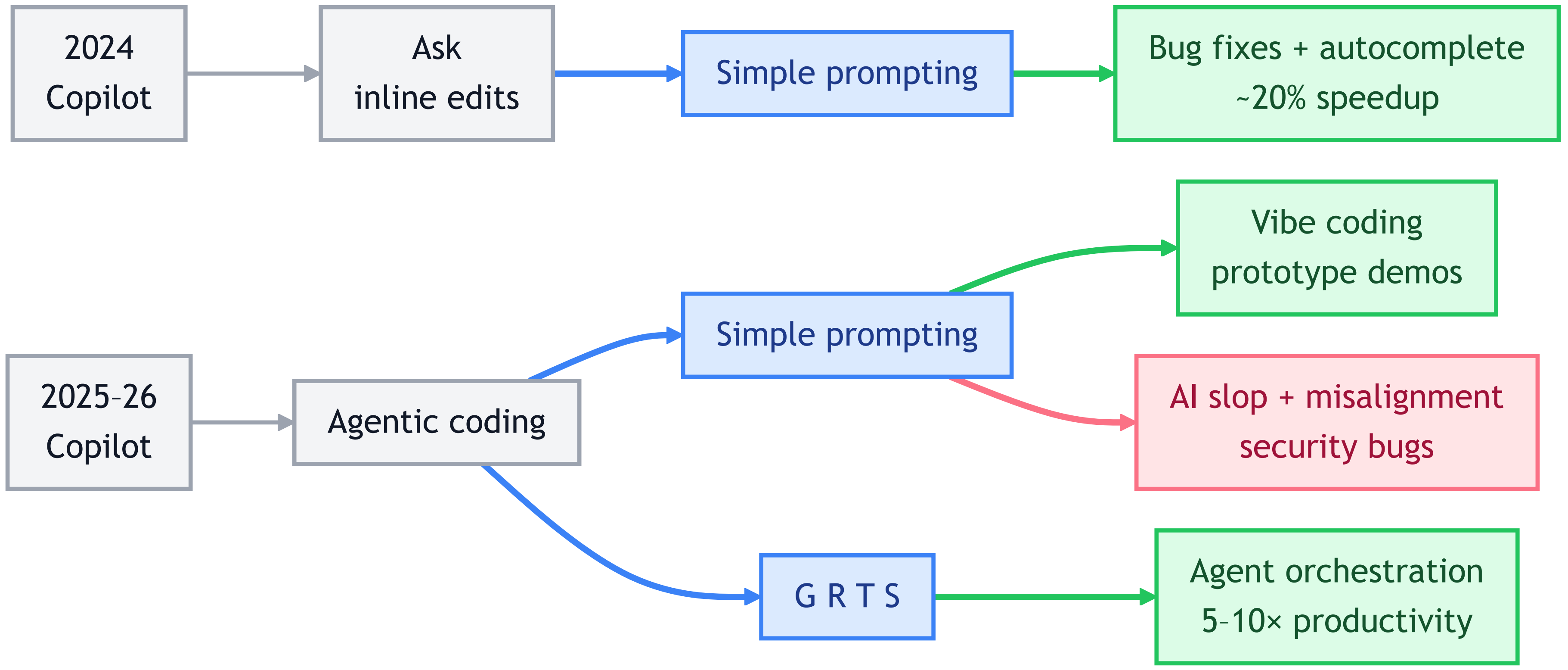
Guidance 🧭 | *Reality Rules Roles* 🧱 | *Tools* 🧰 | *Specifications* ✅

GRTS

Don't build features. **Build the system that builds features.**

Coding Agent: The very eager stupid genius



You will not be replaced by an agent...

but you may be replaced by someone who can direct one well.

🧭 Guidance Prompting | Supervision | Hand-holding

GRTS

- Get familiar with the eccentricities of the stupid genius
  - **First steps (concrete, low context, low risk tasks):** Ask it to (1) review your code, (2) debug your software, (3) refactor a class and (4) research the next idea.
- If the agent falls down a Rabbit hole 🐰🕳 -> ask:
  - Research the best 3 approaches to this issue with pros/cons — cite evidence.
  - Check out this site <stack overflow link>. Does it help?
- If the agent tries to cheat or take a shortcut -> ask:
  - What is the proper way to fix this?
  - Right place for the code? Clean architecture? Utilities library?
- The agent keeps giving me the "average" solution - not the one for my domain!
  - Make the domain explicit: "I am working in this domain and follow such and such guidlines."
  - If possible, use an oracle / golden sample / explicit success criteria.

Takeaway: Spend 1-2 weeks getting a feel for working with the latest LLM's.

Reality | Rules | Roles Instructions | Skills | Orientation | Foundation

- **.github/copilot-instructions.md** -> Repository orientation
  - Agents will always read this file. It should contain the fundamentals of the repo, the reality of what it is for, what base technologies are used, how to build and test and how it is structured.
  - Point to any coding standards or guidelines - make them or move them into the repo if needed.
- **Skills and custom agents** -> wearing different hats
  - Have a repeatable way to focus the agent in on a task or technology.
  - Find the best ones and iterate. Is the agent getting better at understanding your context>?
- **Keep it concise | Structure matters | Be direct | Show examples**
- **Use this blog post for instructions and this repo for a curated list of skills and custom agents**

Takeaway: After bootstrapping the initial set of files, refine them heavily in the first week, putting the best guidance in the relevant places.

Tools Capabilities | Abilities | Touch the external world

To get this	use this
Issue read and write	GitHub MCP, Atlassian MCP
Up-do-date API	Context7 + research online
Agent intellisense	Serena
Deterministic actions	Generate a script for repetitive/complex work (e.g., convert NUnit3 tasks to NUnit4 across many files)
Truth sources	diffs, test output, CI checks, benchmarks/sim logs
Easy local action	Make VS Code tasks for Build, test, lint, format, etc.
Stop the approvals!	**chat.tools.terminal.autoApprove** for git status, list files, get file content

Takeaway: 1 week - Craft a thoughtful set of MCP tools, VSCode tasks, restrictions and auto-approvals (and update custom agent instructions) to make your "guided, rule-constrained stupid genius" powerful.

✅ Specifications Objectives | Checklists | Planning | Requirements

GRTS

- You mostly review outputs and spot-check code/decisions.
- For larger, more ambiguous work, spend time planning to reduce churn and wrong turns.

How much ambiguity?	Recommended workflow
Level 1 (Low)	For bugs and scoped changes, we can just make a single prompt
Level 2 (Low → Medium)	1–3 planning prompts (research/explain/3 options) → then implement
Level 3 (Medium → High)	Create a checklist in Markdown → review → implement

A1: Multi-agent workflows

- You’ll often be waiting on agents. Use that time:
  - Research the next feature in parallel
  - Use git worktrees / background agents for parallel branches
  - Delegate to cloud agents (ensure CI/CD is set up)
  - Split roles: Planner → Implementer → Reviewer

A2: Risk -> Gating | Ambiguity -> Planning

The good	The bad and the ugly
reason about the goal plan steps take actions (read/edit/run tools) iterate until done (or stuck)	confidently wrong misses hidden constraints misapplies “best practices” thrashes without a stable hypothesis misuses tools (wrong env/partial runs)

	Risk: Low	Risk: High
Ambiguity: Low	Great for agents (docs, refactors, small tests)	Needs gates + review (small but critical changes)
Ambiguity: High	Clarify first (spec + oracle)	Human-first (architecture/safety-critical/unclear bugs)

A3: References (Core)

- G — Building effective agents (Anthropic) — Designing agent loops: checkpoints, tool feedback, stop conditions.
- R — Security (VS Code Copilot) — Trust boundaries, tool approvals, prompt injection risks.
- R — Workspace Trust (VS Code) — Restricted Mode and why it matters for agents.
- R — LLM01:2025 Prompt Injection — Threat model + mitigations.
- R — About GitHub Copilot coding agent — Capabilities, limits, and governance.
- T — Tutorial: Work with agents in VS Code — Local/plan/background/cloud agent workflows + worktrees.
- T — Use tools in chat (VS Code) — Tool approval — Tool approvals, URL post-approval, and auto-approval tradeoffs.
- S — Spec Kit — Spec → Plan → Tasks to make “done” measurable.
- S — CI (GitHub Actions) — CI as a repeatable verification oracle.
- S — A Minimal, Reproducible Example — Make bugs/tasks reproducible.
- S — Responsible use of GitHub Copilot coding agent — Scope, acceptance criteria, review gates.

A4: References (More)

- G — Lessons from Anthropic (secondary write-up)
- T — GitHub Copilot in VS Code
- T — Get started with GitHub Copilot in VS Code
- T — Asking GitHub Copilot questions in your IDE
- T — Review AI-generated code edits (VS Code)
- R — Adding repository custom instructions for GitHub Copilot
- T — Context7 (GitHub)
- T — Serena docs
- T — git-worktree documentation
- S — About issue and pull request templates (GitHub)
- S — Configuring issue templates for your repository (GitHub)
- T — GitHub Copilot Workspace (GitHub Blog)
- T — What is Foundry Agent Service?