAYER · MEMORY OR FILE?

# Memory is emergent. A file is deliberate. You need both.

| | GITHUB GITHUB COPILOT MEMORY | HIERARCHY FILES |
|---|---|---|
| Origin | The agent discovers it on its own while working | You write it and version it in Git |
| Persistence | Expires in 28 days if unused | Permanent until someone edits it |
| Review | Does not go through a pull request | Reviewed in a PR, auditable in history |
| Best for | Patterns that emerge from real use | Rules you want to guarantee, always |

Rule of thumb: let memory discover, then graduate what repeats into a file. If GitHub Copilot relearns the same thing every week, that is a sign an instruction is missing.

THE MEMORY LAYER · GRADUATION

# The virtuous loop: memory discovers, the file perpetuates.

MEMORY LEARNS

The agent notices a pattern in real use and saves it as a fact.

REPEATS TOO MUCH

The same lesson reappears every week. A clear signal.

BECOMES A FILE

You promote it to copilot-instructions.md or ARCHITECTURE.md.

GRADUATE WHEN

"Always check ISafeAreaView and ISafeAreaView2 together" reappears every week, make it an instruction.

WHY IT MATTERS

Without graduation, valuable learning expires in 28 days. With graduation, the repo accumulates permanent knowledge.

Memory and file do not compete, they complete each other. Memory is the draft the agent writes on its own. The file is the version you decide to make law.

CLOSING

# Configuration is code for your agents.

*Building the future of software development with AI and Agentic DevOps.*

CONTACT

**Paula Silva**
Software Global Black Belt

paulasilva@microsoft.com

NEXT STEP

**Config stack review**
One repository, one week, five layers in place

Published 2026-06-11 · v6

TERMINAL · START HERE MONDAY

```
# 1. Create the structure
$ mkdir -p .github/instructions \
    .github/agents .github/skills .github/prompts

# 2. Start with the global brain
$ touch .github/copilot-instructions.md
  stack, conventions, key commands. Short.

# 3. Add one scoped rule where pain lives
$ touch .github/instructions/api.instructions.md
  applyTo glob + the rules for that area only
```