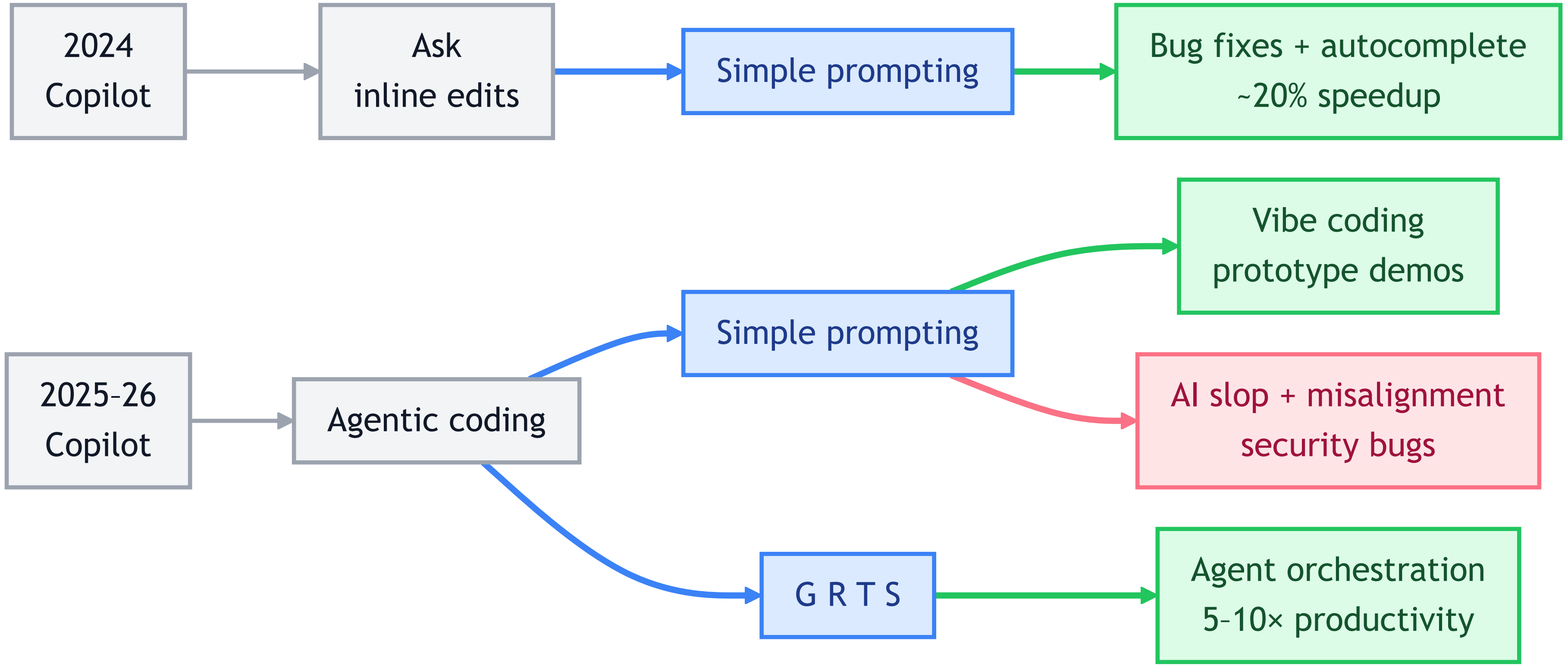
Guidance 🧭 | *Reality Rules Roles* 🧱 | *Tools* 🧰 | *Specifications* ✅

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 review your code, debug your software and 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?
- Always define "good":
  - **Verification vs Validation** — built the thing right vs build the right thing ← we need this!
  - 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

- TODO - keep reviewing from here forward!
- As of Dec 2025, the new models are able to hold 8 things in their mind at the same time - they are limited by the context you give them, not their intelligence.
- Build repo overview + working rules (.github/copilot-instructions.md) ← **Copilot can help with this!**
- Cite docs: build scripts, library locations, standards, linting guides
- For the top 3–8 areas, define "roles" with repo-specific best practices (custom agents + optional .github/instructions/*.instructions.md)
- Define boundaries: excluded (secrets), auto-approved (git fetch/status), and restricted files (CI/CD, security)

Tips:

- Keep instruction files short (100–300 lines). Refine or split as needed.
- Spend 1–2 days iterating; this pays off.
- When prompts miss something (edge cases, wrong file, global var), explain the failure and update the

Tools Capabilities | Abilities | Touch the external world

Without extra tools, agents are limited to reading and writing files in your repo.

- Issue read and write: GitHub MCP Atlassian MCP
- Accurate API and function calls: Context7 + research online
- Semantic repo navigation on large codebases: 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
- VS Code tasks: Build, test, lint, format, etc.
- Shell commands: git status, list files, get file content
- Tip: Use settings.json ->"chat.tools.terminal.autoApprove" for read-only commands

✅ Specifications Objectives | Checklists | Planning | Requirements

- You mostly review outputs and spot-check code/decisions.
- For larger work, spend time planning to reduce churn and wrong turns.
- **Level 1**: For bugs and scoped changes, we can just make a single prompt
- **Level 2**: 1–3 planning prompts (research/explain/3 options) → then implement
- **Level 3**: Create a checklist in Markdown → review → implement
- 1-page templates help a lot: Task spec + rules of engagement
- **Level 4**: Use OpenSpec or Spec Kit: specify → clarify → plan → analyze → 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?