Strecht Simulados · Guia de preparação



Guia de preparação GH-300

Tudo o que a prova cobra, organizado por domínio: conceitos-chave, comparações que confundem, pegadinhas reais, gatilhos do enunciado, questões comentadas e um plano para chegar pronto ao exame.

6
capítulos por domínio

36
questões comentadas

70%
nota de corte

6
provas completas no curso

ANTES DE COMEÇAR

Como usar este guia

Este guia foi feito para ser estudado junto com as 6 provas completas do curso GH-300: o guia explica o que cai e onde estão as armadilhas; as provas treinam o ritmo e mostram o que você ainda não domina.

Quando	O que fazer
Semana 1	Faça o teste diagnóstico (página seguinte) e leia os capítulos 1 e 2. No fim da semana, faça a Prova 1 do curso no tempo do exame.
Semana 2	Leia os demais capítulos, começando pelos domínios em que você errou mais no diagnóstico. Faça as provas 2 e 3.
Semana 3	Releia os alertas de pegadinha e as tabelas de comparação. Faça as provas 4 e 5 e revise cada erro pela explicação.
Semana 4	Estude a folha de cola, faça a Prova 6 e refaça as provas abaixo de 80%. Agende o exame quando estiver acima de 80% em todas (a nota de corte é 70%).

Mapa do exame

Cada prova do curso tem 50 questões e 100 minutos, no formato do exame. Referência de aprovação: 70%.

Domínio (capítulo)	Peso	Neste guia
1. Aplicar engenharia de prompt e criação de contexto	10-15%	17 conceitos · 14 pegadinhas
2. Configurar privacidade, exclusões de conteúdo e proteções	10-15%	18 conceitos · 14 pegadinhas
3. Entender a arquitetura e os dados do GitHub Copilot	10-15%	18 conceitos · 12 pegadinhas
4. Melhorar a produtividade do desenvolvedor com o GitHub Copilot	10-15%	17 conceitos · 12 pegadinhas
5. Usar o GitHub Copilot com responsabilidade	15-20%	18 conceitos · 14 pegadinhas
6. Usar recursos do GitHub Copilot	25-30%	18 conceitos · 14 pegadinhas

Como ler os capítulos: os **tópicos-chave** são o que mais cai; a **tabela de memorização** separa o que a prova costuma confundir; os **alertas de prova** mostram as armadilhas das alternativas erradas; os **gatilhos** ligam palavras do enunciado à resposta; e o **checklist** diz se você está pronto para o domínio.

Estude um pouco todo dia, de graça

Instagram @strecht.simulados

Um post de prova por dia: questão comentada, pegadinhas e o “não confunda” de cada certificação.

Seguir no Instagram →

YouTube Strecht Simulados

Shorts diários com pegadinhas de prova e, aos sábados, um simulado comentado.

Inscrever-se no canal →

DIAGNÓSTICO

Você já sabe isso?

Responda às 12 questões sem consultar nada. As respostas, com a explicação e o capítulo para revisar, estão no fim do guia. Comece o estudo pelos capítulos em que errou.

D1 · Aplicar engenharia de prompt e criação de contexto

Para a documentação de engenharia de prompt do GitHub, o que o Copilot usa, além do próprio texto do prompt, para gerar uma resposta?

A. O contexto, como o arquivo atual e o histórico
B. Os e-mails e o calendário do desenvolvedor na empresa
C. O histórico de commits dos repositórios públicos do GitHub
D. Apenas as palavras digitadas, sem nenhum outro contexto

D2 · Aplicar engenharia de prompt e criação de contexto

Em uma conversa do Copilot Chat, uma desenvolvedora da Rede Bistrô percebe que um pedido antigo já não serve como contexto e está atrapalhando as respostas novas. O que as boas práticas do GitHub sugerem?

A. Excluir esse pedido da conversa atual
B. Pedir ao Copilot que esqueça a conta da usuária
C. Reenviar o mesmo pedido com mais pontos de exclamação
D. Trocar de IDE para limpar a memória do Copilot

D3 · Aplicar engenharia de prompt e criação de contexto

Um desenvolvedor da Importadora Leme escreve prompts longos para o Copilot Chat. Segundo a documentação do GitHub, em que posição do prompt ele deve colocar os requisitos mais importantes?

A. Em um comentário separado, enviado após a resposta
B. No meio do texto, onde o modelo presta mais atenção
C. Repetidos em todas as frases para reforçar a ideia
D. No início, que recebe peso extra, assim como o final

D4 · Configurar privacidade, exclusões de conteúdo e proteções

No VS Code, o Copilot de um desenvolvedor da Química Nova não consegue alcançar os serviços por causa do proxy e do firewall corporativos. Qual comando abre um diagnóstico de conectividade com a seção Reachability?

A. Developer: Set Log Level
B. Preferences: Open Keyboard Shortcuts
C. Developer: Reload Window
D. Developer: Chat Diagnostics

D5 · Configurar privacidade, exclusões de conteúdo e proteções

Depois de configurar a exclusão de conteúdo, o admin da Indústria Prisma quer confirmar que o Copilot Chat no IDE não usa um arquivo excluído. Qual teste a documentação recomenda?

A. Criar um arquivo .copilotignore na raiz e conferir o histórico de commits
B. Pedir ao Copilot CLI que liste as regras de exclusão ativas
C. Rodar o comando /context e procurar o arquivo excluído na lista de tokens
D. Abrir só o arquivo excluído, anexá-lo ao prompt e pedir explain this file

D6 · Configurar privacidade, exclusões de conteúdo e proteções

Um desenvolvedor da Pesquisa Aurora recebe a mensagem de que atingiu um limite de taxa (rate limit) do Copilot. Segundo a documentação, qual deve ser a primeira ação?

A. Trocar a conta do GitHub por outra conta da empresa
B. Reinstalar a extensão e apagar as configurações locais do editor
C. Desativar a exclusão de conteúdo do projeto atual
D. Aguardar e tentar de novo, já que o limite é temporário

D7 · Entender a arquitetura e os dados do GitHub Copilot

O application card do Copilot Chat descreve, em etapas, como o Chat transforma a pergunta do usuário em resposta. Qual é a ordem dessas etapas?

A. Entrada processada, análise do modelo, geração, formatação
B. Formatação, análise do modelo, entrada processada, geração
C. Geração, entrada processada, formatação e análise do modelo
D. Análise do modelo, geração, formatação e entrada processada

D8 · Entender a arquitetura e os dados do GitHub Copilot

Segundo o Microsoft Learn, qual sequência descreve corretamente o ciclo de vida de uma sugestão de código no Copilot, do prompt até a entrega ao usuário?

A. Contexto e prompt, proxy, filtro de toxicidade, LLM, pós-processamento, entrega
B. Contexto e prompt, LLM, proxy, entrega, filtro de toxicidade e pós-processamento
C. LLM, contexto, pós-processamento, proxy, filtro de toxicidade e entrega final
D. Proxy, contexto e prompt, LLM, filtro de toxicidade, entrega, pós-processamento

D9 · Entender a arquitetura e os dados do GitHub Copilot

O gerente de engenharia da Metalúrgica Forte, que contratou o Copilot Business, pergunta se o código e os prompts dos desenvolvedores serão usados para treinar modelos de IA. O que a documentação do GitHub afirma?

A. Não; dados do Business e do Enterprise não treinam modelos, conforme o Data Protection Agreement
B. Não, porque o Copilot processa tudo no computador e não envia código a servidores
C. Sim, mas só os trechos de código aceitos como sugestão inline entram no treinamento
D. Sim, a menos que cada desenvolvedor desligue a opção de treinamento nas Copilot settings

D10 · Melhorar a produtividade do desenvolvedor com o GitHub Copilot

Uma desenvolvedora da Bilheteria Palco quer que o Copilot monte a infraestrutura de testes de um serviço que depende de uma API externa de pagamentos. Qual pedido está alinhado aos usos descritos no Microsoft Learn?

A. Copiar dados do ambiente de produção para usar nos testes
B. Chamar a API real de pagamentos em todos os testes do serviço
C. Gerar mocks da API, fixtures e funções auxiliares de teste
D. Pular os testes que envolvem a integração com a API externa

D11 · Melhorar a produtividade do desenvolvedor com o GitHub Copilot

Os desenvolvedores da Fintech Lago perdem foco alternando entre o editor, o navegador e a documentação de APIs. Segundo o Microsoft Learn, como o Copilot ajuda a reduzir essa troca de contexto?

A. Enviando um resumo diário das APIs por e-mail para a equipe
B. Abrindo a documentação oficial em uma nova janela do navegador
C. Bloqueando o navegador enquanto o desenvolvedor está programando
D. Com ajuda dentro do editor, sugerindo chamadas e parâmetros

D12 · Melhorar a produtividade do desenvolvedor com o GitHub Copilot

Um desenvolvedor da Papelaria Nobre usa o GitHub Desktop e pede ao Copilot sugestões de mensagem de commit. Segundo o application card, como ele melhora a precisão da mensagem gerada?

A. Selecionando as linhas ou os arquivos específicos antes
B. Escrevendo a mensagem em inglês e pedindo tradução ao Copilot
C. Gerando a mensagem antes de fazer qualquer mudança no código
D. Deixando o Copilot confirmar o commit sem a revisão do autor

01

CAPÍTULO 1 · 10-15% DA PROVA

Aplicar engenharia de prompt e criação de contexto

Este domínio (10 a 15%) cobra como escrever prompts e dar contexto ao Copilot em cada superfície. As perguntas mais repetidas são sobre zero-shot, one-shot e few-shot, os 4 S (Single, Specific, Short, Surround), role prompting e as estratégias da documentação do GitHub: objetivo geral antes dos detalhes, evitar ambiguidade, dar exemplos, quebrar tarefas, iterar e manter o histórico relevante. Preciso saber como anexar contexto (abas abertas, #codebase, @ e # no Chat, @caminho e #número na CLI, critérios de aceite no cloud agent) e como lidar com janela de contexto cheia. Onde eu mais erro é diferenciar os 4 S entre si e escolher entre nova conversa, excluir um pedido ou resumir o contexto anterior.

Neste capítulo
17 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Zero-shot, one-shot e few-shot

Zero-shot não traz exemplos e o modelo usa só o treinamento (exemplo do Learn: comentário pedindo conversão de Celsius para Fahrenheit). One-shot traz um único exemplo, útil para manter implementações consistentes no padrão do projeto. Few-shot traz vários exemplos e equilibra a imprevisibilidade do zero-shot com a precisão de um ajuste fino.

Role prompting

Pedir que o Copilot atue como especialista de um domínio (segurança, desempenho, testes, dev sênior de C++) para incorporar essa expertise e acertar mais na primeira tentativa. Exemplo cobrado: validação de senhas considerando OWASP pede especialista em segurança.

Os 4 S

Single: uma tarefa por prompt. Specific: instruções explícitas e detalhadas (regras, formato de retorno). Short: conciso, sem repetição nem explicação inútil. Surround: contexto por nomes de arquivo descritivos e arquivos relacionados abertos.

Estrutura do prompt

Comece pelo objetivo geral e depois liste os requisitos específicos (exemplo oficial: função JavaScript que diz se um número é primo, recebe inteiro, retorna true, dá erro se não for inteiro positivo). Instruções no início e no fim do prompt recebem mais peso, então o requisito crítico vai no começo.

Evitar ambiguidade

Em vez de 'o que isto faz?', nomeie o alvo: a função createUser, um bloco, ou 'o código da sua última resposta'. Escreva requisitos como regra direta ('retorne o resultado como JSON'), não como 'lembre-se de'. Para bibliotecas, declare os imports no topo do arquivo ou diga qual usar.

Exemplos e testes como especificação

Dê exemplos dos formatos de entrada e da saída esperada. Testes de unidade também são exemplos: peça primeiro os testes e depois a função que os atenda.

Quebrar tarefas complexas

Divida um pedido grande em passos simples (exemplo oficial do caça-palavras: grade de letras, busca de palavras e depois a montagem), reaproveitando as funções anteriores. É você quem quebra a tarefa, não o Copilot.

Experimentar e iterar (assert and iterate)

Trate a primeira saída como começo de diálogo: na sugestão inline, apague e recomece ou mantenha e peça ajustes; no comentário, enriqueça com detalhes e exemplos e repita; no Chat, refira-se à resposta anterior ou apague e tente de novo. Resposta ruim pede reformular ou dividir o prompt, e dá para dar feedback com polegar.

Histórico do chat

A pergunta de acompanhamento ('e como testar isso?') funciona porque o histórico da sessão é contexto. Assunto novo pede conversa nova; pedido antigo que atrapalha deve ser excluído da conversa; em conversa longa, resuma o contexto essencial no novo pedido em vez de arrastar o histórico ou repetir código inteiro.

Voltar turnos por superfície

No Copilot CLI dá para voltar a um prompt anterior com /undo ou /rewind. No Copilot cloud agent não dá para voltar turnos; Archive session só tira a sessão da lista e Stop session encerra.

Contexto por superfície

Sugestões inline: abra os arquivos relevantes e feche os irrelevantes. Chat no IDE ou no GitHub.com: selecione o código e anexe com @, # ou slash commands. CLI: @caminho para arquivo e #número para issue. Cloud agent: repositório, branch base, objetivo claro, critérios de aceite e arquivos relevantes, com imagens quando o visual importa, e nunca segredos.

#codebase e outras referências

No Chat do VS Code, #codebase faz o Copilot buscar no workspace o contexto relevante sem anexar arquivo por arquivo. #file anexa um arquivo, #selection a seleção e #web aciona busca na web; também há o botão Add Context e arrastar arquivos.

Imagens como contexto

Mockups, capturas de tela e fluxogramas podem ser colados, arrastados ou anexados ao prompt, com um modelo que aceite imagens. Formatos: JPEG, PNG, GIF, WEBP, PDF, HEIC e HEIF; disponível em todos os planos, sem política para ligar.

Janela de contexto no uso diário

Pedido gigante com várias análises de uma vez gera respostas incompletas: divida em consultas menores e focadas. No Copilot CLI, /context mostra o uso e pesquisas pesadas podem ser delegadas a subagentes; também ajuda dividir em sessões menores e pedir para resumir saídas grandes de ferramentas.

Boas instruções personalizadas

Frases curtas e autocontidas, úteis para a maioria dos pedidos: visão geral do projeto, padrões e convenções, ferramentas e bibliotecas com versões. Em repositórios grandes podem não funcionar: pedir para consultar recursos externos (como um styleguide de outro repositório), impor estilo de resposta ou exigir tamanho fixo (como 'menos de 1000 caracteres'). Tarefas pontuais combinam com prompt files.

Boas práticas de Copilot Spaces

Uma tarefa por Space, instruções claras, de um a três exemplos de boa saída, fontes atualizadas e as mais importantes primeiro. Space inchado que cobre vários sistemas deve ser dividido em Spaces menores, podando fontes ruidosas.

Comentários e código limpo como contexto

Comentários que descrevem o código esperado (por exemplo, usar recursão ou um singleton) guiam a sugestão inline. Código com nomes descritivos e estilo consistente melhora as respostas; o próprio Copilot pode ajudar a comentar ou dividir funções grandes.

Macetes para memorizar

Shots

Zero = nenhum exemplo, One = um, Few = vários. Conte os exemplos do enunciado.

4 S

Single = UMA coisa; Specific = DETALHE útil; Short = SEM enrolação; Surround = ARREDORES (arquivos abertos e nomes).

Peso no prompt

Começo e fim gritam; meio sussurra.

Histórico

Assunto novo, conversa nova; pedido velho atrapalhando, apaga; conversa longa no mesmo tema, resume.

Contexto na CLI

@ para arquivo (arroba de anexo), # para número de issue.

Estrutura

Do geral ao específico: primeiro O QUE, depois COMO.

Iteração

Sugestão ruim não se conserta à mão: melhore o pedido e peça de novo.

Tabela de memorização: não confunda

Quando é...	...e quando é
One-shot Um único exemplo no prompt para o Copilot seguir o padrão.	**Few-shot** Vários exemplos de entrada e saída cobrindo casos diferentes.
Zero-shot Pedido sem exemplos, apoiado só no treinamento do modelo.	**Role prompting** Pedido que atribui uma persona de especialista para direcionar o foco.

Single O problema é misturar várias tarefas no mesmo prompt (API, testes e deploy juntos).	**Short** O problema é prolixidade: detalhes repetidos e explicações que não mudam o resultado.
Specific Deixar regras e formato de retorno explícitos no próprio pedido.	**Surround** Melhorar o contexto com nomes de arquivo descritivos e arquivos relacionados abertos.
Iniciar nova conversa Assunto totalmente novo, sem relação com a conversa anterior.	**Resumir o contexto anterior no novo pedido** Continuação do mesmo tema após conversa longa: leve só o essencial.
Excluir um pedido da conversa No Copilot Chat, um pedido antigo deixou de servir e atrapalha as respostas.	**/rewind ou /undo na CLI** No Copilot CLI, voltar a um prompt anterior descartando turnos; no cloud agent não há equivalente.

Números que caem

Item	Valor
Peso do domínio na prova	**10 a 15%**
Exemplos de boa saída recomendados num Copilot Space	**De um a três**
Formatos de imagem aceitos no Chat	**JPEG, PNG, GIF, WEBP, PDF, HEIC e HEIF**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Requisitos importantes vão no meio do prompt, onde o modelo presta mais atenção, ou repetidos em toda frase.
Como não cair: Início e fim recebem peso extra; coloque o requisito crítico no começo e não repita.

⚠ **Alerta de prova:** Listar todas as validações antes de dizer qual função se quer.
Como não cair: Objetivo geral primeiro, depois os requisitos específicos.

⚠ **Alerta de prova:** Para dar ênfase, escrever em maiúsculas, com exclamações ou repetir a meta-instrução no início e no fim.
Como não cair: Ênfase visual não é orientação documentada; escreva a regra direta, como 'retorne o resultado como JSON'.

⚠ **Alerta de prova:** Mais contexto sempre ajuda: abrir mais abas, adicionar todos os repositórios ao Space, copiar o manual inteiro nas instruções.
Como não cair: Contexto irrelevante vira ruído. Abra o relevante e feche o resto, divida Spaces inchados e mantenha instruções concisas.

⚠ **Alerta de prova:** Fechar todas as abas e deixar só o arquivo atual melhora as sugestões.
Como não cair: Se as funções estão em outro arquivo, abra esse arquivo relacionado em outra aba (Surround).

⚠ **Alerta de prova:** Trocar de modelo, de linguagem ou de IDE resolve resposta confusa ou histórico irrelevante.
Como não cair: A orientação é reformular, dividir, melhorar o código ou gerenciar o histórico (nova conversa, excluir pedido).

⚠ **Alerta de prova:** Pedir ao Copilot que 'esqueça' algo muda o histórico.
Como não cair: O pedido não remove nada; exclua o pedido da conversa ou comece uma nova.

⚠ **Alerta de prova:** Abrir conversa nova sem mencionar nada, ou colar todo o código anterior em cada turno, para continuar o mesmo tema.
Como não cair: Resuma o contexto essencial no novo pedido; recomeçar do zero perde contexto e colar tudo incha o processamento.

⚠ **Alerta de prova:** Na CLI, ! e & servem para anexar arquivo e issue, ou / para arquivo e @ para issue.
Como não cair: @caminho para arquivo e #número para issue; ! roda shell, & delega ao cloud agent, / inicia slash command.

⚠ **Alerta de prova:** No cloud agent dá para voltar turnos com Archive session, Stop session ou /rewind.
Como não cair: Só a CLI volta turnos (/undo, /rewind). No cloud agent isso não existe.

⚠ **Alerta de prova:** Para usar uma biblioteca específica, basta citá-la num comentário no fim do arquivo ou escolher o modelo mais caro.
Como não cair: Declare os imports no topo do arquivo ou diga explicitamente qual biblioteca usar.

⚠ **Alerta de prova:** Para guiar uma sugestão inline simples (recursão), usar plan mode, trocar de modelo ou mexer no filtro de código público.
Como não cair: Escreva um comentário pedindo recursão: comentários orientam a sugestão inline.

⚠ **Alerta de prova:** Para listar casos extremos sem mudar arquivos, usar o cloud agent, /fixTestFailure ou Agent com Bypass Approvals.
Como não cair: Use o Ask mode para analisar antes de gerar; depois o Agent gera e roda os testes.

⚠ **Alerta de prova:** Incluir na tarefa do cloud agent a senha de produção ou pedir merge direto na main. **Como não cair:** Objetivo, critérios de aceite e arquivos relevantes; segredos ficam fora e o PR passa por revisão e merge humanos.

Como cai na prova

📝 **Como cai:** Cenário descreve um prompt e pergunta qual técnica (zero, one, few-shot, role, chain). **Como resolver:** Conte os exemplos dados; se não há exemplo mas há persona, é role prompting; turnos encadeados construindo a solução é chain prompting.

📝 **Como cai:** Cenário descreve um defeito de prompt e pergunta qual dos 4 S foi violado ou reforçado. **Como resolver:** Várias tarefas = Single; falta de detalhes = Specific; excesso de texto = Short; nomes ruins e arquivos fechados = Surround. Cuidado: prompt curto pode violar Single.

📝 **Como cai:** Resposta ruim ou confusa e pergunta o que a documentação recomenda. **Como resolver:** Procure a alternativa que reformula, divide, dá exemplos, nomeia o alvo ou ajusta o contexto. Descarte trocar idioma, IDE, modelo, maiúsculas e repetir o mesmo pedido.

📝 **Como cai:** Gestão do histórico em conversa longa ou mudança de assunto. **Como resolver:** Mesmo tema: resumir o essencial. Tema novo: conversa nova. Pedido antigo inútil: excluí-lo. CLI: /rewind; cloud agent: não volta.

📝 **Como cai:** Como fornecer contexto numa superfície específica (inline, Chat, CLI, cloud agent). **Como resolver:** Inline = abas abertas; Chat = seleção, @ e #; CLI = @caminho e #número; cloud agent = objetivo, critérios de aceite e arquivos.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
vários exemplos de entrada e saída	**Few-shot**
função existente usada como modelo	**One-shot**
só o comentário, sem exemplos	**Zero-shot**
aja como especialista em segurança ou desempenho	**Role prompting**
API, testes e deploy no mesmo pedido	**Quebra o Single**
pedido vago reescrito com regras e formato de retorno	**Specific**

prompt de 25 linhas repetitivo	**Quebra o Short**
temp.js, novo2.js, arquivos relacionados abertos	**Surround**
contexto do workspace inteiro no Chat	**#codebase**
'o que isto faz?' respondido sobre o trecho errado	**Nomear o alvo (função, bloco ou última resposta)**
assunto novo depois de conversa longa	**Nova conversa**
mockup ou captura de tela	**Anexar imagem com modelo que aceite imagens**
arquivo enorme com várias análises de uma vez	**Dividir em consultas menores**
medir o contexto na CLI e aliviar pesquisas	**/context e subagentes**

🎯 Se você lembrar de só uma coisa deste capítulo...

- Zero-shot sem exemplo, one-shot com um, few-shot com vários.
- 4 S: Single, Specific, Short, Surround; saiba qual é violado em cada cenário.
- Objetivo geral primeiro, requisitos depois; início e fim do prompt pesam mais.
- Nova conversa para assunto novo; excluir pedido que atrapalha; resumir contexto em conversa longa.
- CLI: @arquivo e #issue; /rewind volta turnos; cloud agent não volta.

Essenciais para a prova (checklist)

- ☐ Sei diferenciar zero-shot, one-shot, few-shot e role prompting.
- ☐ Sei identificar qual dos 4 S está sendo aplicado ou violado num cenário.
- ☐ Sei estruturar um prompt do geral ao específico e onde pôr o requisito crítico.
- ☐ Sei como evitar ambiguidade (nomear o alvo, regra direta, imports no topo).
- ☐ Sei usar exemplos e testes como especificação.
- ☐ Sei gerenciar o histórico do chat em cada situação e voltar turnos na CLI.
- ☐ Sei como dar contexto em cada superfície (inline, Chat, CLI, cloud agent).
- ☐ Sei as boas práticas de instruções personalizadas e de Spaces.

Questões de revisão do capítulo

R1.1 · Aplicar engenharia de prompt e criação de contexto

Um desenvolvedor da Agência Vitrine precisa gerar o código de uma tela a partir de um mockup feito pelo time de design. Como ele dá esse contexto visual ao Copilot Chat no IDE?

A. Anexando a imagem do mockup ao prompt, com um modelo que aceite imagens
B. Pedindo ao code review que leia o mockup anexado ao pull request
C. Citando só o nome do arquivo da imagem no texto do prompt
D. Salvando o mockup na pasta .github/prompts do repositório

R1.2 · Aplicar engenharia de prompt e criação de contexto

As respostas do Copilot sobre o código legado da Têxtil Norte vêm confusas. As variáveis se chamam a1, x2 e tmp, não há comentários e o estilo muda a cada arquivo. O que a documentação do GitHub recomenda?

A. Melhorar o código com nomes descritivos e estilo consistente
B. Aumentar o número de abas abertas no editor de código
C. Desabilitar as sugestões inline no projeto inteiro
D. Trocar para um modelo maior antes de tentar outras medidas no projeto

R1.3 · Aplicar engenharia de prompt e criação de contexto

No Chat do VS Code, uma desenvolvedora da Clínica Pulsar quer que o Copilot busque no workspace o contexto relevante para gerar testes, sem anexar arquivo por arquivo. Que referência com # o Microsoft Learn indica para isso?

A. #web
B. #selection
C. #file
D. #codebase

R1.4 · Aplicar engenharia de prompt e criação de contexto

Um desenvolvedor quer uma função JavaScript que informe se um número é primo, retorne true para primos e gere erro se a entrada não for um inteiro positivo. Que estrutura de prompt a documentação do GitHub recomenda?

A. Colar o código de outra função parecida, sem nenhuma explicação
B. Escrever só o nome da função e deixar o Copilot deduzir o resto
C. Descrever o objetivo geral e depois os requisitos específicos
D. Listar todas as validações antes de dizer qual função é desejada

Respostas comentadas

R1.1. A) Anexando a imagem do mockup ao prompt, com um modelo que aceite imagens

Por quê: Formatos aceitos incluem JPEG, PNG, GIF, WEBP, PDF, HEIC e HEIF. Imagens podem ser coladas, arrastadas ou anexadas e estão disponíveis em todos os planos, sem política para ligar.

✗ B) Incorreta. O code review analisa mudanças de código, não gera telas a partir de imagens.
✗ C) Incorreta. Só o nome do arquivo não entrega o conteúdo visual ao modelo.
✗ D) Incorreta. Essa pasta guarda prompt files em Markdown, não imagens de referência.

> **Dica de prova:** Contexto visual: anexe a imagem e use um modelo com visão.

R1.2. A) Melhorar o código com nomes descritivos e estilo consistente

Por quê: Seguir boas práticas de código melhora as respostas do Copilot. A própria documentação sugere usar o Copilot para isso, pedindo comentários ou a divisão de funções grandes.

✗ B) Incorreta. Mais abas com código confuso não trazem clareza.
✗ C) Incorreta. Desligar o Copilot elimina a ajuda em vez de melhorá-la.
✗ D) Incorreta. Trocar de modelo não resolve um código difícil de ler; a recomendação é melhorar o código.

> **Dica de prova:** Código confuso gera respostas confusas: nomes claros e estilo consistente.

R1.3. D) #codebase

Por quê: Outras formas de dar contexto no Chat: botão Add Context, arrastar arquivos e mencionar com # arquivos, pastas e símbolos.

✗ A) Incorreta. #web aciona a busca na web.
✗ B) Incorreta. #selection anexa a seleção atual do editor.
✗ C) Incorreta. #file anexa um arquivo específico.

> **Dica de prova:** Contexto do workspace inteiro: #codebase.

R1.4. C) Descrever o objetivo geral e depois os requisitos específicos

Por quê: O exemplo oficial é: escreva uma função JavaScript que diga se um número é primo; a função deve receber um inteiro e retornar true se for primo; deve dar erro se a entrada não for inteiro positivo.

✗ A) Incorreta. Exemplos ajudam, mas sem explicação o Copilot não sabe o que mudar ou validar.
✗ B) Incorreta. Um pedido vago tende a gerar uma solução genérica.
✗ D) Incorreta. Começar pelos detalhes sem o objetivo dificulta o entendimento do pedido.

> **Dica de prova:** Primeiro o objetivo, depois os requisitos.

02

CAPÍTULO 2 · 10-15% DA PROVA

Configurar privacidade, exclusões de conteúdo e proteções

Este domínio (10 a 15%) gira em torno de três blocos: exclusão de conteúdo, filtro de código público com referências de código e solução de problemas no IDE. Na exclusão, a prova cobra planos (só Business e Enterprise), quem configura, alcance de cada nível, sintaxe dos caminhos, tempo de propagação, como testar e, principalmente, as limitações (Edit e Agent no VS Code, symlinks, sistemas de arquivos remotos e dados semânticos do IDE). No filtro de código público preciso saber Allow e Block, onde o usuário individual configura, a herança da organização e onde ficam os logs de referência no VS Code e no JetBrains. Meu ponto fraco são os detalhes de troubleshooting, como Chat Diagnostics, Set Log Level, Reload Window e o ícone riscado.

Neste capítulo

18 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Exclusão de conteúdo: planos e quem configura

Disponível só no Copilot Business e no Copilot Enterprise. Configurada por administradores do repositório, owners da organização e owners da enterprise; quem tem papel Maintain só vê as configurações, e licença do Copilot ou permissão write não bastam para editar.

Efeitos da exclusão

Nos arquivos excluídos não há sugestões inline, o conteúdo deles não informa sugestões em outros arquivos nem respostas do Copilot e eles não entram no Copilot code review. Os arquivos continuam visíveis no GitHub e nada é apagado.

Alcance por nível

A regra da enterprise vale para todos os usuários do Copilot na enterprise; a da organização, só para quem recebe licença dela; a do repositório afeta quem usa o Copilot ao trabalhar naquele repositório. No repositório, regras herdadas de cima aparecem como caixas cinza não editáveis no topo da página.

Sintaxe no repositório

Em Paths to exclude in this repository, cada caminho vai numa linha no formato - "/CAMINHO". Aceita fnmatch e não diferencia maiúsculas de minúsculas (*.cfg pega SERVIDOR.CFG). Com barra inicial o caminho é a partir da raiz; sem barra vale em qualquer lugar: "secrets.json" em qualquer pasta, "secret*" para nomes que começam com secret, "*.cfg" para a extensão em qualquer lugar.

Sintaxe na organização

Cada regra aponta um repositório (várias sintaxes de URL, inclusive de outras plataformas) seguido de caminhos, ou usa a chave "*" para valer em qualquer repositório e até em arquivos fora de repositórios Git. Não existe .copilotignore nem arquivo versionado de exclusão: tudo fica nas settings do GitHub.

Propagação e recarga

Mudanças podem levar até 30 minutos para valer em IDEs já abertos. No VS Code use Developer: Reload Window; no Visual Studio e nos JetBrains feche e reabra o aplicativo; no Vim/Neovim as regras são buscadas a cada arquivo aberto.

Limitações da exclusão

No VS Code cobre sugestões inline e o Chat em ask mode, mas não os modos Edit e Agent. Não vale para symlinks nem repositórios em sistemas de arquivos remotos, e o IDE pode repassar informação semântica (tipos, definições) e propriedades do projeto, como configuração de build. Para dados críticos, combine com outros controles.

Testar e diagnosticar a exclusão

Para sugestões inline, edite um arquivo excluído e confirme que nada é sugerido. Para o Chat, feche os outros arquivos, anexe o excluído e peça explain this file: ele não deve ser usado nem listado nas referências. O ícone do Copilot com traço diagonal na barra de status indica arquivo sob exclusão, e o tooltip mostra a origem da regra.

Auditar mudanças na exclusão

O rodapé da página de exclusão mostra quem mudou e quando, com link para o evento no audit log; os detalhes trazem o campo excluded_paths com as regras após a mudança. No audit log da organização o evento é copilot.content_exclusion_changed.

Filtro de código público (Allow e Block)

Em Suggestions matching public code: Block oculta sugestões iguais ou quase iguais a código público; Allow mostra a sugestão com referências (repositório e licença) para você decidir usar, atribuir ou remover. Assinante individual (que paga a própria assinatura, como no Copilot Pro) configura nas Copilot settings; com licença da organização ou enterprise, a política vem de cima e o usuário herda.

Referências de código nos IDEs

No VS Code ficam em Output, GitHub Copilot Log (Code References), com data e hora, arquivo, posição, trecho, licença (ou unknown) e URL do arquivo no GitHub.com. Nos JetBrains ficam no idea.log, buscando por [Public Code References]. Só são geradas se a configuração permitir correspondências ou se o produto não suportar Block; para testar, aceite algo comum como um fizzBuzz.

Código público no Copilot Chat

No Visual Studio, JetBrains, Eclipse e GitHub Mobile o Chat usa filtros que bloqueiam correspondências; no VS Code, com correspondências permitidas, o Chat mostra mensagem com link para o código e a licença. Em qualquer caso, trate como código de terceiros: testes, IP scanning e checagem de vulnerabilidades.

Proteções contratuais

Copilot Business e Enterprise incluem indenização de propriedade intelectual e o Data Protection Agreement. A indenização exige o filtro de código público configurado em Block, definido pelos admins na seção Privacy das políticas.

Configurações do editor

No VS Code, ligar ou desligar sugestões por linguagem é na chave github.copilot.enable do settings.json. Nos JetBrains, pelo ícone de status (desabilitar para a linguagem do arquivo) ou languageAllowList no github-copilot.xml, reiniciando o IDE. Recursos em preview do editor, como next edit suggestions (github.copilot.nextEditSuggestions.enabled), também dependem da política Editor preview features na organização.

Configurações pessoais de privacidade

Copilot access to Bing vem desabilitado por padrão no plano individual e a própria pessoa liga nas Copilot settings (em organizações, quem controla é o administrador). Para desligar o Copilot Free no GitHub.com, Show Copilot = Disabled; o Free é desativado sozinho ao assinar um plano pago ou receber licença de organização.

Aparelho perdido

Revogue o GitHub for VS Code em Authorized OAuth Apps e o app do Copilot nas autorizações. Depois de revogar, o VS Code ainda pode usar o Copilot na sessão atual por até 30 minutos.

Logs e diagnóstico no VS Code

Logs em View > Output, GitHub Copilot; para mais detalhe, Developer: Set Log Level com Trace (e voltar para Info depois). Para rede, proxy e firewall, Developer: Chat Diagnostics mostra a seção Reachability. Nos JetBrains, o log é o idea.log.

Problemas comuns e soluções

'GitHub Copilot could not connect to server. Extension activation failed' indica falta de plano ou falha no token: saia e entre de novo. Chat parou após atualização: atualize o VS Code e a extensão. Rate limit: aguarde e tente de novo, pois é temporário. Poucas sugestões em código muito comum (fizzBuzz) com plano individual: provavelmente o filtro de código público está bloqueando.

Macetes para memorizar

💡 Efeitos da exclusão

Sem inline, sem contexto, sem code review; mas visível e intacto.

💡 Limitações da exclusão

SAESR: Symlinks, Agent e Edit, Semântica do IDE, Remoto. Nesses o filtro falha.

💡 Barra no caminho

Com barra = raiz; sem barra = em todo lugar; maiúscula não importa.

💡 Block x Allow

Block BLOQUEIA a sugestão; Allow ALERTA com referência.

💡 Logs de referência

VS Code = Output (Code References); JetBrains = idea.log.

💡 Recarregar exclusões

30 minutos ou Reload Window; Vim olha a cada arquivo aberto.

💡 Troubleshooting VS Code

Rede = Chat Diagnostics; detalhe = Set Log Level Trace; conexão = sair e entrar; atualização = atualizar tudo.

Tabela de memorização: não confunda

Quando é...	...e quando é
Exclusão de conteúdo Impedir que o Copilot use certos arquivos como contexto e sugira neles (Business e Enterprise).	**Filtro de código público** Ocultar (Block) ou referenciar (Allow) sugestões que coincidem com código público.
Block Sugestão correspondente nem aparece; condição para a indenização de PI.	**Allow** Sugestão aparece e o Copilot registra referências com repositório e licença.

Regra da enterprise Vale para todos os usuários do Copilot na enterprise.	**Regra da organização** Vale só para quem recebe licença daquela organização.
Caminho com barra inicial ("/secrets.json") Aponta para o arquivo na raiz do repositório.	**Caminho sem barra ("secrets.json")** Vale em qualquer pasta do repositório.
Developer: Chat Diagnostics Diagnóstico de conectividade (seção Reachability) para proxy e firewall.	**Developer: Set Log Level** Aumentar o detalhe dos logs (Trace) em Output > GitHub Copilot.
github.copilot.enable (VS Code) Ligar ou desligar sugestões por linguagem no settings.json.	**languageAllowList / ícone de status (JetBrains)** Mesmo objetivo nos IDEs JetBrains, com reinício do IDE.
Output, GitHub Copilot Log (Code References) Referências de código público no VS Code.	**idea.log com [Public Code References]** Referências de código público nos IDEs JetBrains.

Números que caem

Item	Valor
Peso do domínio na prova	**10 a 15%**
Propagação de mudanças na exclusão para IDEs abertos	**Até 30 minutos**
Uso do Copilot após revogar a autorização no VS Code	**Até 30 minutos na sessão atual**
Trecho comparado pelo filtro de código público	**Cerca de 150 caracteres**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Exclusão de conteúdo está nos planos individuais mais caros (Pro+, Max) ou pode ser usada para desligar o Copilot por linguagem.
Como não cair: Só Business e Enterprise, configurada por admins; preferência por linguagem é configuração do editor.

⚠ **Alerta de prova:** Excluir arquivos esconde-os dos membros no GitHub ou os apaga da cópia local.
Como não cair: Só afeta o uso pelo Copilot; permissões e arquivos continuam iguais.

⚠ **Alerta de prova:** Quem tem papel Maintain, licença do Copilot ou write pode editar exclusões.
Como não cair: Editam admin do repositório, owner da organização e owner da enterprise; Maintain só lê.

⚠ **Alerta de prova:** Criar .copilotignore, .gitignore ou um .github/copilot-exclusions.yml versionado para excluir arquivos.
Como não cair: Exclusão é configurada nas settings do repositório, organização ou enterprise; .gitignore só controla versionamento.

⚠ **Alerta de prova:** A regra nova não pegou porque depende de commit, de 24 horas ou do próximo ciclo de cobrança.
Como não cair: Pode levar até 30 minutos; recarregue com Developer: Reload Window no VS Code ou reabra o Visual Studio e o JetBrains.

⚠ **Alerta de prova:** O agent mode leu o arquivo excluído porque a regra ainda está propagando ou porque falta BYOK.
Como não cair: A exclusão não é suportada nos modos Edit e Agent do VS Code; é limitação documentada.

⚠ **Alerta de prova:** O Copilot resolve symlinks antes de aplicar a regra, ou caminhos absolutos do servidor cobrem repositórios remotos.
Como não cair: Symlinks e sistemas de arquivos remotos não são cobertos.

⚠ **Alerta de prova:** O padrão '*.cfg' não pega SERVIDOR.CFG, ou '/cfg/*' pega a extensão .cfg em qualquer pasta.
Como não cair: Padrões são case insensitive; '/cfg/*' pega arquivos dentro da pasta /cfg, não a extensão.

⚠ **Alerta de prova:** Com Block, a sugestão aparece com link e licença ou destacada em vermelho.
Como não cair: Com Block ela não aparece; mostrar referência é comportamento de Allow.

⚠ **Alerta de prova:** O filtro de código público se configura no settings.json, num .copilotignore ou por comando na CLI.
Como não cair: Assinante individual: Copilot settings, Suggestions matching public code. Licença da organização: herdado da política.

⚠ **Alerta de prova:** Referências de código ficam no audit log, na aba Problems ou no github-copilot.xml.
Como não cair: VS Code: Output, GitHub Copilot Log (Code References). JetBrains: idea.log.

⚠ **Alerta de prova:** A indenização de PI depende de repositório público, BYOK ou code review.
Como não cair: Depende do filtro de código público em Block.

⚠ **Alerta de prova:** 'could not connect to server' é rate limit ou arquivo excluído; Chat quebrado após atualização se resolve voltando a versão antiga.
Como não cair: Conexão falhou: plano ou token, saia e entre de novo. Chat quebrado: atualize VS Code e extensão. Rate limit: espere e tente de novo.

⚠ **Alerta de prova:** Licença do Copilot expirada ou serviço fora do ar explica o ícone riscado.
Como não cair: Ícone com traço diagonal sinaliza exclusão de conteúdo no arquivo; o tooltip mostra a origem.

Como cai na prova

📝 **Como cai:** Arquivo deveria estar excluído, mas algo ainda o usa (agent mode, tipos em outro arquivo, IDE aberto há horas, symlink).
Como resolver: Identifique qual limitação explica: Edit e Agent não cobertos, dados semânticos do IDE, propagação de até 30 minutos, symlinks ou FS remoto. Descarte BYOK, plano errado e prazos inventados.

📝 **Como cai:** Escolha DUAS entre entradas de caminho para excluir arquivos.
Como resolver: Teste cada padrão: barra inicial ancora na raiz, sem barra vale em todo lugar, * casa qualquer trecho, maiúsculas são ignoradas; confira se a extensão bate exatamente.

📝 **Como cai:** Onde configurar ou consultar algo (filtro de código público, referências, logs, exclusão, Bing).
Como resolver: Pessoal = Copilot settings no GitHub; organização = políticas; editor = settings.json ou github-copilot.xml; logs = Output ou idea.log; exclusão = settings do repo, org ou enterprise.

📝 **Como cai:** Mensagem de erro ou sintoma no IDE e pergunta a primeira ação.
Como resolver: Associe o sintoma: could not connect = plano ou token; rate limit = esperar; Chat quebrou após update = atualizar; rede = Chat Diagnostics; ícone riscado = exclusão.

📝 **Como cai:** Usuário com licença da empresa não consegue mudar uma opção ou usar um recurso do editor.
Como resolver: A configuração vem herdada da política da organização ou enterprise (código público, Editor preview features); plano individual não resolve.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
ícone do Copilot com traço diagonal	**Arquivo sob exclusão de conteúdo**

excluir .env em qualquer repositório e fora do Git	**Chave "*" na exclusão da organização**
caixas cinza não editáveis na exclusão do repositório	**Regras herdadas da organização ou enterprise**
regra nova não aplicada no VS Code aberto desde cedo	**Até 30 minutos ou Developer: Reload Window**
agent mode acessou arquivo excluído	**Exclusão não cobre Edit e Agent no VS Code**
tipo de arquivo excluído usado corretamente em outro arquivo	**IDE repassa informação semântica**
testar se o Chat respeita a exclusão	**Anexar só o arquivo excluído e pedir explain this file**
quem mudou a exclusão e quando	**Rodapé da página com link para o audit log (excluded_paths)**
indenização de propriedade intelectual	**Filtro de código público em Block**
poucas sugestões em código comum, como fizzBuzz	**Filtro de código público bloqueando**
proxy e firewall corporativos, Reachability	**Developer: Chat Diagnostics**
notebook perdido	**Revogar GitHub for VS Code em Authorized OAuth Apps e o app do Copilot**
next edit suggestions não aparecem com licença Business	**Política Editor preview features**
Vim/Neovim e exclusões	**Regras buscadas a cada arquivo aberto**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- Exclusão de conteúdo: só Business e Enterprise; não cobre Edit e Agent no VS Code, symlinks, FS remoto nem dados semânticos do IDE.
- Propagação de até 30 minutos; VS Code recarrega com Developer: Reload Window.
- Block oculta, Allow referencia; indenização de PI exige Block.
- Referências: VS Code em Output (Code References), JetBrains no idea.log.
- Licença da organização herda as políticas; plano individual não contorna.

Essenciais para a prova (checklist)

- ☐ Sei em quais planos existe exclusão de conteúdo e quem pode configurá-la.
- ☐ Sei os efeitos e as limitações da exclusão de conteúdo.
- ☐ Sei escrever caminhos de exclusão no repositório e na organização (barra, fnmatch, "*").

- ☐ Sei quanto tempo a exclusão leva para valer e como recarregar em cada IDE.
- ☐ Sei testar a exclusão e auditar quem a alterou.
- ☐ Sei como funcionam Allow e Block e onde cada tipo de usuário configura.
- ☐ Sei onde ficam as referências de código no VS Code e no JetBrains.
- ☐ Sei ligar ou desligar sugestões por linguagem no VS Code e no JetBrains.
- ☐ Sei resolver erros comuns: conexão, rate limit, Chat após atualização, rede e proxy.
- ☐ Sei como proteger a conta quando um aparelho é perdido.

Questões de revisão do capítulo

R2.1 · Configurar privacidade, exclusões de conteúdo e proteções

Na conta pessoal de um desenvolvedor da Startup Fóton, a opção Suggestions matching public code está em Block. O que acontece quando o Copilot gera uma sugestão igual ou quase igual a código público?

A. A sugestão não é mostrada ao usuário
B. A sugestão aparece com link do repositório e a licença
C. O arquivo é excluído do Copilot
D. A sugestão é exibida em vermelho

R2.2 · Configurar privacidade, exclusões de conteúdo e proteções

O owner da organização da Rede Clara quer excluir do Copilot os arquivos .env em qualquer repositório e também os que estão em pastas fora de repositórios Git. Como ele faz isso na exclusão de conteúdo da organização?

A. Criando um .gitignore com .env em cada um dos repositórios
B. Pedindo a cada dev que feche os arquivos .env no editor
C. Listando cada repositório da organização com o caminho /.env
D. Usando a chave "*" com um padrão para arquivos .env

R2.3 · Configurar privacidade, exclusões de conteúdo e proteções

No IntelliJ IDEA, um desenvolvedor da Tecno Pampa quer desligar as sugestões do Copilot só para arquivos Kotlin, mantendo-as nas demais linguagens. O que a documentação do plugin indica?

A. Editar o copilot-instructions.md pedindo para ignorar arquivos Kotlin
B. Criar uma regra de exclusão de conteúdo para os arquivos .kt
C. Clicar no ícone de status e desabilitar para a linguagem do arquivo
D. Remover o plugin e instalar uma versão antiga sem Kotlin

R2.4 · Configurar privacidade, exclusões de conteúdo e proteções

Uma desenvolvedora usa o Copilot Free e quer desabilitar todos os recursos do Copilot no GitHub.com, sem assinar um plano pago. O que a documentação indica?

A. Configurar exclusão de conteúdo para todos os seus repositórios
B. Definir Show Copilot como Disabled nas Copilot settings
C. Excluir a conta do GitHub e criar outra sem o Copilot ativo
D. Revogar o token pessoal usado pelo navegador dela

Respostas comentadas

R2.1. A) A sugestão não é mostrada ao usuário

Por quê: Em produtos que não suportam o modo Block, ou com Allow, o Copilot mostra referências à correspondência para você decidir.

✗ B) Incorreta. Mostrar a sugestão com referência é o comportamento de Allow.
✗ C) Incorreta. O filtro não exclui arquivos.
✗ D) Incorreta. Não há destaque em vermelho; a sugestão é ocultada.

> **Dica de prova:** Block: sugestão igual a código público nem aparece.

R2.2. D) Usando a chave "*" com um padrão para arquivos .env

Por quê: Na organização, cada regra pode apontar um repositório (por várias sintaxes de URL, inclusive de outras plataformas) seguido de caminhos, ou usar "*" para qualquer caminho.

✗ A) Incorreta. O .gitignore controla o versionamento, não o uso pelo Copilot.
✗ B) Incorreta. Fechar arquivos não é uma política de exclusão.
✗ C) Incorreta. Listar repositório por repositório não cobre arquivos fora de repositórios Git.

> **Dica de prova:** Exclusão em qualquer lugar, até fora do Git: chave "*".

R2.3. C) Clicar no ícone de status e desabilitar para a linguagem do arquivo

Por quê: Nos IDEs JetBrains, o languageAllowList do github-copilot.xml aceita o curinga e exceções por linguagem, por exemplo desligar tudo menos Python e YAML. É preciso reiniciar o IDE após mudar.

✗ A) Incorreta. Instruções personalizadas orientam respostas; não desligam sugestões.
✗ B) Incorreta. Exclusão de conteúdo é política administrativa de Business e Enterprise, não preferência de linguagem.
✗ D) Incorreta. Remover o plugin não é necessário e não existe versão sem Kotlin.

> **Dica de prova:** JetBrains: desligar por linguagem pelo ícone ou languageAllowList.

R2.4. B) Definir Show Copilot como Disabled nas Copilot settings

Por quê: O Copilot Free é desativado automaticamente ao assinar um plano pago ou receber licença de uma organização. Nos IDEs, ele pode ser desligado nas configurações do Copilot.

✗ A) Incorreta. Exclusão de conteúdo é recurso de Business e Enterprise.

✗ C) Incorreta. Não é preciso excluir a conta.

✗ D) Incorreta. O navegador não usa um token pessoal para isso.

> **Dica de prova:** Desligar o Copilot Free no GitHub: Show Copilot = Disabled.

03

CAPÍTULO 3 · 10-15% DA PROVA

Entender a arquitetura e os dados do GitHub Copilot

Este domínio (10 a 15%) cobra o caminho do prompt até a sugestão: coleta de contexto, proxy, filtro de toxicidade, LLM, pós-processamento e entrega, e a ordem exata dessas etapas cai bastante. Também preciso saber o que vira contexto em cada cliente, como funciona a janela de contexto e a compactação, o que é LLM, fine-tuning e LoRA, e como os dados são tratados (DPA no Business e Enterprise, opção de treinamento nos planos individuais, indexação semântica, BYOK, Bing). Modelos também entram: Auto, utilitários, base e LTS. Onde mais erro é na ordem do fluxo e em achar que indexar ou usar o Copilot significa treinar modelos.

Neste capítulo

18 tópicos-chave · 12 pegadinhas · 6 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Fluxo de entrada

Primeiro o prompt é enviado por HTTPS e o contexto é coletado; depois o proxy filtra o tráfego; em seguida o filtro de toxicidade; por fim o LLM gera a sugestão. O proxy fica hospedado num tenant Microsoft Azure de propriedade do GitHub e, na entrada, barra tentativas de hackear o prompt.

Fluxo de saída

Pós-processamento e validação: o filtro de toxicidade remove conteúdo nocivo gerado e o proxy aplica checagens finais de qualidade, segurança e ética, como bugs comuns e vulnerabilidades (XSS, SQL injection), e opcionalmente o filtro de código público. Resposta reprovada é truncada ou descartada; depois vêm a entrega e o ciclo de feedback.

Ciclo de vida da sugestão

Ordem completa: contexto e prompt, proxy, filtro de toxicidade, LLM, pós-processamento, entrega. Só respostas que passam em todos os filtros chegam ao usuário, e o processo se repete a cada novo prompt.

Contexto coletado no editor

Código antes e depois do cursor, nome e tipo do arquivo, abas abertas, estrutura e caminhos do projeto, linguagens e frameworks. O pré-processamento Fill-in-the-Middle (FIM) faz o modelo considerar o código antes e depois do cursor ao mesmo tempo.

Filtro de toxicidade

Atua antes da geração contra discurso de ódio, conteúdo inapropriado e dados pessoais, como nomes e endereços; na saída também remove conteúdo nocivo. Não filtra idioma, licença de código nem tamanho de função.

LLM, fine-tuning e LoRA

LLMs são redes neurais treinadas em grandes volumes de texto que entendem e geram linguagem; não são regras manuais, buscadores de respostas prontas nem compiladores. Fine-tuning adapta um modelo pré-treinado com um conjunto menor e específico. LoRA adiciona partes treináveis menores e mantém o modelo original, alternativa eficiente ao ajuste fino completo.

Não determinismo

Modelos geram saídas de forma probabilística, então a mesma instrução pode não ser seguida sempre. Por isso uma instrução como 'responda em português' funciona quase sempre, mas não sempre; instruções curtas, claras e sem conflito aumentam a consistência sem garanti-la.

Janela de contexto

Tamanho fixo em tokens, que varia por modelo. É ocupada por instruções do sistema, instruções personalizadas, definições de ferramentas e MCP, mensagens, respostas e resultados de ferramentas; arquivos entram quando lidos ou anexados, não o repositório inteiro de uma vez.

Compactação (compaction)

Quando a janela se aproxima do limite, o histórico é substituído por um resumo estruturado, mantido junto com as instruções originais do usuário e o estado de planos e listas de tarefas. Não pode ser desfeita, e detalhes finos podem se perder. Sem ela, mensagens antigas seriam descartadas sem resumo nem registro. Existe na maioria dos clientes, mas não no Xcode.

Etapas do Copilot Chat

Processamento da entrada, análise do modelo, geração da resposta e formatação de saída. Na formatação, a resposta recebe destaque de sintaxe e indentação e pode trazer links para o contexto usado, como arquivos, issues ou documentação.

Contexto automático por cliente

No VS Code o Chat combina ao prompt o nome do repositório, os arquivos abertos e o copilot-instructions.md, se houver; com @github pode buscar contexto do GitHub e do Bing, se permitido. No Windows Terminal, o contexto é o nome do shell ativo e o histórico do chat, e os comandos sugeridos não rodam sozinhos.

Dados e treinamento

Copilot Business e Enterprise têm os dados protegidos pelo Data Protection Agreement e não treinam modelos; para eles a opção de treinamento nem aparece. Nos planos Free, Pro, Pro+ e Max, desde 24 de abril de 2026 o GitHub pode usar as interações para treinamento, com recusa em Allow GitHub to use my data for AI model training.

Indexação semântica

Repositórios do GitHub são indexados automaticamente para o Chat no GitHub e no VS Code e para o cloud agent, que acha código pelo significado. O índice não treina modelos e respeita a exclusão de conteúdo, filtrando os dados antes do Chat. Repositórios fora do GitHub exigem a política Semantic indexing for non-GitHub repositories (desabilitada por padrão), fazem upload dos dados ao GitHub e o recurso existe só no GitHub.com (não no GHE.com nem no GitHub Enterprise Server).

Correspondência com código público

A sugestão e cerca de 150 caracteres ao redor são comparados com um índice de repositórios públicos do GitHub.com, atualizado periodicamente (pode não ter código recente). Correspondências ocorrem em menos de 1% das sugestões; detalhes são registrados quando uma sugestão aceita, sem alterações, corresponde a código público. A geração probabilística é o motivo de isso poder acontecer.

BYOK e Bing

Com BYOK, prompts e respostas vão ao provedor escolhido, sob as políticas de retenção e privacidade dele, mas as respostas passam pela API do GitHub e podem ter filtro de conteúdo; no agent mode algumas ações ainda usam modelos integrados. Na busca do Bing, o prompt e contexto adicional geram uma consulta, sob a Privacy Statement da Microsoft.

Tipos de modelo

Escolha manual, Auto (decide pela complexidade da tarefa e pela disponibilidade, respeitando plano e políticas, com desconto em planos pagos), BYOK, modelos utilitários (rodam em segundo plano, como mensagem de commit e títulos de sessões de chat, fora do seletor) e, para Business e Enterprise, modelo base e modelo LTS. Os modelos disponíveis dependem do plano, da superfície e das políticas dos administradores.

Prompts de recursos de PR

O resumo de PR (Summary) monta o prompt a partir dos diffs dos arquivos e ignora a descrição já existente. O preenchimento de texto da descrição usa o título, o texto já escrito, títulos de commits, diffs parciais e títulos de PRs e issues vistos recentemente; em PRs grandes parte do conteúdo pode não caber na chamada à API.

Pipeline das sugestões inline e exclusão

Nas sugestões inline o modelo fica restrito a prever edições de código em formato rígido, com contexto estruturado e delimitado e prompt de sistema com políticas de conteúdo. Com exclusão de conteúdo configurada, o IDE envia ao GitHub só a URL do repositório atual para obter a política, e essas URLs não são registradas em log.

Macetes para memorizar

💡 Ordem do fluxo

C-P-T-L-P-E: Contexto, Proxy, Toxicidade, LLM, Pós-processamento, Entrega.

💡 Etapas do Chat

Entra, pensa, gera, formata.

💡 LoRA

LoRA = Low-Rank: peças pequenas plugadas, modelo base intacto.

💡 Indexar x treinar

Índice serve para BUSCAR, nunca para TREINAR.

💡 Treinamento por plano

Empresa (Business/Enterprise) = DPA, sem treino; Pessoa (Free/Pro/Pro+/Max) = pode treinar, salvo opt-out.

💡 Código público

150 caracteres, menos de 1%, só repos públicos do GitHub.com.

Tabela de memorização: não confunda

Quando é...	...e quando é
Proxy na entrada Filtra o tráfego e barra tentativas de hackear o prompt, antes do filtro de toxicidade e do LLM.	**Proxy no pós-processamento** Aplica checagens finais de bugs comuns, vulnerabilidades e, opcionalmente, código público; reprovou, é truncada ou descartada.
Fine-tuning Conceito geral: adaptar um modelo pré-treinado com dados menores e específicos.	**LoRA** Método eficiente de fine-tuning: adiciona partes treináveis pequenas e mantém o modelo original.
Fill-in-the-Middle (FIM) Pré-processamento que usa o código antes e depois do cursor na mesma sugestão.	**Few-shot ou RAG** Few-shot é dar vários exemplos no prompt; RAG não é a técnica descrita para o cursor.

Business e Enterprise DPA, dados não treinam modelos e a opção de treinamento não aparece.	**Free, Pro, Pro+ e Max** Interações podem treinar modelos desde 24/04/2026, salvo recusa em Allow GitHub to use my data for AI model training.
Modelo Auto Escolhe o modelo a cada tarefa por complexidade e disponibilidade.	**Modelo LTS** Estabilidade: suporte por um ano a partir da designação (Business e Enterprise).
Modelo utilitário Roda em segundo plano para mensagem de commit e títulos de sessões de chat; não aparece no seletor.	**Modelo base** Usado automaticamente em Business e Enterprise quando nenhum outro está habilitado.

Números que caem

Item	Valor
Peso do domínio na prova	**10 a 15%**
Trecho comparado com código público	**Cerca de 150 caracteres**
Frequência de correspondência com código público	**Menos de 1% das sugestões**
Início do uso de dados dos planos individuais para treinamento	**24 de abril de 2026 (com opção de recusa)**
Suporte de um modelo LTS	**Um ano a partir da designação**
Indexação inicial em repositórios grandes	**Até cerca de 60 segundos**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Na ordem do fluxo, o LLM vem antes do proxy, ou a entrega acontece antes do pós-processamento.
Como não cair: Contexto, proxy, toxicidade, LLM, pós-processamento, entrega. Nada é gerado antes do prompt filtrado e nada é entregue antes das checagens.

⚠ **Alerta de prova:** A primeira etapa do fluxo de entrada é o proxy no tenant Azure.
Como não cair: Primeiro vem o envio por HTTPS e a coleta de contexto; o proxy é a segunda etapa.

⚠ **Alerta de prova:** Resposta reprovada nas checagens é exibida com aviso, reenviada ao modelo ou guardada para treino.
Como não cair: É truncada ou descartada antes de chegar ao usuário.

⚠ **Alerta de prova:** O pós-processamento compila ou executa o código numa sandbox, ou traduz a resposta.
Como não cair: As checagens são bugs comuns, vulnerabilidades e o filtro opcional de código público; não há compilação, execução nem tradução.

⚠ **Alerta de prova:** Business não treina porque o Copilot processa tudo localmente, sem enviar código.
Como não cair: O prompt com contexto vai ao serviço; o motivo é contratual (Data Protection Agreement).

⚠ **Alerta de prova:** O Copilot Free nunca usa dados para treinamento, então migrar para ele resolve.
Como não cair: O Free também está entre os planos cujas interações podem ser usadas, salvo recusa.

⚠ **Alerta de prova:** Indexar o repositório entrega o código para treinar modelos ou o deixa público.
Como não cair: O índice não treina modelos e serve a quem tem acesso ao repositório.

⚠ **Alerta de prova:** Indexação de repositórios do GitLab funciona local e vem liberada por padrão, inclusive no GHE.com.
Como não cair: Faz upload ao GitHub, depende de política desabilitada por padrão e só existe no GitHub.com.

⚠ **Alerta de prova:** O índice de código público inclui GitLab, Bitbucket, npm ou repositórios privados.
Como não cair: Só repositórios públicos do GitHub.com, com atualização periódica.

⚠ **Alerta de prova:** Com BYOK as respostas vão direto ao IDE sem filtro, ou são reescritas pelo modelo padrão.
Como não cair: Passam pela API do GitHub e podem ter filtro de conteúdo; a qualidade depende do provedor.

⚠ **Alerta de prova:** O Auto escolhe pelo idioma do prompt, por sorteio ou sempre o modelo mais caro.
Como não cair: Escolhe pela complexidade da tarefa e disponibilidade em tempo real, respeitando plano e políticas.

⚠ **Alerta de prova:** A compactação preserva todos os detalhes ou descarta as instruções originais.
Como não cair: É resumo (detalhes podem se perder), irreversível, e mantém as instruções originais do usuário e o estado de planos.

Como cai na prova

📝 **Como cai:** Quatro sequências com as mesmas etapas embaralhadas; pede a ordem correta do ciclo da sugestão.
Como resolver: Ancore nas pontas: começa com contexto e prompt, termina com entrega. Proxy e toxicidade vêm antes do LLM; pós-processamento vem depois do LLM e antes da entrega.

📝 **Como cai:** Pergunta sobre o papel de um componente (proxy, filtro de toxicidade, pós-processamento).
Como resolver: Proxy entrada = barra ataque ao prompt; toxicidade = ódio, impróprio e dados pessoais; pós-processamento = bugs, vulnerabilidades, código público opcional e descarte do que reprova.

📝 **Como cai:** Gestor ou auditor preocupado com treinamento de modelos (licença, indexação, BYOK).
Como resolver: Identifique o plano: Business e Enterprise têm DPA. Indexação não treina. BYOK leva os dados ao provedor escolhido, sob as políticas dele.

📝 **Como cai:** Definição conceitual de LLM, fine-tuning ou LoRA com distratores de outras áreas.
Como resolver: Elimine regras manuais, buscadores, compiladores, filtragem de respostas e ajuste de prompt; fique com rede neural, adaptação de modelo pré-treinado e camadas pequenas.

📝 **Como cai:** O que ocupa a janela de contexto ou o que acontece na compactação.
Como resolver: Janela = sistema, instruções, ferramentas, MCP, mensagens e resultados. Compactação = resumo estruturado, sem volta, com instruções originais preservadas.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
código antes e depois do cursor ao mesmo tempo	**Fill-in-the-Middle (FIM)**
tenant Azure do GitHub, barrar hacking do prompt	**Proxy na entrada**
ódio, conteúdo inapropriado, nomes e endereços	**Filtro de toxicidade**
XSS, SQL injection, checagem final	**Pós-processamento pelo proxy**
partes treináveis menores, modelo original mantido	**LoRA**
instrução seguida na maioria das vezes, mas não sempre	**IA não determinística**
janela quase cheia, histórico vira resumo	**Compactação**
dados do Business não treinam	**Data Protection Agreement**
opção de treinamento não aparece nas settings	**Licença Business ou Enterprise**

repositório fora do GitHub com busca semântica	**Política Semantic indexing for non-GitHub repositories, upload ao GitHub**
cerca de 150 caracteres	**Comparação com código público**
mensagem de commit e título de chat gerados em segundo plano	**Modelos utilitários**
suporte garantido por período estendido	**Modelo LTS, um ano**
Windows Terminal	**Shell ativo + histórico do chat**

🎯 Se você lembrar de só uma coisa deste capítulo...

- Ordem: contexto, proxy, toxicidade, LLM, pós-processamento, entrega.
- Reprovou no pós-processamento: truncada ou descartada.
- Business e Enterprise: DPA, sem treinamento; individuais: opt-out em Allow GitHub to use my data for AI model training.
- Indexação não treina modelos e respeita a exclusão de conteúdo.
- IA é não determinística: instruções aumentam consistência, não garantem.

Essenciais para a prova (checklist)

- ☐ Sei colocar em ordem as etapas do fluxo de entrada e de saída.
- ☐ Sei o papel do proxy, do filtro de toxicidade e do pós-processamento.
- ☐ Sei o que é coletado como contexto no editor e o que é FIM.
- ☐ Sei definir LLM, fine-tuning e LoRA.
- ☐ Sei o que ocupa a janela de contexto e como a compactação funciona.
- ☐ Sei como os dados são tratados por plano e onde fica a opção de treinamento.
- ☐ Sei as regras da indexação semântica, inclusive para repositórios fora do GitHub.
- ☐ Sei como funcionam BYOK, Bing, Auto, modelos utilitários, base e LTS.

Questões de revisão do capítulo

R3.1 · Entender a arquitetura e os dados do GitHub Copilot

Em uma sessão longa do Copilot CLI, a desenvolvedora da Fintech Rubi percebe que a janela de contexto está quase cheia. Além das mensagens que ela digitou, o que ocupa essa janela?

A. As credenciais salvas da usuária e o histórico de navegação dela
B. Somente o último prompt, porque o histórico é descartado a cada turno da conversa
C. O repositório inteiro, carregado de uma vez ao abrir a sessão da CLI
D. Instruções do sistema, ferramentas, respostas e resultados de ferramentas

R3.2 · Entender a arquitetura e os dados do GitHub Copilot

Segundo o módulo do Microsoft Learn sobre LLMs, o que caracteriza os grandes modelos de linguagem usados pelo GitHub Copilot?

A. Regras escritas à mão por engenheiros do GitHub
B. Bancos de dados que buscam respostas prontas em fóruns de programação da internet aberta
C. Compiladores que traduzem código entre linguagens
D. Redes neurais treinadas em grandes volumes de texto que entendem e geram linguagem

R3.3 · Entender a arquitetura e os dados do GitHub Copilot

A documentação de gerenciamento de contexto explica o que aconteceria se o Copilot não tivesse o recurso de compactação quando a janela de contexto enchesse. O que aconteceria?

A. Mensagens antigas seriam descartadas sem resumo nem registro
B. O histórico completo seria enviado ao usuário por e-mail
C. A sessão seria salva e reaberta sem perder informação alguma
D. O Copilot trocaria sozinho para um modelo com janela maior

R3.4 · Entender a arquitetura e os dados do GitHub Copilot

Na Distribuidora Oeste, a busca na web está habilitada para o Copilot Chat no GitHub.com. Quando o Chat recorre ao Bing para responder, como o prompt do usuário é usado?

A. O prompt e contexto adicional geram uma consulta, sob a Privacy Statement da Microsoft
B. O Bing recebe o repositório inteiro do usuário para indexar e buscar no código
C. A consulta sai do navegador do desenvolvedor usando os cookies da sessão dele
D. O prompt vai ao Bing sem alteração e fica publicado na web para outros usuários verem

Respostas comentadas

R3.1. D) Instruções do sistema, ferramentas, respostas e resultados de ferramentas

Por quê: A janela de contexto tem tamanho fixo em tokens, que varia por modelo. Na CLI, /context mostra o uso por categoria: prompt do sistema, instruções, ferramentas, MCP, mensagens e espaço livre.

✗ A) Incorreta. Credenciais e navegação não fazem parte do contexto descrito.

✗ B) Incorreta. O histórico permanece e acumula; por isso existe a compactação.

✗ C) Incorreta. Arquivos entram quando lidos por ferramentas ou anexados, não o repositório todo de uma vez.

> **Dica de prova:** Contexto = sistema + ferramentas + conversa + resultados de ferramentas.

R3.2. D) Redes neurais treinadas em grandes volumes de texto que entendem e geram linguagem

Por quê: No Copilot, o LLM considera o arquivo atual e outros arquivos e abas abertos para gerar sugestões contextuais. LLMs podem ser ajustados para tarefas específicas.

✗ A) Incorreta. O comportamento vem do treinamento, não de regras manuais.

✗ B) Incorreta. LLMs geram texto a partir do que aprenderam; não são buscadores de respostas prontas.

✗ C) Incorreta. LLMs não são compiladores, embora ajudem a traduzir código.

> **Dica de prova:** LLM: rede neural treinada em muito texto.

R3.3. A) Mensagens antigas seriam descartadas sem resumo nem registro

Por quê: A compactação substitui o histórico por um resumo inteligente, preservando objetivos, decisões e próximos passos, em vez de simplesmente apagar mensagens.

✗ B) Incorreta. Não há envio do histórico por e-mail.

✗ C) Incorreta. Haveria perda de contexto; é isso que a compactação evita.

✗ D) Incorreta. A documentação não descreve troca automática de modelo nesse caso.

> **Dica de prova:** Sem compactação: mensagens antigas sumiriam sem resumo.

R3.4. A) O prompt e contexto adicional geram uma consulta, sob a Privacy Statement da Microsoft

Por quê: A busca na web ajuda em temas recentes ou muito específicos. O administrador da enterprise pode habilitar ou desabilitar o Bing; para planos individuais, a opção Copilot access to Bing vem desabilitada por padrão.

✗ B) Incorreta. O Bing não recebe o repositório para indexar.

✗ C) Incorreta. A consulta é feita pelo serviço em nome do usuário, não pelo navegador dele.

✗ D) Incorreta. O prompt é usado para gerar uma consulta; não é publicado na web.

> **Dica de prova:** Bing no Chat: prompt vira consulta, sob a política de privacidade da Microsoft.

04

CAPÍTULO 4 · 10-15% DA PROVA

Melhorar a produtividade do desenvolvedor com o GitHub Copilot

Este domínio (10 a 15%) é sobre escolher o recurso certo para cada tarefa do dia a dia: gerar código, refatorar, documentar, aprender, depurar, gerar dados de exemplo, modernizar legado e, principalmente, testar. Testes caem muito: /tests, /setupTests, /fixTestFailure, Fix Test Failure no Test Explorer, ghost text em arquivo que já tem padrão, Ask para explorar, Plan para estratégia complexa e Agent para gerar e rodar. Também preciso saber quando usar sugestões inline e quando usar o Chat, e o que o Copilot faz bem (testes, código repetitivo, explicar, regex). Erro quando confundo os slash commands de testes entre si e os do VS Code com os do Visual Studio (/doc e /optimize).

Neste capítulo
17 tópicos-chave · 12 pegadinhas · 6 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Inline x Chat

Sugestões inline servem para completar código, nomes e funções enquanto você digita, gerar código repetitivo, transformar comentários em código e gerar testes em TDD. O Chat serve para perguntas em linguagem natural, gerar grandes blocos e iterar, tarefas com palavras-chave e personas.

Onde o Copilot se sai melhor

Escrever testes e código repetitivo, depurar e corrigir sintaxe, explicar e comentar código e gerar expressões regulares. Não decide arquitetura sozinho, não responde bem fora de tecnologia e não substitui a expertise do desenvolvedor.

Slash commands de produtividade

/explain explica o código ativo; /fix propõe correções; /tests gera testes de unidade do código selecionado (no agent mode pode gravar direto no arquivo de testes, detectando framework e estilo); /new cria projeto. No Visual Studio, /doc adiciona comentário de documentação e /optimize propõe melhorias de tempo de execução.

Comandos de testes

/setupTests recomenda e configura um framework quando o projeto ainda não tem testes. /fixTestFailure ou o botão Fix Test Failure no Test Explorer diagnosticam e corrigem um teste que falha; a correção pode mudar o código, o teste ou ambos, aplicada com Keep ou descartada com Undo, e depois o teste roda de novo.

Fluxo de testes com os modos

Ask para comparar casos, achar casos extremos e escolher framework sem mudar arquivos; Plan para estratégia que abrange várias classes, fixtures e convenções, como contrato revisável; Agent para gerar e rodar. Para o Agent rodar, corrigir e rodar até passar, peça isso explicitamente no prompt; para testes de integração, peça em linguagem natural e mande executar.

Ghost text em arquivo de testes

Quando o arquivo já tem casos no padrão (por exemplo Arrange-Act-Assert), escrever um comentário após o último teste e deixar o ghost text completar é o jeito mais rápido de ampliar a cobertura. Para criar classes novas, volte ao Chat com Ask, Plan ou Agent.

Casos extremos e asserções

O Chat sugere casos extremos como nulos, tipos inesperados e limites, além de parâmetros, saídas esperadas e asserções adequadas ao comportamento. Testes gerados são ponto de partida: podem não cobrir tudo.

Infraestrutura de testes e dados de exemplo

Para dependências externas (como uma API de pagamentos), peça mocks, fixtures e funções auxiliares em vez de chamar a API real. Quando dados reais são proibidos, o Copilot gera dados de exemplo realistas; não anonimiza bancos de produção.

Refatoração e modernização

O Copilot sugere sintaxe moderna e idiomas mais concisos, ajuda a manter consistência de estilo e pode dividir uma função gigante em funções menores. Em código legado (por exemplo COBOL para Java), explica o código e ajuda a traduzir, sempre com validação humana.

Agent mode para mudanças amplas

Migrar para um framework moderno ou renomear um conceito em dezenas de arquivos, atualizando imports, testes e documentação e verificando o build: agent mode, que entende a estrutura do projeto, edita vários arquivos e roda comandos.

Depuração

Selecione o método e pergunte no inline chat citando o erro (por exemplo NullReferenceException). Para bug difícil de reproduzir, peça instruções de log estratégicas e casos de teste. Em falha de workflow do GitHub Actions, o Copilot explica o erro a partir dos logs e sugere a correção.

Aprendizado e menos troca de contexto

Ajuda dentro do editor, sugerindo chamadas e parâmetros de APIs e explicando código, reduz idas ao navegador e à documentação. Acelera aprender uma linguagem nova e entender um repositório desconhecido; no GitHub.com o Chat usa o índice de busca semântica criado automaticamente.

Alcance além do código de aplicação

Além de Python, JavaScript, TypeScript, Ruby, Go, C# e C++, o Copilot ajuda em consultas de banco de dados, APIs e infraestrutura como código, e em ciência de dados (valores ausentes, codificação de variáveis, escalonamento, gráficos com Matplotlib e Seaborn). Gerar boilerplate e estrutura inicial libera tempo para a lógica de negócio.

Documentação e mensagens

Gera comentários inline, descrições de funções com parâmetros e retorno e sugestões para o README. No GitHub Desktop, selecionar as linhas ou arquivos específicos antes melhora a mensagem de commit sugerida, que deve ser revisada e editada.

Segurança e revisão cedo

Peça uma revisão do Copilot no IDE antes do commit para achar problemas de estilo e segurança cedo; depois vêm o code review no GitHub.com e os checks. Exemplo cobrado: trocar eval() com entrada do usuário por um parser seguro como JSON.parse(). Sugestões do code review podem ser agrupadas em um único commit.

Cloud agent para backlog

Bons usos: melhorar cobertura de testes, tratar dívida técnica bem delimitada e atualizar documentação, com descrição clara, critérios de aceite e dicas de arquivos. O agente valida o próprio código com CodeQL, secret scanning e a GitHub Advisory Database, sem exigir GHAS, e tenta resolver o que encontra antes de concluir o PR.

Medir impacto

Use as métricas de uso da API do Copilot (usuários ativos, taxa de aceitação, linhas aceitas) junto com pesquisas periódicas com os desenvolvedores, nas fases de avaliação, adoção, otimização e eficiência contínua.

Macetes para memorizar

Comandos de teste

SETUP monta o framework, TESTS cria testes, FIX TEST FAILURE conserta o que quebrou.

Fluxo de testes

Ask explora, Plan combina, Agent executa, ghost text estende.

Visual Studio

/doc DOCUMENTA, /optimize OTIMIZA; os dois são do Visual Studio no cheat sheet.

Inline x Chat

Inline completa enquanto eu digito; Chat conversa e itera.

Testes sem dependência real

Fora do teste: produção e API real. Dentro: mocks, fixtures e dados de exemplo.

Impacto

Números da API + voz do time (pesquisa).

Tabela de memorização: não confunda

Quando é...	...e quando é
/tests Gerar testes de unidade para o código selecionado quando já existe framework.	**/setupTests** O projeto ainda não tem framework de testes: recomenda e configura.
/fixTestFailure Um teste já existe e está falhando.	**/tests** Criar testes novos; não roda nem corrige sozinho o ciclo até passar.
Ghost text Estender casos parecidos num arquivo de testes que já tem padrão.	**Plan seguido de Agent** Estratégia de testes complexa com várias classes, fixtures e convenções não documentadas.
Agent mode Mudança em muitos arquivos com comandos, dependências, testes e build.	**Inline chat ou quick chat** Pergunta ou ajuste perto do código atual (inline) ou pergunta rápida sobre API e sintaxe (quick chat).
/optimize (Visual Studio) Melhorar o tempo de execução do método selecionado.	**/doc (Visual Studio)** Adicionar comentário de documentação ao símbolo selecionado.
Sugestões inline TDD no editor, completar enquanto digita, comentário que vira código.	**Copilot Chat** Perguntas, blocos grandes com iteração e personas.
Mocks e fixtures Simular uma dependência externa nos testes.	**Dados de exemplo** Massa de teste realista quando é proibido usar dados reais.

Números que caem

Item	Valor
Peso do domínio na prova	**10 a 15%**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** /tests gera, roda e corrige os testes sozinho até passarem.
Como não cair: /tests gera testes; o ciclo rodar, corrigir e rodar de novo deve ser pedido explicitamente ao Agent.

⚠ **Alerta de prova:** O Ask mode cria arquivos de teste e roda a suíte.
Como não cair: Ask não edita arquivos nem roda comandos; serve para analisar antes.

⚠ **Alerta de prova:** /fixTestFailure serve para criar testes de integração do zero ou levantar cenários.
Como não cair: É só para corrigir teste que falha. Testes novos: Agent ou /tests; cenários: Ask.

⚠ **Alerta de prova:** /setupTests entrega testes de integração prontos ou reconfigura um framework que já existe.
Como não cair: Ele recomenda e configura framework quando ainda não há testes.

⚠ **Alerta de prova:** Ghost text funciona bem num arquivo de testes vazio.
Como não cair: Ghost text rende quando o arquivo já mostra o padrão a seguir.

⚠ **Alerta de prova:** Copiar dados de produção ou chamar a API real nos testes deixa tudo mais realista.
Como não cair: Gere dados de exemplo e mocks; dados reais trazem risco de privacidade e a API real deixa testes lentos, frágeis e pode gerar cobranças.

⚠ **Alerta de prova:** Trocar eval() por setTimeout com a mesma string ou mover para arquivo excluído resolve a falha.
Como não cair: setTimeout com string também avalia código e exclusão só esconde do Copilot; use parser seguro como JSON.parse().

⚠ **Alerta de prova:** Para melhorar uma função gigante, só comentar cada linha ou apagá-la e gerar outra do zero.
Como não cair: Peça para dividir em funções menores; comentar não resolve a estrutura e reescrever do zero arrisca regressão.

⚠ **Alerta de prova:** O plan mode converte um sistema legado inteiro com um clique, e a tradução dispensa revisão.
Como não cair: Plan monta um plano revisável; o Copilot explica e ajuda a traduzir, e tudo exige revisão e testes.

⚠ **Alerta de prova:** O cloud agent pode aprovar e mesclar seus PRs ou mudar vários repositórios na mesma tarefa.
Como não cair: Não aprova nem mescla e altera só o repositório indicado ao iniciar.

⚠ **Alerta de prova:** O cloud agent bloqueia toda dependência nova ou exige GHAS para validar segurança.
Como não cair: Valida com CodeQL, secret scanning e Advisory Database sem exigir GHAS; dependências novas são verificadas contra malware e vulnerabilidades altas ou críticas.

⚠ **Alerta de prova:** Medir impacto pelo número de commits às sextas, linhas apagadas ou issues fechadas pelo owner.
Como não cair: Métricas de uso da API do Copilot mais pesquisa com os desenvolvedores.

Como cai na prova

📝 **Como cai:** Cenário de testes e pergunta qual comando, modo ou recurso usar.
Como resolver: Pergunte em que estado o projeto está: sem framework (/setupTests), teste quebrado (/fixTestFailure), arquivo com padrão (ghost text), estratégia complexa (Plan e Agent), só analisar (Ask), gerar do selecionado (/tests).

📝 **Como cai:** Tarefa de produtividade e escolha entre inline, inline chat, Chat, agent mode e cloud agent.
Como resolver: Tamanho e autonomia decidem: digitando = inline; pergunta no código = inline chat; conversa = Chat; muitos arquivos com comandos = agent mode; backlog delimitado em segundo plano = cloud agent.

📝 **Como cai:** Como o Copilot ajuda numa tarefa (legado, linguagem nova, ciência de dados, depuração).
Como resolver: A alternativa certa descreve o Copilot sugerindo, explicando ou gerando código com revisão humana; as erradas dizem que ele executa, publica, acessa produção ou decide sozinho.

📝 **Como cai:** Slash command com alternativas de outros comandos reais.
Como resolver: Associe cada comando a um verbo: explain explica, fix corrige, tests testa, doc documenta, optimize otimiza, new cria projeto, help ajuda.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
projeto sem framework de testes	**/setupTests**
teste falhando no Test Explorer	**Fix Test Failure ou /fixTestFailure**
arquivo de testes já com casos no padrão	**Ghost text a partir de comentário**
estratégia de testes com várias classes e convenções	**Plan, depois Agent**
rodar, corrigir e rodar de novo até passar	**Pedir explicitamente ao Agent**
dependência externa nos testes	**Mocks, fixtures e helpers**
política proíbe dados reais em teste	**Dados de exemplo gerados**
COBOL ou código legado que poucos entendem	**Explicar e ajudar a traduzir, com revisão**
renomear conceito em dezenas de arquivos e checar build	**Agent mode**
desempenho no Visual Studio	**/optimize**
comentário de documentação no Visual Studio	**/doc**
eval() com entrada do usuário	**Parser seguro como JSON.parse()**
workflow do Actions falhou	**Copilot explica o erro pelos logs e sugere correção**
regex, boilerplate, testes, explicar código	**Pontos fortes do Copilot**

🎯 Se você lembrar de só uma coisa deste capítulo...

- /setupTests configura framework, /tests gera, /fixTestFailure corrige teste que falha.
- Para o Agent rodar e corrigir até passar, peça isso explicitamente.
- /doc e /optimize aparecem no cheat sheet do Visual Studio.
- Testes gerados são ponto de partida; revisão e teste manual continuam.
- Cloud agent valida com CodeQL, secret scanning e Advisory Database, sem exigir GHAS.

Essenciais para a prova (checklist)

- ☐ Sei quando usar sugestões inline e quando usar o Chat.
- ☐ Sei os slash commands de produtividade e de testes, inclusive os do Visual Studio.
- ☐ Sei montar o fluxo de testes com Ask, Plan, Agent e ghost text.
- ☐ Sei pedir casos extremos, asserções, mocks, fixtures e dados de exemplo.

- ☐ Sei como o Copilot ajuda em refatoração, legado, documentação e depuração.
- ☐ Sei quando o agent mode é a escolha para mudanças amplas.
- ☐ Sei bons usos do cloud agent e como ele valida segurança.
- ☐ Sei como medir o impacto do Copilot na produtividade.

Questões de revisão do capítulo

R4.1 · Melhorar a produtividade do desenvolvedor com o GitHub Copilot

Um desenvolvedor da Rádio Onda recebe uma NullReferenceException em um método e não entende a causa. Segundo o Microsoft Learn, como ele pode usar o Copilot para depurar sem sair do código?

A. Selecionar o método e pedir ajuda no inline chat, citando o erro
B. Esperar que o Copilot detecte o erro e o corrija por conta própria
C. Pedir ao Copilot que desative as exceções lançadas pelo projeto
D. Abrir uma issue e atribuí-la ao Copilot sem descrever o erro

R4.2 · Melhorar a produtividade do desenvolvedor com o GitHub Copilot

No VS Code, um desenvolvedor da Rede Plano seleciona uma função de cálculo de frete e quer que o Copilot gere testes de unidade para ela. Qual slash command ele usa?

A. /fix
B. /explain
C. /new
D. /tests

R4.3 · Melhorar a produtividade do desenvolvedor com o GitHub Copilot

A Seguradora Pioneira mantém um módulo legado em COBOL que precisa ser migrado para Java, e poucos desenvolvedores ainda entendem esse código. Como o Copilot pode apoiar a modernização, segundo o Microsoft Learn?

A. Convertendo o sistema inteiro com um clique no plan mode
B. Explicando o código legado e ajudando a traduzi-lo
C. Garantindo uma tradução livre de erros, dispensando revisão
D. Executando o COBOL em um emulador embutido no próprio editor

R4.4 · Melhorar a produtividade do desenvolvedor com o GitHub Copilot

Uma desenvolvedora da Hotelaria Vale quer que o Agent rode os testes xUnit do projeto e, se algum falhar, proponha e aplique correções até que passem. Segundo o Microsoft Learn, como ela deve fazer o pedido?

A. Usar ghost text no arquivo de testes até que todos fiquem verdes
B. Pedir no prompt que rode os testes, corrija as falhas e rode de novo até passar
C. Usar o Ask mode, que executa e corrige os testes sem editar arquivos
D. Pedir só /tests, que gera e também roda e corrige os testes sozinho

Respostas comentadas

R4.1. A) Selecionar o método e pedir ajuda no inline chat, citando o erro

Por quê: O Copilot Chat ajuda a analisar mensagens de erro, sugerir correções e explicar passo a passo trechos problemáticos. O inline chat mantém a interação junto ao código.

✗ B) Incorreta. O Copilot responde a pedidos; é preciso informar o erro e o contexto.
✗ C) Incorreta. Esconder exceções não resolve a causa do problema.
✗ D) Incorreta. Uma issue sem a descrição do erro não dá ao agente o contexto necessário.

> **Dica de prova:** Erro no método: selecione e pergunte no inline chat.

R4.2. D) /tests

Por quê: No agent mode, /tests pode gravar os testes direto em um arquivo de testes adequado, detectando o framework e o estilo existentes. Revise e mantenha ou desfaça as mudanças.

✗ A) Incorreta. /fix propõe correções para problemas no código selecionado.
✗ B) Incorreta. /explain explica o código.
✗ C) Incorreta. /new cria um novo projeto.

> **Dica de prova:** Testes de unidade do código selecionado: /tests.

R4.3. B) Explicando o código legado e ajudando a traduzi-lo

Por quê: Na manutenção, o Copilot ajuda a entender código legado com explicações e equivalentes modernos, sugere refatorações e auxilia a tradução entre linguagens. A validação humana continua obrigatória.

✗ A) Incorreta. O plan mode monta um plano para revisão, não converte sistemas inteiros automaticamente.
✗ C) Incorreta. Código gerado sempre precisa de revisão e testes; não há garantia de ausência de erros.
✗ D) Incorreta. O Copilot não traz emulador de COBOL; ele gera e explica código.

> **Dica de prova:** Legado: peça explicação e tradução, e revise tudo.

R4.4. B) Pedir no prompt que rode os testes, corrija as falhas e rode de novo até passar

Por quê: Quando o Agent roda os testes, ele monitora a saída, identifica falhas e tenta corrigir, o que é útil ao montar um projeto de testes novo ou após grandes mudanças.

✗ A) Incorreta. Ghost text sugere código no editor; não roda testes.

✗ C) Incorreta. O Ask mode não edita arquivos nem roda comandos.

✗ D) Incorreta. /tests gera testes; o ciclo de rodar e corrigir é pedido explicitamente no prompt.

> **Dica de prova:** Rodar e corrigir até passar: peça explicitamente ao Agent.

05

CAPÍTULO 5 · 15-20% DA PROVA

Usar o GitHub Copilot com responsabilidade

Este domínio vale de 15 a 20% e mistura duas coisas: os seis princípios de IA responsável (que a prova cobra pelo cenário, sem dizer o nome) e os riscos e mitigações do próprio Copilot descritos nos application cards. Eu preciso reconhecer na hora alucinação, overreliance, viés e injeção de prompt, e saber que o Copilot é ferramenta: quem revisa, testa, executa e responde pelo resultado sou eu. A prova adora alternativas que prometem garantia (filtro garante, teste garante, instrução garante, plano Enterprise garante); todas estão erradas. No cloud agent e no Copilot CLI caem as mitigações concretas: quem pode acionar o agente, branch próprio, workflows que aguardam Approve and run workflows, aprovação de quem pediu que não conta e diretórios confiáveis. Meu erro mais comum é confundir os princípios parecidos, como fairness com inclusiveness e transparency com accountability.

Neste capítulo

18 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Seis princípios de IA responsável

O GitHub segue os seis princípios do Responsible AI Standard da Microsoft: fairness, reliability and safety, privacy and security, inclusiveness, transparency e accountability. Eles são aplicados na prática por meio dos application cards. Lucratividade e autonomia do modelo NÃO são princípios; a supervisão humana é que é salvaguarda central.

Fairness (equidade)

A IA deve dar recomendações consistentes a pessoas em situação semelhante, sem impactos diferentes sobre grupos. Na prova aparece como testar com amostras demográficas balanceadas, treinar com dados diversos e monitorar desempenho por segmento, em contextos como crédito, emprego ou saúde.

Reliability and safety (confiabilidade e segurança)

O sistema funciona como projetado, sem variações indesejadas, responde com segurança a condições inesperadas e resiste a manipulação maliciosa. Segurança aqui significa minimizar danos não intencionais, não proteger dados (isso é privacy and security).

Privacy and security (privacidade e segurança)

Obter consentimento antes de coletar dados pessoais, coletar só o necessário, anonimizar dados pessoais e criptografá-los em trânsito e em repouso. Consentimento e minimização de dados são as palavras-chave do enunciado.

Inclusiveness (inclusão)

Todos devem conseguir acessar o sistema, independentemente de deficiência, idioma ou infraestrutura. Leitores de tela, legendas, controle por voz e funcionamento com conectividade limitada apontam para inclusão.

Transparency e accountability

Transparência é tornar o sistema compreensível e auditável: documentar dados e modelos, justificar decisões de design, ser honesto sobre limitações e manter registros para auditoria. Accountability é o princípio de que pessoas respondem pelos sistemas de IA, acompanhando o desempenho e mitigando riscos de forma contínua.

Application card

Documento oficial do GitHub que descreve capacidades, usos pretendidos, limitações e mitigações de um recurso (há cards das sugestões inline e do Copilot Chat). Operacionaliza os princípios de IA responsável e pode ser compartilhado com partes interessadas; não confunda com audit log, relatório de uso ou copilot-instructions.md.

Alucinação (hallucination)

Resposta plausível e dita com segurança, porém falsa ou inventada, como um parâmetro de biblioteca que não existe ou uma frase de PR dizendo que corrigiu um módulo que nem foi alterado. É risco conhecido dos LLMs e um dos motivos para revisar toda saída de IA.

Overreliance (dependência excessiva)

Aceitar respostas sem conferir porque os erros costumam ser sutis e difíceis de notar. Pode gerar queda de produtividade, perda de confiança, prejuízo financeiro e até danos físicos ou psicológicos; a salvaguarda é a supervisão humana.

Viés e escopo limitado

As fontes de treinamento podem conter vieses e erros que o Copilot reproduz, como preferir padrões antigos e estilos muito comuns em repositórios públicos. Linguagens pouco representadas nos dados de treinamento recebem sugestões piores (JavaScript está entre as mais bem suportadas), e as sugestões inline usam o contexto do código em edição, sem enxergar a arquitetura como um todo.

Injeção de prompt e mitigações do cloud agent

Injeção de prompt é inserir instruções escondidas para manipular o agente. Mitigações documentadas: caracteres ocultos, como comentários HTML, são filtrados antes de chegar ao agente; só quem tem permissão de escrita aciona o agente (comentário @copilot de quem não tem write não é apresentado a ele); firewall limita a internet; PRs exigem revisão humana.

Controles do PR do cloud agent

O agente só faz push no próprio branch e está sujeito às proteções de branch. Por padrão os workflows do GitHub Actions aguardam o botão Approve and run workflows, e a aprovação de quem pediu o PR ao agente não conta para as aprovações exigidas. O cloud agent não aprova nem mescla pull requests.

Rastreabilidade dos commits do agente

Commits do cloud agent têm o Copilot como autor, quem iniciou a tarefa como coautor, são assinados (aparecem como Verified) e trazem na mensagem o link para o log da sessão; a atividade também aparece no audit log. Isso existe para os administradores não perderem visibilidade do trabalho do agente.

Firewall do cloud agent

O agente tem acesso restrito à internet por padrão para reduzir o risco de vazamento de código e dados sensíveis. Há uma lista recomendada que libera repositórios de pacotes comuns; o firewall pode ser personalizado ou desabilitado, mas desligá-lo aumenta o risco de exfiltração.

Supervisão humana nas sugestões e no agent mode

Nas sugestões inline a sugestão aparece destacada e só entra no arquivo se você aceitar: nenhuma mudança sem ação deliberada. No agent mode do IDE, por padrão, você confirma cada comando de terminal antes da execução e revisa as edições. O Copilot code review, sem configuração extra, deixa um review do tipo Comment, que não conta como aprovação.

Uso responsável do Copilot CLI

Ao iniciar uma sessão a CLI pede para confirmar que você confia nos arquivos da pasta e subpastas; não inicie no diretório home, use só diretórios confiáveis. 'Yes, and approve TOOL for the rest of the running session' libera qualquer uso daquela ferramenta até o fim da sessão. Com --allow-all-tools, rode em VM ou contêiner com rede e permissões restritas ou no sandbox local ou de nuvem do Copilot.

Decisões consequentes e domínios críticos

Decisões que afetam acesso a educação, emprego, finanças, benefícios, saúde, moradia, seguros ou serviços jurídicos pedem cautela e garantia de que os afetados possam entender, contestar e corrigir dados. Agentes com ações irreversíveis pedem precauções extras, e avaliar obrigações legais e regulatórias do próprio setor cabe ao cliente.

Content filtering, red teaming e BYOK

Filtro de conteúdo examina prompts e respostas para bloquear conteúdo nocivo; conteúdo ofensivo deve ser reportado a copilot-safety@github.com e com feedback na resposta. Red teaming é o teste estruturado em que pessoas tentam provocar comportamentos inseguros ou nocivos. Com BYOK, o cliente protege a chave, arca com custos e cotas do provedor, valida a saída e cumpre os termos do provedor.

Macetes para memorizar

💡 Seis princípios

F-R-P-I-T-A: Fairness, Reliability and safety, Privacy and security, Inclusiveness, Transparency, Accountability. Lucro e autonomia do modelo ficam de fora.

💡 Transparência x responsabilização

Transparência EXPLICA o sistema; accountability RESPONDE por ele.

💡 Inclusão x equidade

Inclusão = ACESSO (todos conseguem usar). Equidade = JUSTIÇA (ninguém é tratado pior).

💡 Garantias

Filtro reduz, instrução orienta, teste ajuda: nenhum GARANTE. Se a alternativa diz 'garante', desconfie.

💡 Saída de IA

Saída do Copilot = trabalho de colega: avalie, teste e confirme, em qualquer superfície (IDE, CLI ou cloud agent).

💡 PR do cloud agent

Quem pede não aprova; agente não aprova nem mescla; CI espera Approve and run.

💡 Commits do agente

Autor Copilot, coautor humano, assinado, link da sessão.

Tabela de memorização: não confunda

Quando é...	...e quando é
Fairness (equidade) Cenário de tratar grupos de forma justa: amostras demográficas balanceadas, dados diversos, monitorar por segmento, decisões de crédito ou emprego consistentes.	**Inclusiveness (inclusão)** Cenário de acesso para todos: leitor de tela, legendas, controle por voz, idiomas, conectividade limitada.

Transparency (transparência) Documentar dados e modelos, explicar decisões de design, ser honesto sobre limitações, manter logs para auditoria.	**Accountability (responsabilização)** Pessoas ou comitê respondem pelos efeitos do sistema e acompanham desempenho e riscos continuamente.
Reliability and safety Funcionar como projetado, responder com segurança a condições inesperadas, resistir a manipulação.	**Privacy and security** Consentimento antes de coletar dados, minimização, anonimização e criptografia.
Alucinação O Copilot inventa algo que não existe (parâmetro, afirmação sem base) com total segurança.	**Viés dos dados de treinamento** O Copilot repete padrões antigos ou estilos muito comuns na internet, favorecendo certas linguagens ou estilos.
Overreliance Pessoas aceitam saída sem conferir porque o erro é sutil.	**Injeção de prompt** Alguém insere instruções maliciosas escondidas na entrada, por exemplo em comentário HTML de uma issue.
Content filtering (filtro de conteúdo) Bloquear conteúdo nocivo ou ofensivo em prompts e respostas.	**Filtro de código público** Ocultar ou referenciar sugestões que correspondem a código público; não trata de segurança, qualidade nem conformidade regulatória.
Yes (só esta vez) na CLI Você quer aprovar apenas aquela execução específica.	**Yes, and approve TOOL for the rest of the running session** Libera a ferramenta inteira (qualquer rm, por exemplo) sem novo pedido até o fim da sessão em execução; novas sessões pedem de novo.

Números que caem

Item	Valor
Peso do domínio na prova	**15 a 20%**
Princípios de IA responsável seguidos pelo GitHub	**6**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** A sugestão passou pelos filtros do Copilot, então pode ser aceita como correta e segura.
Como não cair: Filtros reduzem riscos, mas não garantem correção nem segurança. Entenda o código (peça ao Chat para explicar), revise e teste antes de usar.

⚠ **Alerta de prova:** Compilou sem erro e os testes passaram, então pode aprovar.
Como não cair: Código pode ser sintaticamente correto e inseguro, e a revisão também cobre legibilidade e manutenção. Testes gerados podem não cobrir todos os cenários; revisão e teste manual continuam.

⚠ **Alerta de prova:** Escrever 'siga a política X' nas instruções personalizadas garante conformidade, especialmente se definido na organização ou com o modelo Auto.
Como não cair: Pela natureza não determinística da IA, instruções orientam mas não garantem nem bloqueiam saídas. Controle real exige revisão, testes, checagens automáticas e políticas.

⚠ **Alerta de prova:** Com o plano Enterprise, filtro em Block ou PR abaixo de 400 linhas, o Copilot pode substituir a revisão humana.
Como não cair: Nenhum plano, filtro ou tamanho dispensa revisão. O Copilot não foi feito para substituir a expertise; o desenvolvedor segue no comando.

⚠ **Alerta de prova:** O filtro de código público serve como checagem de segurança ou de conformidade legal.
Como não cair: Ele só trata de correspondência com código público. Segurança se valida com revisão, testes, linting, code scanning e IP scanning.

⚠ **Alerta de prova:** O GitHub ou o provedor do modelo responde pelos bugs, vulnerabilidades ou comandos destrutivos sugeridos.
Como não cair: O risco é de quem usa: o usuário revisa e testa o código, e quem executa um comando sugerido no terminal decide e responde pelo resultado. O Copilot pode, sim, sugerir comandos destrutivos.

⚠ **Alerta de prova:** Quem pediu o PR ao cloud agent pode aprová-lo, já que o código foi escrito pelo agente.
Como não cair: Justamente por ter pedido o trabalho, a aprovação dele não conta para as aprovações exigidas. E o agente não aprova nem mescla PRs.

⚠ **Alerta de prova:** O CI não rodou no PR do cloud agent porque o agente não tem permissão de usar o Actions ou pausou o CI para economizar.
Como não cair: Por padrão os workflows aguardam Approve and run workflows, uma mitigação deliberada de segurança (a organização pode optar por rodar automaticamente).

⚠ **Alerta de prova:** Os commits do agente saem em nome de quem pediu, sem marca do Copilot, ou sem assinatura.
Como não cair: Autor é o Copilot, coautor é quem pediu, commits assinados (Verified) e com link para o log da sessão.

⚠ **Alerta de prova:** O agente pede confirmação ao autor da issue ou registra a injeção no audit log e segue a instrução oculta.
Como não cair: A mitigação é impedir que o conteúdo oculto chegue: caracteres escondidos, como comentários HTML, são filtrados antes do agente.

⚠ **Alerta de prova:** Aprovar rm 'para o resto da sessão' só vale para aquele arquivo temporário, ou vale também nas próximas sessões.
Como não cair: Vale para a ferramenta inteira e só na sessão em execução: qualquer rm roda sem pedir de novo até ela acabar.

⚠ **Alerta de prova:** Usar BYOK transfere a responsabilidade pela validação e conformidade ao provedor do modelo.
Como não cair: Com BYOK o cliente continua responsável: protege a chave, paga custos e cotas, valida a saída e cumpre os termos do provedor.

⚠ **Alerta de prova:** Texto sem sentido e chamadas de ferramenta estranhas no agent mode indicam conta invadida; ou basta deixar o agente terminar.
Como não cair: É comportamento probabilístico do modelo, não comprometimento da conta. Pare o agente, revise e desfaça mudanças inesperadas e reenvie o prompt.

⚠ **Alerta de prova:** Por padrão o Copilot code review deixa Approve ou Request changes.
Como não cair: Sem configuração de Copilot approvals, ele deixa um review Comment, que não conta como aprovação nem bloqueia o merge.

Como cai na prova

📝 **Como cai:** Cenário de uma empresa adotando uma prática (sem nomear o princípio) e pergunta qual princípio de IA responsável está sendo aplicado.
Como resolver: Ache a palavra-chave: acesso/deficiência é inclusão; grupos e vieses é equidade; explicar e auditar é transparência; pessoas respondendo é accountability; funcionar como projetado é confiabilidade; consentimento e minimização é privacidade. Descarte o princípio vizinho que só parece encaixar.

📝 **Como cai:** Alguém quer pular revisão porque algo 'já foi verificado' (filtro, testes verdes, plano caro, instrução escrita).
Como resolver: Marque a alternativa que mantém a revisão humana e combina testes e ferramentas automáticas. Toda opção que dispensa revisão por causa de um recurso está errada.

📝 **Como cai:** Risco descrito em linguagem comum e pedido o nome dado pelo application card.
Como resolver: Inventou algo = alucinação; aceitou sem conferir = overreliance; padrão antigo repetido = viés; instrução maliciosa escondida = injeção de prompt; coincide com repositório público = correspondência com código público.

📝 **Como cai:** Situação no cloud agent (issue maliciosa, comentário de visitante, CI parado, aprovação do solicitante, auditoria de commits).
Como resolver: Lembre o pacote de mitigações: filtro de caracteres ocultos, só write aciona, firewall, branch próprio, workflows segurados, aprovação do solicitante não conta, autoria Copilot com coautor e link da sessão.

📝 **Como cai:** Uso do Copilot CLI com aprovações amplas, home ou --allow-all-tools.
Como resolver: A resposta certa reduz exposição: diretórios confiáveis, nada de home, aprovação por sessão entendida como ampla, e --allow-all-tools só em sandbox, VM ou contêiner restrito.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
amostras balanceadas, monitorar por segmento, crédito justo	**Fairness**
leitor de tela, legendas, controle por voz, conectividade limitada	**Inclusiveness**
documentar, justificar design, honestidade sobre limitações, auditoria	**Transparency**
comitê ou pessoas respondem pelos efeitos do sistema	**Accountability**
funcionar como projetado, condições inesperadas, resistir a manipulação	**Reliability and safety**
consentimento, coletar só o necessário	**Privacy and security**
parâmetro que não existe explicado com segurança	**Alucinação**
aceitam sem conferir porque o erro é sutil	**Overreliance**

comentário HTML oculto em issue para o agente	**Injeção de prompt; caracteres ocultos são filtrados antes do agente**
documento oficial com capacidades, usos pretendidos e limitações	**Application card**
pessoas tentam provocar comportamento nocivo de propósito	**Red teaming**
resposta ofensiva no Chat	**Reportar a copilot-safety@github.com e dar feedback na resposta**
PR do cloud agent sem CI rodando	**Approve and run workflows**
sugestão de função que não entendo	**Pedir ao Copilot Chat que explique antes de decidir**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- O Copilot é ferramenta, não substituto: quem usa revisa, testa e responde pelo código e pelos comandos executados.
- Nenhum filtro, plano, instrução ou teste gerado garante correção, segurança ou conformidade.
- Cloud agent: só write aciona, conteúdo oculto filtrado, branch próprio, CI segurado, aprovação do solicitante não conta.
- Avaliar obrigações legais e regulatórias do próprio setor cabe ao cliente.
- Conteúdo ofensivo: copilot-safety@github.com e feedback na resposta.

Essenciais para a prova (checklist)

☐ Sei associar cada um dos seis princípios a um cenário prático sem ver o nome dele.

☐ Sei diferenciar alucinação, overreliance, viés dos dados e injeção de prompt.

☐ Sei o que é um application card e o que ele contém.

☐ Sei por que filtros e testes gerados não dispensam revisão humana.

☐ Sei as mitigações de segurança do cloud agent (filtro de ocultos, write, firewall, CI, aprovações, autoria).

☐ Sei o efeito de aprovar uma ferramenta para o resto da sessão no Copilot CLI e por que não iniciar no home.

☐ Sei o que o Copilot code review deixa por padrão (review Comment).

☐ Sei as responsabilidades do cliente no BYOK e em decisões consequentes.

☐ Sei o que fazer quando o agent mode sai do controle (parar, revisar, desfazer, reenviar).

Questões de revisão do capítulo

R5.1 · Usar o GitHub Copilot com responsabilidade

A Saúde Popular vai usar código gerado com o Copilot em um sistema que influencia o acesso de pacientes a tratamentos. Segundo as boas práticas do application card, o que deve ser observado em decisões consequentes como essa?

A. Delegar a decisão final ao Copilot para reduzir vieses humanos
B. Esconder dos pacientes o uso de IA para evitar contestações e atrasos
C. Liberar o uso sem revisão, porque a IA decide de forma mais objetiva e rápida
D. Ter cautela e garantir que os afetados entendam, contestem e corrijam dados

R5.2 · Usar o GitHub Copilot com responsabilidade

A Engenharia Ponte quer que seu sistema de IA funcione como projetado, responda com segurança a condições inesperadas e resista a manipulações maliciosas. Qual princípio de IA responsável descreve esse objetivo?

A. Reliability and safety (confiabilidade e segurança)
B. Privacy and security (privacidade e segurança dos dados)
C. Transparency (transparência)
D. Fairness (equidade)

R5.3 · Usar o GitHub Copilot com responsabilidade

Nas sugestões inline do editor, qual salvaguarda descrita no application card garante que nenhuma mudança entre no arquivo sem uma ação deliberada do desenvolvedor?

A. A sugestão aparece destacada e só entra no arquivo se for aceita
B. O Copilot aplica a mudança e a desfaz quando os testes falham
C. O Copilot grava a sugestão e cria um commit que pode ser revertido
D. O filtro de toxicidade aprova cada mudança antes de aplicá-la

R5.4 · Usar o GitHub Copilot com responsabilidade

O Copilot code review analisou um pull request da Têxtil Nova. Sem configurações extras de aprovação, que tipo de review o Copilot deixa e como isso afeta as aprovações exigidas pela branch?

A. Nenhum review, só uma issue listando os problemas achados
B. Um review Comment, que não conta como aprovação
C. Um review Request changes, que bloqueia o merge do PR
D. Um review Approve, que já satisfaz a regra de aprovação

Respostas comentadas

R5.1. D) Ter cautela e garantir que os afetados entendam, contestem e corrijam dados

Por quê: Decisões consequentes afetam acesso a educação, emprego, finanças, benefícios, saúde, moradia, seguros ou serviços jurídicos, ou podem causar danos físicos, psicológicos ou financeiros.

✗ A) Incorreta. O card enfatiza supervisão humana, não delegar a decisão.
✗ B) Incorreta. Esconder o uso contraria a transparência e o direito de contestação.
✗ C) Incorreta. A IA também pode errar e carregar vieses; a revisão é essencial.

> **Dica de prova:** Decisão consequente: cautela e direito de entender e contestar.

R5.2. A) Reliability and safety (confiabilidade e segurança)

Por quê: No Learn, segurança significa minimizar danos não intencionais, e confiabilidade significa funcionar como planejado, sem variações indesejadas.

✗ B) Incorreta. Privacidade e segurança tratam da proteção de dados e do consentimento.
✗ C) Incorreta. Transparência trata de tornar o sistema compreensível.
✗ D) Incorreta. Equidade trata de tratar as pessoas de forma justa.

> **Dica de prova:** Funcionar como projetado e resistir a manipulação: reliability and safety.

R5.3. A) A sugestão aparece destacada e só entra no arquivo se for aceita

Por quê: Entre as salvaguardas das sugestões inline estão o processamento controlado do contexto, filtros de conteúdo e de código, correspondência com código público e a supervisão humana: nenhuma mudança sem ação deliberada.

✗ B) Incorreta. Não há aplicação automática com desfazer por testes nas sugestões inline.
✗ C) Incorreta. As sugestões inline não criam commits; nada é aplicado sem aceite.
✗ D) Incorreta. O filtro de toxicidade trata de conteúdo nocivo, não de autorizar mudanças.

> **Dica de prova:** Inline: nada entra no arquivo sem o seu aceite.

R5.4. B) Um review Comment, que não conta como aprovação

Por quê: O papel do Copilot na revisão é consultivo: ele aponta problemas e sugere mudanças, mas a decisão final e as aprovações seguem com pessoas, salvo configuração explícita de aprovações do Copilot.

✗ A) Incorreta. A revisão aparece como review no PR, com comentários nas linhas.
✗ C) Incorreta. O Copilot não deixa Request changes por padrão.
✗ D) Incorreta. Aprovar só é possível se as Copilot approvals forem configuradas.

> **Dica de prova:** Por padrão, o Copilot comenta; não aprova nem bloqueia.

06

CAPÍTULO 6 · 25-30% DA PROVA

Usar recursos do GitHub Copilot

É o domínio mais pesado (25 a 30%) e o mais decoreba: modos do chat, atalhos, slash commands, participantes, arquivos de instruções, Copilot CLI, cloud agent, code review, Spaces, MCP, políticas, audit log e REST API. A prova pergunta muito 'qual recurso atende' (Ask, Plan, Edit ou Agent; instruções ou prompt file; Space ou prompt file) e 'onde fica' (qual arquivo, qual pasta, qual comando, qual evento). Eu erro quando confundo disponibilidade por IDE (plan e edit não estão no Visual Studio), precedência de instruções e de políticas, e comandos parecidos da CLI como /context, /usage e /compact. Também caem detalhes de planos: o Free não tem cloud agent nem resumo de PR, e o Spark está no Pro+ e no Enterprise.

Neste capítulo

18 tópicos-chave · 14 pegadinhas · 8 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Modos do chat no IDE

No VS Code o seletor oferece Ask (entender e explorar sem alterar nada), Plan (pesquisa somente leitura e plano revisável) e Agent (executa tarefas de várias etapas de forma autônoma); o Edit mode (Copilot Edits) trabalha num working set de arquivos que você escolhe, com aceite por arquivo. Edit mode existe só no VS Code e nos IDEs JetBrains; no Visual Studio (a partir da 17.14) os modos são Ask e Agent, sem Plan.

Agent mode

Indicado para tarefas complexas, com vários passos, iteração, tratamento de erros ou ferramentas externas como MCP: descobre arquivos, edita, roda comandos e testes até concluir. Está no VS Code, Visual Studio, JetBrains, Xcode e Eclipse. Comandos pedem sua confirmação; você pode redirecionar com novo prompt e rejeitar comandos no meio da tarefa. Se a opção Agent sumiu do seletor, provavelmente o administrador desabilitou por política.

Plan mode

O plan agent pesquisa o código, faz perguntas sobre requisitos ambíguos e monta um plano sem alterar código até a aprovação; depois você clica em Start Implementation ou abre o plano no editor como Markdown. No Copilot CLI, Shift+Tab alterna para o plan mode.

Atalhos de sugestões inline (VS Code, Windows)

Tab aceita a sugestão inteira, Esc rejeita, Alt+] mostra a próxima alternativa e Alt+[a anterior, Ctrl+Enter abre várias sugestões numa nova aba e Ctrl+→ aceita só a próxima palavra (aceitar a próxima linha exige atalho personalizado para editor.action.inlineSuggest.acceptNextLine). Ctrl+I abre o inline chat no editor. Next edit suggestions mostram uma seta na margem com a próxima edição prevista, navegada com Tab.

Participantes, variáveis e slash commands

Participantes (@): @terminal (shell e erros do terminal), @vscode (o próprio editor), @github (habilidades do GitHub, como busca na web com #web) e @azure (preview). Variáveis (#) no VS Code: #selection, #file, #function, #class, #comment, #block, #line, #path, #project, #sym e #codebase. Slash commands no VS Code: /explain, /fix, /tests, /fixTestFailure, /new, /clear, /help; o Visual Studio tem /doc e /optimize.

Instruções do repositório

Três tipos: de todo o repositório em .github/copilot-instructions.md; por caminho em .github/instructions/NOME.instructions.md com applyTo (glob de arquivos); e de agentes em AGENTS.md (pode haver vários, vale o mais próximo na árvore) ou um único CLAUDE.md ou GEMINI.md na raiz. Se o caminho bate e também existe o arquivo geral, os dois são usados; excludeAgent: "code-review" ou "cloud-agent" tira o arquivo de um dos dois.

Precedência e alcance das instruções

No Chat do GitHub.com todas as instruções relevantes chegam ao Copilot, com prioridade: pessoais, depois do repositório, depois da organização. Instruções da organização exigem Business ou Enterprise e valem no Chat do GitHub.com, no code review e no cloud agent, não nas sugestões inline nem no Chat dos IDEs.

Prompt files

Arquivos .prompt.md em .github/prompts: modelos reutilizáveis acionados manualmente, que aceitam variáveis de entrada, bons para tarefas focadas como gerar testes ou aplicar um checklist de revisão. Regra de ouro: instruções são sempre ativas no escopo; prompt files são sob demanda. Hooks (por exemplo preToolUse em .github/hooks) executam comandos em eventos do agente e não são prompts.

MCP (Model Context Protocol)

Padrão aberto para conectar o Copilot a ferramentas e dados externos (por exemplo um rastreador de incidentes), no IDE, na CLI e no cloud agent. Em organizações, a política MCP servers in Copilot vem desabilitada por padrão e só governa licenças Business e Enterprise. O GitHub MCP server pode ser usado remoto no VS Code sem Docker, e toolsets limitam os grupos de ferramentas.

MCP no cloud agent e no code review

O cloud agent já vem com os servidores GitHub (token de leitura restrito ao repositório) e Playwright (páginas em localhost). Outros são configurados pelo admin do repositório e valem para cloud agent e code review, mas só tools: sem resources, prompts nem servidor remoto com OAuth; no code review as tools precisam de annotations.readOnlyHint igual a true. O agente usa as tools sem pedir aprovação.

Copilot CLI: instalação e autenticação

Instala por npm (npm install -g @github/copilot, com Node.js 22 ou mais recente), WinGet, Homebrew ou script; no Windows também exige PowerShell 6 ou mais recente. No primeiro uso pede /login; em automação sem navegador use PAT fine-grained da conta pessoal com a permissão Copilot Requests na variável COPILOT_GITHUB_TOKEN. Com licença de organização, a política do Copilot CLI precisa estar habilitada.

Copilot CLI: uso interativo e programático

copilot abre a sessão interativa; copilot -p "prompt" roda um único prompt e termina (scripts). No prompt: @caminho inclui arquivo, #número referencia issue ou PR, ! roda comando de shell sem o modelo, & ou /delegate envia ao cloud agent. A CLI cria branch, workflow, faz push e abre PR em seu nome (você é o autor), usando o GitHub MCP server embutido.

Copilot CLI: sessão e contexto

/usage mostra AI credits, duração, linhas editadas e tokens por modelo; /context mostra o uso da janela; /compact resume o histórico; /clear inicia nova conversa. copilot --continue retoma a sessão local fechada mais recente e /resume escolhe uma sessão; --cloud inicia sessão em sandbox na nuvem. /add-dir libera outra pasta, /list-dirs lista as permitidas, /cwd ou /cd trocam de pasta. --deny-tool sempre vence --allow-all-tools.

Agentes e subagentes

Agentes padrão da CLI: Explore, Task, General purpose, Code review, Research e Rubber duck. Subagentes têm contexto próprio, devolvem o resultado, não criam outros subagentes e usam o mesmo modelo e ferramentas da sessão; no VS Code exigem a ferramenta runSubagent habilitada. /delegate manda a tarefa ao cloud agent, que segue mesmo com o notebook desligado (o autopilot roda local).

Copilot cloud agent

Novo nome do coding agent. Trabalha em segundo plano num ambiente baseado em GitHub Actions, num branch copilot/, e abre PR para revisão; pode ser iniciado por issue atribuída, aba Agents, Chat, IDEs, CLI, API, Slack, Teams e outras entradas. Limitações: até 59 minutos por sessão, um repositório por tarefa, um branch e um PR, só repositórios no GitHub. Sessões: steering aplicado após a chamada de ferramenta atual, Stop session preserva commits, arquivar sim, excluir não. Não está no Copilot Free.

Copilot code review

Peça escolhendo Copilot em Reviewers (via REST API: copilot-pull-request-reviewer[bot]); costuma levar menos de 30 segundos. Revisão automática por ruleset com Automatically request Copilot code review, com opções Review new pushes e Review draft pull requests; sem Review new pushes a revisão ocorre uma vez. Usa as instruções do branch head. Lock files, SVG, logs e pastas como node_modules, dist e vendor ficam fora. Fix with Copilot passa o comentário ao cloud agent; no Free só existe Review selection no VS Code.

Copilot Spaces e Spark

Spaces reúnem repositórios, PRs, issues, notas e imagens como contexto curado e compartilhável para o Chat; seguem as permissões do GitHub (cada um só vê o que já podia ver). Spaces de organização dão acesso admin, editor ou viewer; Spaces pessoais públicos são somente leitura por padrão. No IDE, acesso pelo GitHub MCP server. Spark (preview) cria aplicações a partir de linguagem natural e está no Copilot Pro+ e no Copilot Enterprise.

Políticas, audit log e REST API

Políticas e modelos da organização ficam em Settings > Copilot (Policies e Models); se a enterprise definiu, a organização não muda (só em Let organizations decide). Mesma enterprise com várias organizações: em geral vale a menos restritiva; enterprises diferentes: quase sempre a mais restritiva. Audit log: action:copilot (eventos como cfb_seat_assignment_unassigned), actor:Copilot (agentes); o audit log da enterprise retém os eventos por 180 dias e não guarda prompts enviados localmente. API: GET /orgs/{org}/copilot/billing/seats lista licenças; remover usuário deixa a licença em pending cancellation até o fim do ciclo.

Macetes para memorizar

💡 Modos do chat

Ask PERGUNTA, Plan PLANEJA, Edit EDITA o que eu escolho, Agent AGE sozinho.

💡 Disponibilidade por IDE

Visual Studio só tem Ask e Agent. Agent mode está nos cinco: VS Code, Visual Studio, JetBrains, Xcode, Eclipse.

💡 Atalhos inline

Colchete muda a sugestão (Alt+] próxima, Alt+[anterior); Ctrl+seta pega uma palavra; Ctrl+Enter abre o cardápio numa aba.

💡 Símbolos da CLI

@ arquivo, # issue, ! shell, & nuvem, / comando.

Instruções x prompt file

Instrução é SEMPRE; prompt file é QUANDO EU CHAMAR.

Precedência de instruções no GitHub.com

Eu > repo > org (do mais pessoal para o mais geral).

Cloud agent

Um repo, um branch, um PR, só GitHub, 59 minutos.

MCP no cloud agent

Só TOOLS; padrão vem GitHub + Playwright.

Tabela de memorização: não confunda

Quando é...	...e quando é
Ask mode Só entender ou explorar o código, sem alterar nenhum arquivo.	**Agent mode** Tarefa de várias etapas: descobrir arquivos, editar, rodar testes e corrigir até concluir.
Plan mode Quer um plano revisável, com perguntas sobre requisitos, antes de qualquer mudança no código.	**Edit mode** Você escolhe os arquivos do working set e aceita ou descarta as edições de cada arquivo, com controle das requisições.
Instruções personalizadas (copilot-instructions.md, .instructions.md) Padrões e convenções aplicados automaticamente em todo pedido do escopo.	**Prompt file (.prompt.md)** Tarefa reutilizável acionada sob demanda, com variáveis, como um checklist de revisão de segurança.
Agent mode no IDE Trabalho local, interativo, no seu editor, com você confirmando comandos.	**Copilot cloud agent** Trabalho em segundo plano no GitHub, em branch copilot/, terminando num PR para revisão humana.
Copilot Spaces Coleção curada de fontes (repos, PRs, issues, notas, imagens) compartilhada com o time.	**Prompt files ou Copilot Memory** Prompt file é modelo de prompt sob demanda; Memory guarda fatos que o Copilot deduz trabalhando, não uma curadoria do time.
Várias organizações da mesma enterprise Em geral vale a configuração menos restritiva, salvo exceções.	**Licenças de enterprises diferentes** Quase sempre vale a mais restritiva.
/context Ver quanto da janela de contexto já foi usado (por categoria).	**/usage** Ver AI credits, duração da sessão, linhas editadas e tokens por modelo.

Números que caem

Item	Valor

Peso do domínio na prova	**25 a 30%**
Duração máxima de uma sessão do cloud agent	**59 minutos, sem extensão (pode ser menor com timeout-minutes no copilot-setup-steps.yml)**
Arquivo fora do resumo de PR	**Mais de 400 adições e exclusões somadas**
Histórico do Chat no GitHub.com	**Até 100 conversas; mensagens mantidas por 28 dias**
Retenção do audit log da enterprise	**180 dias**
Tempo típico do Copilot code review	**Menos de 30 segundos**
Node.js para instalar a CLI por npm	**22 ou mais recente (Windows: PowerShell 6 ou mais recente)**
Limite de completions do Copilot Free	**2.000 por mês**
Visual Studio com Copilot embutido	**17.10 ou mais recente (17.8 e 17.9 exigiam instalar extensões)**
Visual Studio com modos Ask e Agent	**17.14 ou mais recente**
Escopos de PAT classic para ler licenças	**manage_billing:copilot ou read:org (adicionar ou remover: manage_billing:copilot ou admin:org)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** O plan mode do Visual Studio foi trocado pelo edit mode, ou exige o plano Max.
Como não cair: No Visual Studio os modos são Ask e Agent: não há plan nem edit mode. Edit mode existe no VS Code e nos JetBrains. A ausência no Visual Studio não tem relação com o plano contratado.

⚠ **Alerta de prova:** Colocar um .prompt.md com applyTo para escopo por caminho, ou uma frase no copilot-instructions.md dizendo 'só para TypeScript'.
Como não cair: Escopo por caminho é .github/instructions/NOME.instructions.md com applyTo. Prompt files são sob demanda; o arquivo geral vale para todo pedido.

⚠ **Alerta de prova:** As instruções da organização têm a maior prioridade, ou só o conjunto de maior prioridade chega ao Copilot.
Como não cair: Todas as relevantes chegam; a prioridade é pessoal, depois repositório, depois organização.

⚠ **Alerta de prova:** Instruções da organização valem nas sugestões inline e no Chat do VS Code.
Como não cair: Valem no Chat do GitHub.com, no code review e no cloud agent.

⚠ **Alerta de prova:** AGENTS.md da raiz prevalece por ser o principal, ou só um AGENTS.md é aceito.
Como não cair: Vários AGENTS.md são permitidos e vale o mais próximo do arquivo em trabalho.

⚠ **Alerta de prova:** O code review usa as instruções do branch base até o merge.
Como não cair: Usa as do branch head, já com as mudanças do PR.

⚠ **Alerta de prova:** Revisão automática se configura com workflow que comenta @copilot ou com CODEOWNERS.
Como não cair: @copilot em comentário aciona o cloud agent. Revisão automática é ruleset com Automatically request Copilot code review.

⚠ **Alerta de prova:** Para MCP no cloud agent, tudo é aceito se o firewall estiver desligado, inclusive resources, prompts e OAuth.
Como não cair: Só tools; sem resources, prompts nem servidor remoto com OAuth, independentemente do firewall.

⚠ **Alerta de prova:** MCP na organização não funciona porque falta AGENTS.md, exclusão de conteúdo ou Pro+ individual.
Como não cair: Verifique a política MCP servers in Copilot, desabilitada por padrão para Business e Enterprise.

⚠ **Alerta de prova:** O Copilot CLI se instala com npm install -g copilot-cli e Node 16, ou com gh extension install.
Como não cair: Pacote @github/copilot e Node.js 22 ou mais recente (ou WinGet, Homebrew, script).

⚠ **Alerta de prova:** Com --allow-all-tools e --deny-tool='shell(rm)', o rm é liberado ou passa a pedir aprovação.
Como não cair: --deny-tool tem precedência: o rm fica bloqueado.

⚠ **Alerta de prova:** Uma mensagem de steering interrompe a ferramenta em execução; sessões do cloud agent podem ser excluídas.
Como não cair: Steering é aplicado depois da chamada de ferramenta atual; sessões são arquivadas, não excluídas; Stop session mantém os commits.

⚠ **Alerta de prova:** Removido pela API, o usuário perde o acesso na hora; ou a cobrança só começa no primeiro uso.
Como não cair: Remoção deixa a licença em pending cancellation até o fim do ciclo; a cobrança começa ao conceder o acesso, mesmo sem uso.

⚠ **Alerta de prova:** O Free tem cloud agent em repositórios públicos ou com limite de 2.000 tarefas.
Como não cair: O Free não inclui cloud agent nem resumo de PR; o limite de 2.000 do Free é de completions por mês.

Como cai na prova

📝 **Como cai:** Cenário com objetivo do usuário e pedido do modo ou recurso mais adequado (Ask, Plan, Edit, Agent, cloud agent, inline chat).
Como resolver: Pergunte: vai alterar arquivos? Quem escolhe os arquivos? Precisa rodar comandos? Precisa seguir sem mim? Sem alterar = Ask; plano antes = Plan; eu escolho = Edit; autônomo local = Agent; segundo plano no GitHub = cloud agent.

📝 **Como cai:** Pergunta de 'onde fica' ou 'qual arquivo' (instruções, prompt files, custom agents, configuração MCP da CLI).
Como resolver: Repasse o mapa: .github/copilot-instructions.md, .github/instructions/*.instructions.md, .github/prompts/*.prompt.md, .github/agents (perfis de custom agents), AGENTS.md, ~/.copilot/mcp-config.json, .vscode/mcp.json (VS Code).

📝 **Como cai:** Comando da CLI ou atalho do IDE com alternativas de comandos reais com outra função.
Como resolver: Todas as alternativas costumam existir; elimine pela função exata de cada uma (/context mede, /compact resume, /usage contabiliza, /clear recomeça, --continue retoma a última, --cloud vai à sandbox).

📝 **Como cai:** Recurso não aparece para um usuário com licença da empresa (CLI, MCP, Agent, preview).
Como resolver: A primeira suspeita é política da organização ou enterprise, não plano individual, AGENTS.md ou exclusão de conteúdo.

📝 **Como cai:** Escolha DUAS ou TRÊS sobre limites ou regras (cloud agent, Spaces, audit log, retomada da CLI).
Como resolver: Marque só afirmações absolutamente documentadas e desconfie de números alterados (7 anos em vez de 180 dias, 24 horas em vez de 59 minutos).

📝 **Como cai:** Tabela de planos: qual plano inclui o recurso.
Como resolver: Free: sem cloud agent, sem resumo de PR, code review só Review selection no VS Code. Spark: Pro+ e Enterprise. Exclusão de conteúdo e instruções da organização: Business e Enterprise.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em

entender o código sem alterar nenhum arquivo	**Ask mode**
plano revisável e perguntas antes do código	**Plan mode (VS Code) ou Shift+Tab (CLI)**
arquivos escolhidos por mim e aceite por arquivo	**Edit mode com working set**
instrução só para certos arquivos	**.instructions.md com applyTo**
prompt reutilizável sob demanda com variáveis	**.prompt.md em .github/prompts**
ferramenta ou sistema externo dentro do Copilot	**Servidor MCP**
prompt único em script, sem sessão interativa	**copilot -p**
tarefa que continua com o notebook desligado, a partir da CLI	**/delegate (cloud agent)**
contexto curado e compartilhado com o time	**Copilot Spaces**
revisão automática em todos os PRs da branch principal	**Ruleset com Automatically request Copilot code review**
comentário do review para o Copilot implementar	**Fix with Copilot**
atividade dos agentes no audit log	**actor:Copilot**
listar licenças com última atividade	**GET /orgs/{org}/copilot/billing/seats**
organização quer liberar o que a enterprise bloqueou	**Não consegue; só em Let organizations decide**

🎯 Se você lembrar de só uma coisa deste capítulo...

- Visual Studio: só Ask e Agent. Edit mode: VS Code e JetBrains.
- Instruções sempre ativas; prompt files sob demanda; escopo por caminho é .instructions.md com applyTo.
- Cloud agent: 59 minutos, um repo, branch copilot/, PR para revisão humana; não está no Free.
- Política MCP servers in Copilot vem desabilitada; no cloud agent só tools.
- Enterprise decidiu, a organização não muda; mesma enterprise = menos restritiva, enterprises diferentes = mais restritiva.

Essenciais para a prova (checklist)

- ☐ Sei escolher entre Ask, Plan, Edit, Agent e cloud agent a partir do cenário.
- ☐ Sei os atalhos de sugestões inline e do inline chat no VS Code.
- ☐ Sei os participantes @, as variáveis # e os slash commands do VS Code e do Visual Studio.

- ☐ Sei os locais e regras dos arquivos de instruções, prompt files e AGENTS.md, incluindo applyTo e excludeAgent.
- ☐ Sei instalar, autenticar e usar o Copilot CLI (interativo, -p, @, #, !, &, /delegate, retomada de sessão).
- ☐ Sei as limitações e o fluxo do cloud agent e como gerenciar suas sessões.
- ☐ Sei configurar o Copilot code review (Reviewers, ruleset, Review new pushes, Fix with Copilot, arquivos excluídos).
- ☐ Sei as regras de MCP (política, servidores padrão do cloud agent, só tools, configuração na CLI).
- ☐ Sei as regras de Spaces e em quais planos estão Spark, cloud agent e resumo de PR.
- ☐ Sei hierarquia de políticas, buscas do audit log e endpoints da REST API de licenças.

Questões de revisão do capítulo

R6.1 · Usar recursos do GitHub Copilot

O owner da organização Vértice precisa descobrir, pelo audit log, quando a licença do Copilot de um usuário foi desatribuída, sabendo que ele perderá o acesso ao fim do ciclo. Qual evento deve procurar?

A. copilot.cfb_org_settings_changed
B. copilot.cfb_seat_assignment_unassigned
C. copilot.content_exclusion_changed
D. copilot.swe_agent_repo_disabled

R6.2 · Usar recursos do GitHub Copilot

O owner da enterprise da Holding Sertão quer ver, no audit log, os registros de atividade dos agentes do Copilot no GitHub. Qual termo de busca a documentação indica?

A. user:agent
B. actor:Copilot
C. repo:copilot-logs
D. action:copilot.prompt_sent

R6.3 · Usar recursos do GitHub Copilot

Um desenvolvedor da Gestão Ágil quer usar o GitHub MCP server no Copilot Chat do VS Code, mas não quer instalar Docker nem manter um servidor rodando na própria máquina. O que a documentação do GitHub indica?

A. Contratar o Copilot Enterprise para liberar o servidor local
B. Usar o servidor no Copilot CLI, já que o VS Code não o aceita
C. Rodar o servidor em Docker, única forma documentada
D. Usar o servidor remoto do GitHub MCP direto no VS Code

R6.4 · Usar recursos do GitHub Copilot

No Copilot Chat do VS Code, um desenvolvedor da Turismo Brisa quer que o Copilot pesquise na web qual é a versão LTS mais recente do Node.js. Como ele pede essa busca de forma explícita?

A. Usar /explain com a variável #codebase no chat
B. Usar @vscode com a variável #file na pergunta
C. Usar @terminal com a variável #selection
D. Usar @github com a variável #web na pergunta

Respostas comentadas

R6.1. B) copilot.cfb_seat_assignment_unassigned

Por quê: O audit log da organização tem eventos com prefixo copilot., como cfb_seat_added, cfb_seat_assignment_unassigned, cfb_seat_cancelled e content_exclusion_changed. A busca action:copilot lista todos.

✗ A) Incorreta. Esse evento registra mudanças nas configurações de recursos do Copilot na organização.
✗ C) Incorreta. Esse evento registra mudanças nos caminhos excluídos do Copilot.
✗ D) Incorreta. Esse evento registra repositórios desabilitados para o agente de codificação do Copilot.

> **Dica de prova:** Licença desatribuída: cfb_seat_assignment_unassigned.

R6.2. B) actor:Copilot

Por quê: Para eventos do plano do Copilot, use action:copilot; para um evento específico, por exemplo action:copilot.cfb_seat_assignment_created. Atividade de agentes: actor:Copilot.

✗ A) Incorreta. Esse não é o termo documentado.
✗ C) Incorreta. Não existe repositório de logs do Copilot com esse nome.
✗ D) Incorreta. O audit log não registra prompts enviados; esse evento não existe.

> **Dica de prova:** Atividade dos agentes no audit log: actor:Copilot.

R6.3. D) Usar o servidor remoto do GitHub MCP direto no VS Code

Por quê: O GitHub MCP server permite automatizar tarefas, conectar ferramentas e invocar recursos do GitHub. Toolsets permitem habilitar só os grupos de ferramentas necessários, o que melhora a precisão e a segurança.

✗ A) Incorreta. O acesso ao servidor não depende do plano Enterprise.
✗ B) Incorreta. O VS Code suporta MCP; a CLI apenas já vem com o GitHub MCP server configurado.
✗ C) Incorreta. Rodar localmente é possível, mas não é a única forma; o acesso remoto dispensa Docker.

> **Dica de prova:** GitHub MCP sem setup local: servidor remoto no VS Code.

R6.4. D) Usar @github com a variável #web na pergunta

Por quê: A busca na web usa o Bing e, em organizações, depende de o administrador permitir. Também é possível pedir em linguagem natural, mas #web força o uso da habilidade de busca.

✗ A) Incorreta. /explain explica código; não faz busca na web.

✗ B) Incorreta. O @vscode responde sobre o editor; #file anexa um arquivo.

✗ C) Incorreta. O @terminal trata do shell do VS Code; #selection anexa o texto selecionado.

> **Dica de prova:** Busca na web no VS Code: @github com #web.

DIA DA PROVA

Estratégia de prova

O GH-300 é uma prova supervisionada de 100 minutos, com questões de múltipla escolha e de múltipla resposta (Escolha DUAS, Escolha TRÊS) e possíveis componentes interativos; a aprovação é com 700 de 1000. Quase todo enunciado é um cenário curto de uma empresa e pergunta qual recurso, comando, arquivo, política ou prática a documentação indica. Os distratores costumam ser recursos reais do Copilot com outra função (prompt file no lugar de instruções, /compact no lugar de /context, cloud agent no lugar de agent mode) ou promessas de garantia e números alterados. O peso maior está em recursos do Copilot (25 a 30%), seguido de IA responsável (15 a 20%); os outros quatro domínios valem de 10 a 15% cada. A maioria das questões cobre recursos em disponibilidade geral, mas pode haver recursos em preview de uso comum, e nomes antigos aparecem só como distrator.

Como responder

✓ Leia a última frase primeiro para saber se pedem recurso, comando, arquivo, política ou princípio, e só então leia o cenário procurando as palavras-chave (sem alterar arquivos, sob demanda, segundo plano, licença da organização).

✓ Em Escolha DUAS ou TRÊS, avalie cada alternativa como verdadeiro ou falso de forma independente; marque exatamente a quantidade pedida.

✓ Quando todas as alternativas são recursos reais, elimine pela função exata de cada um; o erro está em usar uma ferramenta certa para o problema errado.

✓ Desconfie de alternativas com garantia absoluta (garante, nunca, sempre, dispensa revisão) e de números trocados: 59 minutos, 400 linhas, 180 dias, 30 minutos, 100 conversas e 28 dias são os valores documentados.

✓ Se um recurso não aparece para quem tem licença Business ou Enterprise, pense primeiro em política da organização ou enterprise, não em plano individual.

✓ Confira a superfície do enunciado (VS Code, Visual Studio, JetBrains, CLI, GitHub.com, cloud agent): disponibilidade, comandos e comportamento mudam por superfície.

✓ Use os nomes atuais: Copilot cloud agent (antigo coding agent), AI credits no lugar de premium requests nos planos atuais, e lembre que as Copilot Extensions baseadas em GitHub App foram aposentadas.

✓ Não há acesso ao Microsoft Learn durante exames do GitHub: marque as dúvidas para revisão e administre o tempo dos 100 minutos sem contar com consulta.

Erros que derrubam candidatos

✗ Confundir modos do chat e sua disponibilidade por IDE (achar que o Visual Studio tem plan ou edit mode).

✗ Trocar instruções personalizadas (sempre ativas) por prompt files (sob demanda) e errar a precedência pessoal > repositório > organização.

✗ Errar a ordem do fluxo do prompt: contexto, proxy, filtro de toxicidade, LLM, pós-processamento, entrega.

✗ Achar que filtros, testes gerados, instruções ou um plano mais caro dispensam revisão humana.

✗ Esquecer as limitações da exclusão de conteúdo (Edit e Agent no VS Code, symlinks, FS remoto, dados semânticos) e a propagação de até 30 minutos.

✗ Confundir princípios vizinhos de IA responsável, como fairness com inclusiveness e transparency com accountability.

✗ Misturar comandos parecidos da CLI e do Chat: /context, /usage, /compact, /clear; /tests, /setupTests, /fixTestFailure.

Na semana da prova

- ☐ Revise os números documentados de cada capítulo e as listas de disponibilidade por plano e por IDE.
- ☐ Refaça em voz alta a ordem do fluxo do prompt e os seis princípios de IA responsável com um exemplo prático de cada.
- ☐ Monte de memória o mapa de arquivos e pastas: copilot-instructions.md, instructions, prompts, agents, AGENTS.md, mcp-config.json.
- ☐ Revise os comandos da CLI (/usage, /context, /compact, /resume, /add-dir, /delegate, -p, --continue, --deny-tool) e os slash commands do Chat.
- ☐ Releia as pegadinhas de exclusão de conteúdo e de filtro de código público, que concentram detalhes finos.
- ☐ Na véspera, confira os requisitos da prova supervisionada e descanse; nada de conteúdo novo.

REVISÃO FINAL

Folha de cola

Leia na véspera e na manhã da prova: os gatilhos e as pegadinhas mais importantes de cada domínio.

1. Aplicar engenharia de prompt e criação de contexto

‣ vários exemplos de entrada e saída → **Few-shot**

‣ função existente usada como modelo → **One-shot**

‣ só o comentário, sem exemplos → **Zero-shot**

‣ aja como especialista em segurança ou desempenho → **Role prompting**

‣ API, testes e deploy no mesmo pedido → **Quebra o Single**

⚠ Início e fim recebem peso extra; coloque o requisito crítico no começo e não repita.

⚠ Objetivo geral primeiro, depois os requisitos específicos.

⚠ Ênfase visual não é orientação documentada; escreva a regra direta, como 'retorne o resultado como JSON'.

2. Configurar privacidade, exclusões de conteúdo e proteções

‣ ícone do Copilot com traço diagonal → **Arquivo sob exclusão de conteúdo**

‣ excluir .env em qualquer repositório e fora do Git → **Chave "*" na exclusão da organização**

‣ caixas cinza não editáveis na exclusão do repositório → **Regras herdadas da organização ou enterprise**

‣ regra nova não aplicada no VS Code aberto desde cedo → **Até 30 minutos ou Developer: Reload Window**

‣ agent mode acessou arquivo excluído → **Exclusão não cobre Edit e Agent no VS Code**

⚠ Só Business e Enterprise, configurada por admins; preferência por linguagem é configuração do editor.

⚠ Só afeta o uso pelo Copilot; permissões e arquivos continuam iguais.

⚠ Editam admin do repositório, owner da organização e owner da enterprise; Maintain só lê.

3. Entender a arquitetura e os dados do GitHub Copilot

‣ código antes e depois do cursor ao mesmo tempo → **Fill-in-the-Middle (FIM)**

‣ tenant Azure do GitHub, barrar hacking do prompt → **Proxy na entrada**

‣ ódio, conteúdo inapropriado, nomes e endereços → **Filtro de toxicidade**

‣ XSS, SQL injection, checagem final → **Pós-processamento pelo proxy**

‣ partes treináveis menores, modelo original mantido → **LoRA**

⚠ Contexto, proxy, toxicidade, LLM, pós-processamento, entrega. Nada é gerado antes do prompt filtrado e nada é entregue antes das checagens.

⚠ Primeiro vem o envio por HTTPS e a coleta de contexto; o proxy é a segunda etapa.

⚠ É truncada ou descartada antes de chegar ao usuário.

4. Melhorar a produtividade do desenvolvedor com o GitHub Copilot

‣ projeto sem framework de testes → **/setupTests**

‣ teste falhando no Test Explorer → **Fix Test Failure ou /fixTestFailure**

‣ arquivo de testes já com casos no padrão → **Ghost text a partir de comentário**

‣ estratégia de testes com várias classes e convenções → **Plan, depois Agent**

‣ rodar, corrigir e rodar de novo até passar → **Pedir explicitamente ao Agent**

⚠ /tests gera testes; o ciclo rodar, corrigir e rodar de novo deve ser pedido explicitamente ao Agent.

⚠ Ask não edita arquivos nem roda comandos; serve para analisar antes.

⚠ É só para corrigir teste que falha. Testes novos: Agent ou /tests; cenários: Ask.

5. Usar o GitHub Copilot com responsabilidade

‣ amostras balanceadas, monitorar por segmento, crédito justo → **Fairness**

‣ leitor de tela, legendas, controle por voz, conectividade limitada → **Inclusiveness**

‣ documentar, justificar design, honestidade sobre limitações, auditoria → **Transparency**

‣ comitê ou pessoas respondem pelos efeitos do sistema → **Accountability**

‣ funcionar como projetado, condições inesperadas, resistir a manipulação → **Reliability and safety**

⚠ Filtros reduzem riscos, mas não garantem correção nem segurança. Entenda o código (peça ao Chat para explicar), revise e teste antes de usar.

⚠ Código pode ser sintaticamente correto e inseguro, e a revisão também cobre legibilidade e manutenção. Testes gerados podem não cobrir todos os cenários; revisão e teste manual continuam.

⚠ Pela natureza não determinística da IA, instruções orientam mas não garantem nem bloqueiam saídas. Controle real exige revisão, testes, checagens automáticas e políticas.

6. Usar recursos do GitHub Copilot

‣ entender o código sem alterar nenhum arquivo → **Ask mode**

‣ plano revisável e perguntas antes do código → **Plan mode (VS Code) ou Shift+Tab (CLI)**

‣ arquivos escolhidos por mim e aceite por arquivo → **Edit mode com working set**

‣ instrução só para certos arquivos → **.instructions.md com applyTo**

‣ prompt reutilizável sob demanda com variáveis → **.prompt.md em .github/prompts**

⚠ No Visual Studio os modos são Ask e Agent: não há plan nem edit mode. Edit mode existe no VS Code e nos JetBrains. A ausência no Visual Studio não tem relação com o plano contratado.

⚠ Escopo por caminho é .github/instructions/NOME.instructions.md com applyTo. Prompt files são sob demanda; o arquivo geral vale para todo pedido.

⚠ Todas as relevantes chegam; a prioridade é pessoal, depois repositório, depois organização.

GABARITO

Respostas do diagnóstico

D1. A) O contexto, como o arquivo atual e o histórico → revise o capítulo 1

Por quê: Como o contexto influencia a resposta, vale cuidar dele: abrir arquivos relevantes, anexar itens com @ ou #, usar instruções personalizadas e manter o histórico enxuto.

✗ B) Incorreta. E-mails e calendário não fazem parte do contexto do Copilot.

✗ C) Incorreta. Esse histórico não é o contexto usado para responder ao seu pedido.

✗ D) Incorreta. O Copilot combina o prompt com contexto adicional.

> **Dica de prova:** Resposta = prompt + contexto.

D2. A) Excluir esse pedido da conversa atual → revise o capítulo 1

Por quê: Guiar o Copilot inclui dar contexto útil: no IDE, abrir arquivos relevantes; no Chat, remover pedidos inúteis ou começar nova conversa.

✗ B) Incorreta. O pedido não muda o histórico da conversa.

✗ C) Incorreta. Ênfase não remove o contexto que atrapalha.

✗ D) Incorreta. Trocar de IDE não é a orientação documentada.

> **Dica de prova:** Pedido antigo atrapalhando: exclua-o da conversa.

D3. D) No início, que recebe peso extra, assim como o final → revise o capítulo 1

Por quê: Comece geral e depois especifique, com os requisitos importantes primeiro. Instruções no início e no fim do prompt recebem mais peso.

✗ A) Incorreta. Requisitos enviados depois da resposta não orientam a resposta já gerada.

✗ B) Incorreta. A documentação destaca o começo e o fim, não o meio.

✗ C) Incorreta. Repetir em todas as frases deixa o prompt confuso e longo.

> **Dica de prova:** Requisito crítico vai no começo do prompt.

D4. D) Developer: Chat Diagnostics → revise o capítulo 2

Por quê: Para certificados personalizados e problemas de rede, a documentação traz um guia específico. Os logs ficam em View > Output, GitHub Copilot.

✗ A) Incorreta. Set Log Level ajusta o detalhe dos logs, mas não faz o diagnóstico de alcance.

✗ B) Incorreta. Esse comando abre os atalhos de teclado.

✗ C) Incorreta. Reload Window recarrega o VS Code, útil para exclusões de conteúdo.

> **Dica de prova:** Rede e proxy no VS Code: Developer: Chat Diagnostics.

D5. D) Abrir só o arquivo excluído, anexá-lo ao prompt e pedir explain this file → revise o capítulo 2

Por quê: Para testar sugestões inline, edite um arquivo excluído e confirme que nada é sugerido. Para o Chat, feche os outros arquivos, anexe o excluído e peça explain this file: ele não deve ser usado nem listado nas referências.

✗ A) Incorreta. A exclusão é configurada nas settings do repositório, da organização ou da enterprise, não por arquivo de ignore no repositório.

✗ B) Incorreta. O teste documentado é feito no Copilot Chat do IDE, não com a CLI listando regras.

✗ C) Incorreta. O /context mostra o uso da janela de contexto; não é o teste de exclusão descrito.

> **Dica de prova:** Teste de exclusão no Chat: explain this file com o arquivo excluído.

D6. D) Aguardar e tentar de novo, já que o limite é temporário → revise o capítulo 2

Por quê: Rate limits existem por capacidade, picos de uso, equidade e mitigação de abuso. Se o problema se repetir, revise o padrão de uso, considere outro plano ou contate o suporte.

✗ A) Incorreta. Trocar de conta não é a orientação documentada.

✗ B) Incorreta. Reinstalar não muda o limite aplicado pelo serviço.

✗ C) Incorreta. Exclusão de conteúdo não tem relação com rate limit.

> **Dica de prova:** Rate limit: espere e tente de novo.

D7. A) Entrada processada, análise do modelo, geração, formatação → revise o capítulo 3

Por quê: Na formatação, a resposta recebe destaque de sintaxe e indentação e pode trazer links para o contexto usado, como arquivos, issues ou documentação.

✗ B) Incorreta. A formatação é a última etapa, não a primeira.

✗ C) Incorreta. Não há geração antes de processar a entrada.

✗ D) Incorreta. O modelo só analisa depois que a entrada é processada e combinada com contexto.

> **Dica de prova:** Chat: entrada, modelo, geração, formatação.

D8. A) Contexto e prompt, proxy, filtro de toxicidade, LLM, pós-processamento, entrega → revise o capítulo 3

Por quê: Depois da entrega, o processo se repete a cada novo prompt. Só respostas que passam em todos os filtros chegam ao usuário.

✗ B) Incorreta. O proxy e o filtro de toxicidade atuam antes do modelo, e nada é entregue antes do pós-processamento.

✗ C) Incorreta. O modelo não gera nada antes de receber o prompt montado e filtrado.

✗ D) Incorreta. O proxy só atua depois que o contexto é coletado e o prompt montado, e o pós-processamento vem antes da entrega.

> **Dica de prova:** Ordem: contexto, proxy, toxicidade, LLM, pós-processamento, entrega.

D9. A) Não; dados do Business e do Enterprise não treinam modelos, conforme o Data Protection Agreement → revise o capítulo 3

Por quê: Para Copilot Business e Copilot Enterprise, os dados ficam protegidos pelo Data Protection Agreement e não são usados para treinar modelos. Já nos planos Free, Pro, Pro+ e Max, desde 24 de abril de 2026 o GitHub pode usar as interações para treinamento, com opção de recusa nas configurações.

✗ B) Incorreta. O prompt com contexto é enviado ao serviço do Copilot para gerar a sugestão; o motivo correto é a proteção contratual.

✗ C) Incorreta. Não há essa exceção para trechos aceitos nos planos Business e Enterprise.

✗ D) Incorreta. A opção Allow GitHub to use my data for AI model training existe para planos individuais e nem aparece para quem tem licença Business ou Enterprise.

> **Dica de prova:** Business e Enterprise: DPA, sem treinamento com os seus dados.

D10. C) Gerar mocks da API, fixtures e funções auxiliares de teste → revise o capítulo 4

Por quê: Entre os usos avançados de automação, o Learn lista configurar frameworks de teste com mock data, fixtures e helpers. Pedir que o Copilot atue como especialista em testes também tende a produzir mocks e cenários de erro.

✗ A) Incorreta. Copiar dados de produção traz risco de privacidade; o Copilot pode gerar dados de exemplo.

✗ B) Incorreta. Chamar a API real torna os testes lentos e frágeis e pode gerar cobranças; o recomendado é simular a dependência.

✗ D) Incorreta. Pular os testes reduz a cobertura justamente em uma parte crítica.

> **Dica de prova:** Dependência externa nos testes: mocks e fixtures.

D11. D) Com ajuda dentro do editor, sugerindo chamadas e parâmetros → revise o capítulo 4

Por quê: Menos troca de contexto significa mais foco na solução de problemas complexos e entregas mais rápidas.

✗ A) Incorreta. Não há resumo diário por e-mail.

✗ B) Incorreta. Abrir o navegador é justamente a troca de contexto a evitar.

✗ C) Incorreta. O Copilot não bloqueia outros aplicativos.

> **Dica de prova:** Menos alternância: ajuda dentro do editor.

D12. A) Selecionando as linhas ou os arquivos específicos antes → revise o capítulo 4

Por quê: A geração de mensagens de commit sugere título e descrição, que podem ser regenerados antes de finalizar. A mensagem não se atualiza sozinha após novas mudanças.

✗ B) Incorreta. Tradução não é a orientação documentada.

✗ C) Incorreta. A mensagem é gerada a partir das mudanças; sem mudanças não há o que descrever.

✗ D) Incorreta. Revisar e editar antes de confirmar é obrigatório nas boas práticas.

> **Dica de prova:** Commit no Desktop: selecione as linhas certas antes de gerar.

Agora, treine no formato real

Faça as 6 provas completas do curso GH-300: 300 questões inéditas, tempo do exame, explicação de cada alternativa e link da documentação para revisar.

Ver as 6 provas completas e o cupom do mês →

Até o dia da prova, uma revisão por dia no seu feed

Instagram @strecht.simulados

Um post de prova por dia: questão comentada, pegadinhas e o "não confunda" de cada certificação.

Seguir no Instagram →

YouTube Strecht Simulados

Shorts diários com pegadinhas de prova e, aos sábados, um simulado comentado.

Inscrever-se no canal →

Material independente de preparação, sem afiliação ou endosso de GitHub. As questões são inéditas e não reproduzem a prova real.