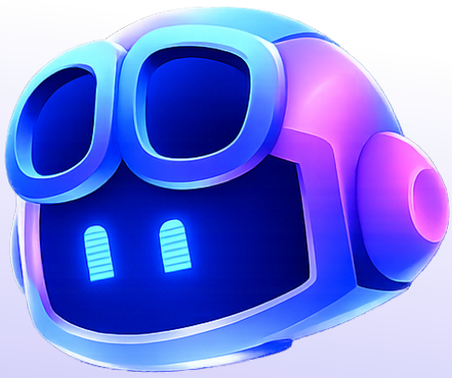
INTERCEPT



GitHub Copilot Checklist

Best Practices

1. Best Practices for Teams

- [ ] **Give Copilot deliberate context:** Open or attach the files that matter, close irrelevant context, and name the intended behaviour before asking for code.
- [ ] **Break work into steps:** Ask for a small function, test, refactor or migration step instead of generating an entire feature at once.
- [ ] **Set clear requirements:** Define the goal, constraints, expected behaviour and relevant examples so Copilot doesn't have to make unneeded assumptions.
- [ ] **Plan complex work first:** For large changes, ask Copilot to propose an approach first. For new solutions, prototype options before committing.
- [ ] **Give Copilot deliberate context:** Clear function names, variables, comments and docstrings help Copilot understand intent and generate relevant suggestions.
- [ ] **Treat output as a draft:** Read, test and refine every Copilot suggestion before accepting it, even when the generated code appears to work correctly.
- [ ] **Use Copilot for review:** Ask Copilot to identify risks, edge cases and missing tests, while keeping human ownership of the final change.
- [ ] **Use tests to steer Copilot:** Define the expected behaviour with tests before accepting an implementation, especially when fixing bugs or changing existing functionality.
- [ ] **Choose the right Copilot mode:** Use completions for coding, Chat for reasoning and agents for multi-step work that requires tools or file changes.
- [ ] **Create repository instructions:** Use `.github/copilot-instructions.md` and `.md` files to align Copilot with coding standards, security rules and organisational practices.
- [ ] **Add path-specific instructions:** Provide targeted guidance for areas such as authentication, billing, infrastructure and data access when their requirements differ.
- [ ] **Keep instructions current:** Include only the rules Copilot needs repeatedly and update them as architecture, codebase and development practices change.
- [ ] **Create reusable workflows:** Capture recurring tasks in prompt files, skills or specialised agents so developers do not repeatedly recreate the same approach.
- [ ] **Scope agent tasks clearly:** Define the exact task, relevant files, expected output, exclusions and approval points before allowing an agent to begin.
- [ ] **Set boundaries for agents:** Specify which agents are approved, where they may run, which tools they can use and which actions require approval.
- [ ] **Measure usage and outcomes:** Compare Copilot usage and AI Credit spend with delivery speed, defects, reviewer load and other meaningful engineering outcomes.

INTERCEPT

GitHub Copilot Checklist

Best Practices

2. Cost Optimisation

- [ ] **Set spending guardrails:** Set budgets and limits early to prevent users or experimental workflows from consuming excessive capacity.
- [ ] **Create tailored budgets:** Differentiate budgets for standard developers, power users, pilot teams, agentic workflows and experiments as needed.
- [ ] **Match models to tasks:** Use stronger models for complex reasoning or security work and lighter models for generation, summaries or formatting.
- [ ] **Measure value per AI Credit:** Compare value produced with AI Credit usage instead of treating higher consumption as the objective.
- [ ] **Use inline completions:** Use inline completions when developers know what to build, and Chat when the task requires reasoning.
- [ ] **Use Auto model selection:** Set Auto as the default for GitHub Copilot and Copilot CLI to reduce unnecessary premium-model usage and benefit from 10% discount.
- [ ] **Limit Agent Mode:** Use Agent Mode only when autonomous editing, tool execution or multi-step work is genuinely required.
- [ ] **Specialise tool access:** Match tool access to each workflow and give agents only the tools necessary for the task.
- [ ] **Avoid repository scans:** Tell Copilot which files, folders or tests matter instead of scanning the entire repository unnecessarily.
- [ ] **Keep tool output focused:** Run the narrowest relevant test and summarise failures instead of adding pages of raw terminal output.
- [ ] **Narrow the context window:** Mention exact files, errors or failing tests and exclude generated files, vendor folders and unrelated artefacts.
- [ ] **Start fresh sessions:** Start a fresh session when the task changes and carry forward only a concise summary of conclusions.
- [ ] **Review plans before execution:** Review Copilot's plan before sequences of edits, tests and tool calls, then narrow the scope if needed.
- [ ] **Keep instructions concise:** Keep persistent guidance focused on rules Copilot needs repeatedly, because long instructions create hidden context overhead.
- [ ] **Automate repeatable work:** Use Copilot to create a script, GitHub Action or workflow once for deterministic, repeatable work.
- [ ] **Review usage weekly:** Review usage by user, model and workload weekly, including premium requests, AI Credit spend and tool usage.

INTERCEPT

GitHub Copilot Checklist

Best Practices

3. Security Best Practices

- [ ] **Security tooling in the workflow:** Apply static analysis, code scanning, secret scanning, dependency checks, linting and tests to all AI-generated code.
- [ ] **Never include real secrets:** Use placeholders and approved secret stores instead of placing API keys, credentials or passwords in prompts.
- [ ] **Protect sensitive data:** Only include customer PII, proprietary algorithms or credentials in Copilot context when organisational policy explicitly allows it.
- [ ] **Configure content exclusions:** Exclude sensitive repositories, files or paths from Copilot context where required by security, privacy or IP policies.
- [ ] **Define your public-code policy:** Decide how to handle suggestions matching public code and configure Copilot policies for licensing or IP requirements.
- [ ] **Validate against controls:** Validate code against internal security policies, regulatory obligations, architecture standards and organisational controls.
- [ ] **Use established cryptographic libraries:** Use maintained libraries and approved patterns instead of asking Copilot to invent cryptographic implementations from scratch.
- [ ] **Keep human in the loop:** Require human approval before authentication, IAM, billing, migrations, dependency upgrades, destructive commands or customer-data actions.
- [ ] **Review security-critical code:** Never auto-accept generated code for authentication, cryptography, input validation, database access or other sensitive areas.
- [ ] **Strengthen generated tests:** Check generated tests validate the actual requirement and include negative, regression and relevant edge cases.
- [ ] **Check architectural fit:** Verify changes respect existing repository boundaries and architecture conventions instead of creating locally sensible shortcuts.
- [ ] **Govern agents separately:** Set separate policies for IDE Agent Mode, Copilot CLI, cloud agents, third-party agents and AI-assisted review.
- [ ] **Restrict agent permissions:** Match file editing, shell, browser, MCP server, database and deployment permissions to each agent's specific job..
- [ ] **Keep agent activity auditable:** Retain prompts, agent summaries, tool activity, review notes and final diffs so autonomous work remains auditable.
- [ ] **Check packages and APIs:** Verify suggested packages, APIs and implementation patterns against current documentation before accepting them into the codebase.
- [ ] **Scope autonomous work:** Define the problem, relevant files, acceptance criteria, out-of-scope boundaries and validation requirements for every agent task.

INTERCEPT

GitHub Copilot Checklist

Best Practices

4. Improving Organisational Management

- [ ] **Baseline the current workflow:** Track cycle time, reviewer load, defects, test gaps, deployment friction and developer sentiment to measure Copilot's impact.
- [ ] **Set clear goals and metrics:** Define success and establish KPIs that can be compared with the pre-Copilot baseline throughout the rollout.
- [ ] **Scale what delivers value:** Expand Copilot use where evidence shows better productivity, quality or delivery without unnecessary cost or risk.
- [ ] **Run knowledge-sharing sessions:** Let teams share useful prompts, workflows, mistakes and new Copilot capabilities so skills improve across the organisation.
- [ ] **Provide role-based training:** Train developers on prompting, IDE integration, pull requests, relevant context, Copilot's limitations and review requirements.
- [ ] **Build a central knowledge base:** Maintain prompt templates, successful examples, FAQs and troubleshooting guidance so knowledge is shared.
- [ ] **Standardise Copilot usage:** Define shared expectations for prompting, model choice, review, agent use and when human approval is required.
- [ ] **Experiment with AI development loops:** Explore workflows where AI plans, implements, tests and reviews against a defined goal, with clear human checkpoints and boundaries.
- [ ] **Add repository instructions:** Standardise architecture, coding, testing and security guidance so teams do not rely on individual prompting habits.
- [ ] **Build shared context:** Curate architecture docs, ADRs, runbooks, glossary terms and useful examples in Copilot Spaces for repeated use.
- [ ] **Set agent boundaries:** Define where agents may run, which tools they can call and which actions require approval.
- [ ] **Update code review:** Add checks for edge cases, generated tests, security scanning, dependency changes and code ownership to pull requests.
- [ ] **Enforce consistent policies:** Align content exclusions, usage metrics, model guidance and escalation paths across teams.
- [ ] **Connect usage to outcomes:** Pair Copilot activity with PR cycle time, defects, deployment signals, reviewer load, spend and developer satisfaction.
- [ ] **Scale agentic workflows carefully:** Expand agentic use only where teams understand the required governance, review expectations and controls.
- [ ] **Introduce agents progressively:** Introduce agentic workflows only after teams understand basic Copilot usage, review expectations, governance and required controls.

INTERCEPT

GitHub Copilot

CIE session

Learn how to improve productivity, code quality and collaboration across your development team.

Want to join? A limited number of places are available.

Reserve your place →