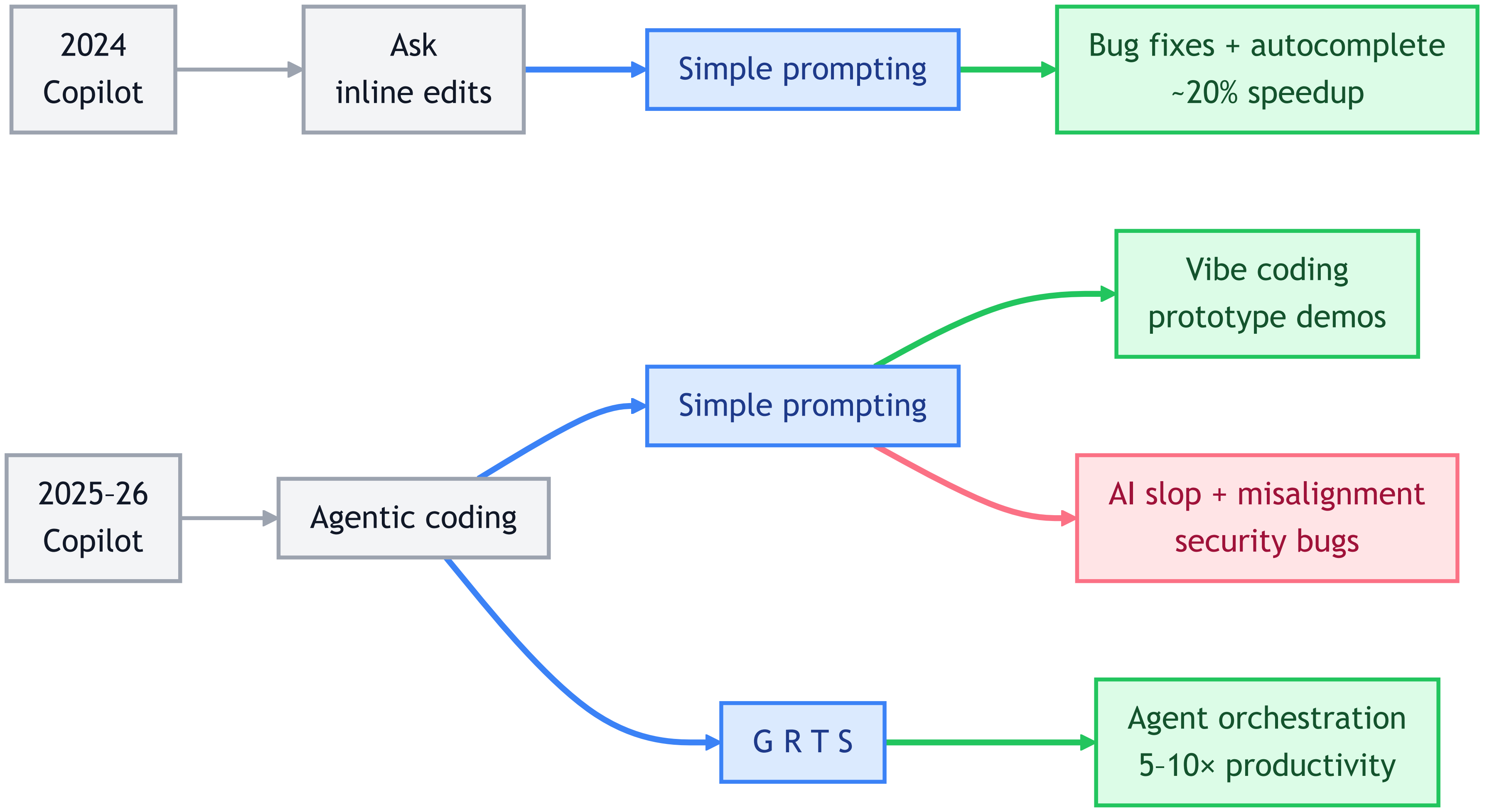
Guidance 🧭 | *Reality Rules Roles* 🧱 | *Tools* 🧰 | *Specifications* ✅

GRTS

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

Coding Agent 🍺🤖 The very eager stupid genius



Guidance Prompting | Supervision | Hand-holding

GRTS

- Rabbit hole:
  - Is this the best approach?
  - Research best practices (≥5 sources).
  - Give the best 3 approaches with pros/cons — cite evidence.
- Shortcut:
  - What are the best practices here?
  - **NEVER IGNORE TESTS OR COMMENT THEM OUT**
  - Right place for the code? Clean architecture? Utilities library?
- Define “good”:
  - **Verification vs Validation** — built the thing right vs build the right thing ← we need this!
  - Use an oracle / golden sample / explicit success criteria.

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

GRTS

- Build repo overview + working rules (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 (xyz.instructions.md / Claude skills)
- 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 right instruction file.

🧰 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
- Deterministic actions: generate a script for repetitive/complex work (e.g., convert NUnit3 tasks to NUnit4 across many files)
- 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
- **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 (VS Code + Agent HQ) for parallel branches
  - Delegate to cloud agents (ensure CI/CD is set up)

Reference: Agent HQ — Background on VS Code's emerging multi-agent orchestration model and how it changes day-to-day workflow. Useful context for why parallel work (e.g., via git worktrees) matters.

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: Main References

- **G** — Lessons from Anthropic — A practical set of prompting lessons for giving better guidance.
- **G** — Lessons from Anthropic — Prompting lessons for better guidance.
- **G** — Building effective agents (Anthropic) — Designing agent loops: checkpoints, tool feedback, stop conditions.
- **R** — Security (VS Code Copilot) — Risk areas and guardrails.
- **R** — Workspace Trust (VS Code) — Trust boundaries + Restricted Mode.
- **R** — LLM01:2025 Prompt Injection — Prompt injection risks + mitigations.
- **T** — Tutorial: Work with agents in VS Code — Running agent workflows in the IDE.
- **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: Auxiliary References

- GitHub Copilot Workspace: Welcome to the Copilot-native developer environment
- GitHub Copilot in VS Code
- Asking GitHub Copilot questions in your IDE
- Get started with GitHub Copilot in VS Code
- Use tools in chat (VS Code) — Tool approval
- Review AI-generated code edits (VS Code)
- Adding repository custom instructions for GitHub Copilot
- About GitHub Copilot coding agent
- What is Foundry Agent Service?
- Context7 (GitHub)
- Serena docs
- git-worktree documentation
- About issue and pull request templates (GitHub)
- Configuring issue templates for your repository (GitHub)