PLAYBOOK DE IMPLEMENTAÇÃO · USAGE-BASED BILLING · GITHUB COPILOT

Implementação das sete recomendações, com tudo pronto.

O companheiro de implementação do runbook: cada recomendação com o porquê, a metáfora, o impacto esperado, os passos exatos e o artefato completo pronto para colar. O apêndice traz todos os arquivos, do copilot-instructions.md à regra de roteamento, e o resumo de uma página para o time.

Escopo: 7 recomendações, artefatos completos, sequência e resumo
Atualizado: 2026-06-10 · **Base:** documentação oficial validada

Paula Silva
Software Global Black Belt
paulasilva@microsoft.com

Sumário

O que tem aqui, e onde.

FUNDAMENTOS

Como usar este playbook

A divisão de trabalho com o runbook

O **runbook** é o guia extenso: explica cada conceito, cada tela e cada erro comum, com os links oficiais. Este **playbook** é a mão na massa: o caminho mais curto da decisão ao commit, com os artefatos completos. Use o playbook para implementar; volte ao runbook quando precisar do detalhe ou da base de evidência.

Papéis

Cada recomendação traz o selo de quem executa: DEV para o desenvolvedor no editor (VS Code como referência) e ADMIN para o owner da organização ou enterprise no plano de controle (GitHub.com). R1, R5 e R6 são do desenvolvedor; R2, R3 e R4 são compartilhadas; R7 é do admin.

> ANTES DE COLAR QUALQUER ARTEFATO
>
> Os arquivos do apêndice são pontos de partida validados contra a documentação oficial, não dogma. Troque os placeholders entre <> pela sua stack, revise com o time como qualquer mudança de código, e confirme os preços e nomes de modelo na tabela viva antes de citar em proposta.

O princípio que organiza tudo

Sob o Usage-Based Billing, custo = modelo × tokens, e 1 AI Credit = US$ 0,01. Toda recomendação ataca um dos dois fatores. A ordem das sete é a ordem de retorno: saída primeiro (a classe de token mais cara), conta depois, input em seguida, governança para tornar permanente. As bandas de impacto são faixas de planejamento com fonte declarada, nunca promessa; elas se compõem, não se somam.

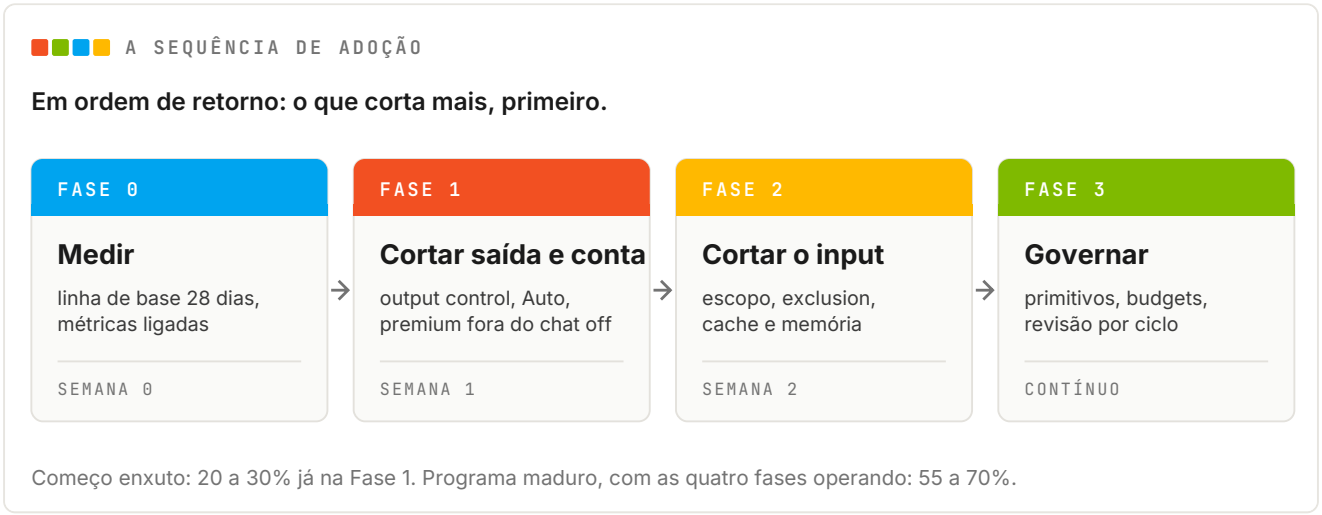
FUNDAMENTOS

A sequência de adoção

As quatro fases

Implemente na ordem do diagrama. A Fase 1 sozinha entrega de 20 a 30%; o programa maduro, com as quatro fases operando, tende a 55 a 70%.

A sequência em quatro fases
Da linha de base à governança contínua.



CRITÉRIO DE PRONTO POR FASE

Fase 0: linha de base de 28 dias exportada. **Fase 1:** instruções commitadas e Auto padronizado no time. **Fase 2:** exclusion ativo e hábito de #-mentions visível nas References. **Fase 3:** ULB universal ativo e primeira revisão de ciclo feita.

RECOMENDAÇÃO 1 DEV

R1 · Output control

Por que isso importa

A saída custa ~4 a 5x o input por token nas tabelas públicas dos provedores, e é onde mora a verbosidade: preâmbulo, recapitulação, código repetido. Controlar o formato da resposta é a maior alavanca isolada, e a mais barata: um arquivo de texto commitado.

METÁFORA · O TELEGRAMA E A CARTA

Pedir resposta sem instrução de saída é encomendar relatório a quem é pago por palavra: vem prefácio, recapitulação e apêndice. A instrução transforma a carta em telegrama: só o que muda, nada do resto.

IMPACTO ESPERADO

40 a 70% da saída. Fonte: razão de preço saída/entrada dos provedores; a escada 400-600 para 30-50 tokens é estimativa direcional de campo.

Passos

1. Crie `.github/copilot-instructions.md` na raiz com o bloco de saída do apêndice A1 e faça commit.
2. Adicione regras por área em `.github/instructions/*.instructions.md` com o campo `applyTo` (exemplos completos no A1).
3. Combine com o time que verbosidade vira ajuste de instrução, não convivência: se a resposta veio longa, a regra é que falta uma linha no arquivo.
4. Peça uma tarefa qualquer no Chat e compare o tamanho da resposta antes e depois.

.GITHUB/COPILOT-INSTRUCTIONS.MD (BLOCO DE SAÍDA)

```
## Saída
- Responda com código ou diff; sem preâmbulo e sem resumo ao final.
- Proponha a menor mudança correta; mostre só as linhas alteradas.
- Não repita o enunciado nem reformule a pergunta.
- Para perguntas conceituais: no máximo 5 linhas.
```

COMO VERIFICAR

No Chat, expanda **References** na resposta: o arquivo de instruções deve aparecer listado como aplicado. Se não aparece, o arquivo está fora do caminho ou a feature está desligada nas settings.

Erros comuns

- Instruções genéricas demais ("seja conciso") em vez de regras de formato verificáveis.
- Arquivo gigante: instrução também é input; mantenha alto sinal, sem ensaio.
- Regras conflitantes entre o geral e os por área; o modelo fica errático.

Referências oficiais

GITHUB DOCS

Add repository instructions in your IDE

https://docs.github.com/en/copilot/how-tos/configure-custom-instructions-in-your-ide/add-repository-instructions-in-your-ide

GITHUB DOCS

Your first custom instructions

https://docs.github.com/en/copilot/tutorials/customization-library/custom-instructions/your-first-custom-instructions

RECOMENDAÇÃO 2 DEV ADMIN

R2 · Model routing

Por que isso importa

O preço por token varia em duas ordens de grandeza entre a classe leve e a fronteira (FrugalGPT documentou a dispersão; a tabela por modelo do GitHub mostra a escada atual). Usar fronteira para formatar, renomear ou gerar docs é pagar caro por tarefa trivial. O roteamento manda cada tarefa ao modelo suficiente mais barato, e o Auto ainda dá 10% de desconto no chat para planos pagos.

METÁFORA · O CAMINHÃO E O ENVELOPE

Usar fronteira para formatar código é contratar um caminhão de mudança para entregar um envelope: entrega, mas o frete não faz sentido. O roteamento escolhe o veículo pelo tamanho da carga.

IMPACTO ESPERADO

40 a 70% da conta. FrugalGPT (Stanford, TMLR): 50 a 98% no cenário do estudo; RouterBench confirma a curva custo-qualidade. Banda deliberadamente conservadora.

Passos

1. Padronize **Auto** no model picker do time; fronteira só em troca manual, para raciocínio difícil, e reasoning effort alto só onde agrega.
2. No Visual Studio, desligue `Enhance non-chat requests with premium models` (Tools > Options > GitHub > Copilot > Editor); no VS Code, aponte o utility model para a classe leve.
3. Admin: fixe a política de modelos da organização (quais modelos os membros podem usar) e habilite a troca de modelo onde fizer sentido.
4. Cole a regra de roteamento (abaixo) no `.agent.md` dos agentes e no acordo do time.

REGRA DE ROTEAMENTO (COLE NO .AGENT.MD E NO ACORDO DO TIME)

```
trivial   → classe leve / incluída, sem reasoning
            (formatar, renomear, docs, commit, boilerplate)
padrão    → classe intermediária
            (implementação guiada, testes, revisão)
complexo  → fronteira, reasoning conforme a tarefa
            (refatoração multi-arquivo, arquitetura, bug difícil)
nunca     → fronteira para tarefa trivial
```

COMO VERIFICAR

Abra a tabela **AI model comparison** e confira as classes; depois acompanhe no dashboard de billing a participação da fronteira no consumo: ela deve cair semana a semana.

Erros comuns

- Pensar por nome de modelo em vez de classe; os nomes mudam, a tabela viva é a referência.
- Trocar o modelo do chat e achar que mudou o das inline suggestions: são independentes.
- Esquecer que Extensions podem sobrepor o modelo selecionado.

Referências oficiais

GITHUB DOCS

Changing the chat model

https://docs.github.com/en/copilot/how-tos/use-ai-models/change-the-chat-model

GITHUB DOCS

AI model comparison (tabela viva de preços)

https://docs.github.com/en/copilot/reference/ai-models/model-comparison

PESQUISA · STANFORD (TMLR)

FrugalGPT

https://arxiv.org/abs/2305.05176

PESQUISA

RouterBench

https://arxiv.org/abs/2403.12031

RECOMENDAÇÃO 3 DEV ADMIN

R3 · Modelos locais

Por que isso importa

Pela mecânica oficial do BYOK, modelos que rodam na sua máquina não consomem AI Credits. Commit message, boilerplate, testes simples e perguntas rápidas saem da conta. É exclusividade do VS Code, governada pela política da organização, e não vale para completions.

METÁFORA · COZINHAR EM CASA

Modelo local é cozinhar em casa: nem toda refeição precisa de restaurante. A rotina sai da conta, e o orçamento sobra para os jantares que importam, as tarefas de fronteira.

IMPACTO ESPERADO

Rotina a custo zero de AI Credits (mecânica oficial). Participação no custo total: um dígito a ~15%, estimativa direcional dependente do mix e do hardware.

Passos

1. Admin: confirme a política `Bring Your Own Language Model Key in VS Code` (ligada por padrão em Business e Enterprise) e, para certos modelos, `Editor Preview Features`.
2. Dev: instale e registre o Ollama, ou o Foundry Local via AI Toolkit (comandos abaixo). Requisitos: VS Code 1.113+, Copilot Chat 0.41.0+.
3. Selecione o modelo local no picker do Chat (grupo Local) e roteie a rotina para ele.
4. Lembre: agent mode exige modelo com tool calling; modelos sem isso não aparecem no picker de agente.

OLLAMA

```
ollama pull llama3.1
ollama launch vscode
# manual: Chat: Manage Language Models → Ollama → http://localhost:11434
```

FOUNDRY LOCAL (WINDOWS / MACOS)

```
winget install Microsoft.FoundryLocal
# macOS: brew tap microsoft/foundrylocal && brew install foundrylocal
foundry model run phi-4
# VS Code: extensão AI Toolkit → Add model → Foundry Local via AI Toolkit
```

COMO VERIFICAR

O modelo local aparece no picker e responde; no dashboard de billing, o consumo das tarefas roteadas não aparece como AI Credits.

Erros comuns

- Mandar tarefa de fronteira para o modelo local e culpar a ferramenta pela qualidade.
- Esquecer o requisito de versão (VS Code 1.113+, Copilot Chat 0.41.0+, Ollama 0.18.3+).
- Esperar BYOK em completions ou fora do VS Code: não existe.

BYOK enterprise e a arquitetura de transbordo

O BYOK também existe no plano de controle (public preview): a key conectada nas settings da enterprise ou org (Anthropic, Microsoft Foundry, OpenAI, xAI, AWS Bedrock, Google AI Studio, OpenAI-compatible) libera os modelos no Copilot Chat do GitHub.com e nas IDEs, cobrados no provedor, sem consumir AI Credits. Não há gatilho automático quando o pool acaba: budgets param, não roteiam. O transbordo se desenha assim:

1. **Erga o muro.** ULB universal no nível do pool incluído (ou pouco acima), com alertas em 75, 90 e 100% e metered bloqueado no estouro. O fim dos créditos vira um evento operacional com data, não uma surpresa na fatura.
2. **Garanta a faixa que sobrevive.** Conecte a key do provedor nas settings da enterprise ou da organização (public preview): Anthropic, Microsoft Foundry, OpenAI, xAI, AWS Bedrock, Google AI Studio ou endpoint OpenAI-compatible. Ao bloquear, completions, modelos locais e modelos BYOK seguem funcionando.
3. **Pré-roteie antes de transbordar.** Custom agents com o campo `model:` fixado no modelo BYOK para a rotina; o pool incluído fica reservado para o que se beneficia dos modelos nativos (Auto com desconto, code review, coding agent).
4. **Governe o lado do provedor.** Uma key ou deployment por unidade de negócio espelhando os cost centers (chargeback), max context window por modelo BYOK, e o budget do Azure cost management como segundo disjuntor.
5. **Failover automático só onde há código.** Em superfícies programáticas (Copilot SDK/CLI), capture o erro de bloqueio e recrie a sessão com a key própria. O SDK aceita só credencial estática (sem Entra ID ou managed identity): rotação de key entra no desenho de segurança.

> ANTES DE LEVAR AO CLIENTE
>
> Enterprise BYOK está em public preview; completions nunca usam BYOK; o desconto do Auto e os recursos do lado GitHub (code review, coding agent) valem só nos modelos nativos. Valide num piloto: bloqueio forçado, BYOK respondendo, consumo aparecendo no provedor. O diagrama das faixas está no runbook, R3.

Referências oficiais

GITHUB CHANGELOG
BYOK in VS Code now available (abril de 2026)
https://github.blog/changelog/2026-04-22-bring-your-own-language-model-key-in-vs-code-now-available/

VS CODE DOCS
Language models (BYOK e locais)
https://code.visualstudio.com/docs/agent-customization/language-models

OLLAMA DOCS
VS Code integration
https://docs.ollama.com/integrations/vscode

MICROSOFT LEARN
Foundry Local CLI
https://learn.microsoft.com/en-us/azure/foundry-local/reference/reference-cli

> GITHUB CHANGELOG
>
> **Enterprise BYOK em public preview (2025-11-20)**
>
> https://github.blog/changelog/2025-11-20-enterprise-bring-your-own-key-byok-for-github-copilot-is-now-in-public-preview/

> GITHUB CHANGELOG
>
> **BYOK enhancements (2026-01-15)**
>
> https://github.blog/changelog/2026-01-15-github-copilot-bring-your-own-key-byok-enhancements/

RECOMENDAÇÃO 4 DEV ADMIN

R4 · Escopo de contexto

Por que isso importa

Contexto é input cobrado: despejar o repositório inteiro custa caro e degrada a resposta, porque o modelo se dilui no ruído. Curar o contexto melhora a qualidade e corta a conta ao mesmo tempo, e o content exclusion tira do alcance o que nunca deveria entrar.

> METÁFORA · A PASTA CERTA NA REUNIÃO
>
> Contexto é a pasta que você leva para a reunião: levar o arquivo morto inteiro confunde e custa caro. Levar a pasta certa resolve em dez minutos, e o modelo trabalha igual.

> IMPACTO ESPERADO
>
> 40 a 80% do input. LLMLingua (Microsoft, EMNLP 2023) comprimiu prompts em até 20x com perda de ~1,5 ponto; a banda usada é conservadora ante esse teto.

Passos

1. Troque o despejo por referência precisa: `#file`, `#sym`, `#changes`; use `#codebase` só quando a busca ampla é o objetivo (`@workspace` é a sintaxe antiga).
2. Admin: configure o content exclusion em Settings > Copilot > Content exclusion com os paths do A1; propaga em até 30 minutos.
3. Mantenha o contexto estável da sessão nos arquivos de instrução, não em colagens repetidas.
4. Atenção: content exclusion não vale em Edit e Agent mode, e bloqueia completions nos arquivos afetados; para agentes, controle escopo via #-mentions e agentes restritos.

CONTENT EXCLUSION (SETTINGS > COPILOT > CONTENT EXCLUSION)

```
- "/**/.env*"
- "/dist/**"
- "/secrets/**"
- "**/*.pem"
- "**/fixtures/large/**"
```

> COMO VERIFICAR
>
> Nas References da resposta, os arquivos citados são só os relevantes; um arquivo excluído não aparece em chat nem em completions (teste com um path da lista).

Erros comuns

- Colar logs e arquivos inteiros no prompt em vez de referenciar.
- Excluir diretórios que o time precisa: exclusion mal calibrado vira atrito.
- Contar com exclusion como segurança em Edit/Agent mode: não é suportado lá.

Referências oficiais

VS CODE DOCS

Manage context for AI (#-mentions)

https://code.visualstudio.com/docs/copilot/chat/copilot-chat-context

GITHUB DOCS

Exclude content from GitHub Copilot

https://docs.github.com/en/copilot/how-tos/configure-content-exclusion/exclude-content-from-copilot

PESQUISA · MICROSOFT (EMNLP 2023)

LLMLingua

https://arxiv.org/abs/2310.05736

RECOMENDAÇÃO 5 DEV

R5 · Cache e memória

Por que isso importa

Em loops de agente, 80 a 95% do input se repete entre os pedidos. O token em cache custa de 10 a 50% do token novo, conforme o modelo (Anthropic 0,1x; OpenAI 50% automático acima de 1.024 tokens de prefixo); a memória elimina a reexplicação entre sessões. São dois ganhos que se compõem.

METÁFORA · O CRACHÁ DE ACESSO

Cache é o crachá: na primeira visita você se registra na portaria; nas seguintes, só encosta o crachá. Reapresentar-se a cada porta é pagar a recepção de novo, todas as vezes.

IMPACTO ESPERADO

30 a 50% do input em loops de agente. Fonte: preço oficial de cache dos provedores; a literatura KV (ChunkKV, KVCompose) sustenta 70 a 90% de compressão com perda mínima.

Passos

1. Estabilize o prefixo: instruções e AGENTS.md no topo, sem editar no meio da sessão; cada edição invalida o cache de tudo que vem depois.
2. Agrupe trabalho relacionado na mesma sessão; não rerode agentes sobre diff inalterado.
3. Ligue o Copilot Memory (public preview; em enterprise vem desativado, decisão de governança) e cure: revise os fatos capturados, o que envelhece sai.
4. Coloque o que é estável e reusável no AGENTS.md (A1), que vira prefixo cacheável e memória de processo do repositório.

AGENTS.MD (RAIZ DO REPOSITÓRIO)

```
# AGENTS.md

## Build e testes
- `npm test` roda a suíte; `npm run lint` antes do commit.
- CI exige cobertura estável; não remova testes para passar.

## Convenções
- Commits convencionais; PRs pequenos e focados.
- Migrations sempre com rollback.

## Custo
- Modelo padrão: Auto. Fronteira só em refatoração multi-arquivo.
- Refira arquivos com #file; não cole trechos longos.
```

COMO VERIFICAR

Compare duas sessões equivalentes no debug/telemetria: com prefixo estável, a fração de tokens cacheados sobe e o custo por interação cai; na memória, os fatos novos aparecem listados para revisão.

Erros comuns

- Editar o copilot-instructions.md no meio de uma sessão longa de agente (invalida o cache).

- Memória sem curadoria: fato velho vira contexto inchado persistente.
- Fragmentar a mesma tarefa em dez sessões curtas.

Referências oficiais

GITHUB DOCS

About GitHub Copilot Memory

https://docs.github.com/en/copilot/concepts/agents/copilot-memory

ANTHROPIC DOCS

Prompt caching (leitura a 0,1x)

https://platform.claude.com/docs/en/build-with-claude/prompt-caching

OPENAI DOCS

Prompt caching (50% automático)

https://platform.openai.com/docs/guides/prompt-caching

RECOMENDAÇÃO 6 DEV

R6 · Primitivos

Por que isso importa

Primitivos transformam disciplina individual em infraestrutura versionada: instruções, agentes, prompts e hooks viram arquivos no repositório, revisados como qualquer mudança. Cada um é governança e economia ao mesmo tempo, porque reduz reexplicação e restringe escopo em todo pedido futuro.

> METÁFORA · O CORRIMÃO NA ESCADA
>
> Primitivos são o corrimão construído na escada: ninguém precisa lembrar de se segurar, ele está lá para todo mundo, em toda descida.

> IMPACTO ESPERADO
>
> Ganho composto, não quantificado em banda única pela literatura: é o que torna permanentes os cortes das outras alavancas.

Passos

1. Adote na ordem: instruções gerais, instruções com escopo (`applyTo`), agentes customizados, prompt files, hooks.
2. No VS Code, gere com `/create-instruction`, `/create-agent`, `/create-prompt`, `/create-hook`; revise e commite como código.
3. Restrinja cada agente a uma tarefa, com modelo e ferramentas mínimos (exemplo completo no A1).
4. Use o catálogo awesome-copilot como inspiração validada antes de inventar o seu.

.GITHUB/AGENTS/COST-AWARE-REVIEWER.AGENT.MD

```
---
name: cost-aware-reviewer
description: Revisão de PR focada em diff, com modelo intermediário.
model: <classe-intermediária>
tools: ['search', 'changes']
---
Você revisa pull requests deste repositório.
Responda apenas com:
1. Lista numerada de problemas (1 linha cada).
2. Diff sugerido por problema, quando houver correção.
Não recapitule o PR, não elogie, não explique o óbvio.
```

.GITHUB/PROMPTS/RELEASE-NOTES.PROMPT.MD

```
---
mode: agent
description: Gera notas de release a partir dos commits da branch.
---
Gere notas de release concisas a partir de #changes:
- Agrupe por tipo (feat, fix, chore).
- 1 linha por item, sem adjetivos.
- Inclua breaking changes em destaque no topo.
```

COMO VERIFICAR

Os arquivos aparecem no picker (agentes e prompts) e nas References quando aplicados; o time para de colar o mesmo prompt mágico no chat.

Erros comuns

- Agente faz-tudo com todas as ferramentas: escopo largo é custo largo.
- Primitivo sem dono: arquivo desatualizado orienta errado para sempre.
- Pular a revisão: primitivo é código, entra por PR.

Referências oficiais

VS CODE DOCS

Agent customization overview

https://code.visualstudio.com/docs/agent-customization/overview

GITHUB

awesome-copilot (catálogo validado)

https://github.com/github/awesome-copilot

RECOMENDAÇÃO 7 ADMIN

R7 · Budgets e medição

Por que isso importa

Caps são guard-rail de variância: não cortam o regime, evitam a surpresa. Cada licença traz AI Credits agrupados em pool (1.900 por usuário em Business, 3.900 em Enterprise; na promoção até 2026-09-01, 3.000 e 7.000); quando o pool acaba, começa o metered, e é aí que os budgets seguram. Os budgets por usuário estão em GA desde 2026-06-01. E a medição contínua é o que transforma banda em número real.

METÁFORA · O DISJUNTOR

Budget é disjuntor, não economizador: ele não reduz a conta de luz do mês, evita o incêndio do pico. A economia vem das outras alavancas; o cap segura a variância.

IMPACTO ESPERADO

Guard-rail de variância, não corte de regime. Sem ULB, um loop de agente pode consumir sozinho uma fatia grande do pool da unidade.

Passos

1. Crie o **ULB universal** em `Settings > Billing > Budgets and alerts > New budget` (Bundled AI credits budget), acima do valor por licença, com alertas em 75, 90 e 100%.
2. Adicione overrides individuais para os consumidores pesados legítimos e cost centers por unidade de negócio.
3. Defina o enterprise spending limit como failsafe global.
4. Ligue `Copilot usage metrics`, exporte a linha de base (28 dias, CSV e NDJSON) e institua a revisão por ciclo: consumo por usuário, por modelo e por feature, comparado com a banda.
5. Lembrete operacional: desde 2026-06-01 o Copilot code review também consome Actions minutes; o admin pode fixar o runner padrão no nível da organização.

CHECKLIST DE CONFIGURAÇÃO (PLANO DE CONTROLE)

```
[ ] ULB universal > valor da licença (alertas 75/90/100)
[ ] Overrides p/ heavy users legítimos
[ ] Cost centers por unidade
[ ] Enterprise spending limit (failsafe)
[ ] Copilot usage metrics ON + export da linha de base
[ ] Revisão de ciclo agendada (usuário x modelo x feature)
```

COMO VERIFICAR

Force um teste controlado: um budget baixo num usuário de teste deve bloquear o consumo metered ao estourar (sem fallback automático; completions seguem). Os alertas chegam nos thresholds.

Erros comuns

- Confundir budget com economia: o regime cai com R1 a R6, não com o cap.
- ULB tão baixo que vira atrito diário e o time desliga a iniciativa.

- Medir uma vez e nunca mais: a banda só aperta com revisão por ciclo.

Referências oficiais

GITHUB DOCS
Set up budgets
https://docs.github.com/en/billing/how-tos/set-up-budgets

GITHUB DOCS
Getting started with budget controls
https://docs.github.com/en/copilot/tutorials/budgets/getting-started-with-budget-controls

GITHUB DOCS
Copilot usage metrics
https://docs.github.com/en/copilot/concepts/copilot-usage-metrics/copilot-metrics

GITHUB CHANGELOG
Updates to Copilot billing and plans (2026-06-01)
https://github.blog/changelog/2026-06-01-updates-to-github-copilot-billing-and-plans/

APÊNDICE A1

Todos os artefatos, prontos para colar

O kit completo

Tudo que os capítulos citam, em versão integral. Ordem de adoção: instruções gerais, instruções por área, AGENTS.md, agente, prompt, exclusion, regra de roteamento. Troque os placeholders entre <> e revise por PR.

.GITHUB/COPILOT-INSTRUCTIONS.MD (COMPLETO)

```
# Instruções do repositório

## Saída
- Responda com código ou diff; sem preâmbulo e sem resumo ao final.
- Proponha a menor mudança correta; mostre só as linhas alteradas.
- Não repita o enunciado nem reformule a pergunta.
- Para perguntas conceituais: no máximo 5 linhas.

## Contexto
- Stack: <linguagem/framework>. Padrões em docs/CONTRIBUTING.md.
- Prefira referências por arquivo (#file) a trechos colados.
- Testes acompanham toda lógica nova.

## Estilo
- Código idiomático da stack; nomes em inglês; comentários só onde o porquê não é óbvio.
```

.GITHUB/INSTRUCTIONS/TESTS.INSTRUCTIONS.MD

```
---
applyTo: "**/*.test.*"
---
- Gere apenas o corpo do teste; sem explicar o scaffolding.
- Um assert claro por caso; nomes de teste descrevem o comportamento.
- Não duplique cenários já cobertos no arquivo.
```

.GITHUB/INSTRUCTIONS/DOCS.INSTRUCTIONS.MD

```
---
applyTo: "docs/**/*.md"
---
- Tom direto, segunda pessoa, sem jargão interno.
- Exemplos executáveis; um por conceito.
- Atualize o índice quando criar página nova.
```

AGENTS.MD

```
# AGENTS.md

## Build e testes
- `npm test` roda a suíte; `npm run lint` antes do commit.

## Convenções
- Commits convencionais; PRs pequenos e focados.

## Custo
- Modelo padrão: Auto. Fronteira só em refatoração multi-arquivo.
- Refira arquivos com #file; não cole trechos longos.
```

.GITHUB/AGENTS/COST-AWARE-REVIEWER.AGENT.MD

```
---
name: cost-aware-reviewer
description: Revisão de PR focada em diff, com modelo intermediário.
model: <classe-intermediária>
tools: ['search', 'changes']
---
Você revisa pull requests deste repositório.
Responda apenas com:
1. Lista numerada de problemas (1 linha cada).
2. Diff sugerido por problema, quando houver correção.
Não recapitule o PR, não elogie, não explique o óbvio.
```

.GITHUB/PROMPTS/RELEASE-NOTES.PROMPT.MD

```
---
mode: agent
description: Gera notas de release a partir dos commits da branch.
---
Gere notas de release concisas a partir de #changes:
- Agrupe por tipo (feat, fix, chore).
- 1 linha por item, sem adjetivos.
- Breaking changes em destaque no topo.
```

CONTENT EXCLUSION (SETTINGS > COPILOT > CONTENT EXCLUSION)

```
- "/**/.env*"
- "/dist/**"
- "/secrets/**"
- "**/*.pem"
- "**/fixtures/large/**"
```

REGRA DE ROTEAMENTO (PARA O ACORDO DO TIME E OS .AGENT.MD)

```
trivial   → classe leve / incluída, sem reasoning
padrão    → classe intermediária
complexo  → fronteira, reasoning conforme a tarefa
nunca     → fronteira para tarefa trivial
```

SOBRE HOOKS

Hooks (guard-rails de loop e de ação do agente) são gerados com `/create-hook` no VS Code; o formato evolui com a extensão, então gere pelo comando em vez de copiar um esquema daqui. O objetivo: impedir reexecução sobre diff inalterado e limitar passos por tarefa.

APÊNDICE A2

Resumo de uma página

A tabela para imprimir

RECOMENDAÇÃO	ONDE SE CONFIGURA	BANDA	QUEM
R1 Output control	.github/copilot-instructions.md	40 a 70% da saída	Dev
R2 Model routing	Auto + política + regra de roteamento	40 a 70% da conta	Dev + Admin
R3 Modelos locais	Ollama / Foundry Local no VS Code	rotina a custo zero	Dev + Admin
R4 Escopo de contexto	#-mentions + content exclusion	40 a 80% do input	Dev + Admin
R5 Cache e memória	prefixo estável + Copilot Memory	30 a 50% do input	Dev
R6 Primitivos	.github/agents, instructions, prompts	composto	Dev
R7 Budgets e medição	ULB + cost centers + métricas	guard-rail de variância	Admin

LEMBRETE

Os ganhos não se somam, compõem-se: 20 a 30% num começo enxuto, 55 a 70% num programa maduro. Bandas com base de evidência declarada (preço oficial, pesquisa publicada, estimativa direcional rotulada); o número que vale é o seu, contra a linha de base de 28 dias.

APÊNDICE A3

Fontes

Documentação oficial

GITHUB DOCS
Usage-based billing for organizations and enterprises
https://docs.github.com/en/copilot/concepts/billing/usage-based-billing-for-organizations-and-enterprises

GITHUB CHANGELOG
Updates to Copilot billing and plans (2026-06-01)
https://github.blog/changelog/2026-06-01-updates-to-github-copilot-billing-and-plans/

GITHUB DOCS
AI model comparison
https://docs.github.com/en/copilot/reference/ai-models/model-comparison

GITHUB DOCS
Set up budgets
https://docs.github.com/en/billing/how-tos/set-up-budgets

VS CODE DOCS
Agent customization
https://code.visualstudio.com/docs/agent-customization/overview

Pesquisa publicada e preço oficial

STANFORD (TMLR)
FrugalGPT: economia de 50 a 98% via cascata de modelos
https://arxiv.org/abs/2305.05176

MICROSOFT (EMNLP 2023)
LLMLingua: compressão de prompt de até 20x
https://arxiv.org/abs/2310.05736

BENCHMARK
RouterBench: roteamento multi-modelo
https://arxiv.org/abs/2403.12031

ANTHROPIC DOCS
Prompt caching (leitura a 0,1x)
https://platform.claude.com/docs/en/build-with-claude/prompt-caching

OPENAI DOCS
Prompt caching (50% automático)
https://platform.openai.com/docs/guides/prompt-caching