RUNBOOK COMPLETO · USAGE-BASED BILLING · GITHUB COPILOT

# Otimização de custo no GitHub Copilot, o runbook completo.

Guia didático e detalhado de por onde começar e de como fazer. Cada procedimento explica o porquê e o como funciona por baixo, traz passos numerados com o caminho exato, o conteúdo pronto para colar, um exemplo trabalhado, como verificar, os erros comuns, e referência aos links oficiais do GitHub, do VS Code e da Microsoft Learn.

**Paula Silva**
Software Global Black Belt
paulasilva@microsoft.com

**Escopo:** fundamentos, 7 procedimentos, operação, anti-padrões e apêndices
**Atualizado:** 2026-06-10 · **Base:** documentação oficial validada

# Conteúdo

PARTE 0 · FUNDAMENTOS DEV

# Antes de começar: instalar, entrar e como usar

## Como usar este runbook

Os procedimentos estão em ordem de retorno: comece pelo topo. Cada um traz um selo de quem executa, DEV para o desenvolvedor no editor e ADMIN para o owner de organização ou enterprise. Onde a tela do produto importa, o passo aponta a documentação oficial, que traz a captura atualizada.

## Instalar e entrar

Este runbook assume o Copilot já funcionando. Se você está começando do zero, faça primeiro estes quatro passos.

1. Instale o **Visual Studio Code** (code.visualstudio.com).
2. No VS Code, abra Extensions e instale **GitHub Copilot** e **GitHub Copilot Chat** pelo Marketplace.
3. Faça login com a sua conta do GitHub quando o VS Code pedir; confirme que a conta ativa é a que tem o plano de Copilot esperado.
4. Verifique que o ícone do Copilot está ativo, que o texto fantasma (completions) aparece ao digitar, e que o painel de Chat abre.

> COMO VERIFICAR
>
> Digite algumas linhas de código e veja se aparece a sugestão em cinza; abra o painel de Chat e faça uma pergunta simples. Se as duas coisas funcionam, o ambiente está pronto para o resto do runbook.

## Referências oficiais

> MICROSOFT LEARN
>
> **Get Started with GitHub Copilot**
> https://learn.microsoft.com/en-us/training/modules/get-started-github-copilot/

> VS CODE DOCS
>
> **Chat in VS Code (visão geral e setup)**
> https://code.visualstudio.com/docs/copilot/chat/copilot-chat

PARTE 0 · FUNDAMENTOS

# A equação, os tokens e o UBB

## Por que isso importa

Sob o Usage-Based Billing, você não paga mais por assento com requests fixos: paga pelo que consome, medido em AI Credits. Para otimizar, é preciso entender exatamente o que forma o custo de cada interação. Sem isso, qualquer ajuste é chute.

**A equação do custo, em uma imagem**
Por que toda alavanca ataca o modelo, os tokens, ou os dois.

A EQUAÇÃO DO CUSTO

**CUSTO = MODELO × TOKENS**

Dois fatores, e só dois. Toda alavanca deste runbook ataca um deles, ou os dois.

| ENTRADA | SAÍDA | CACHE |
|---|---|---|
| **1x** | **~4 a 5x** | **0,1 a 0,5x** |
| prompt, histórico, arquivos, instruções e contexto | o que o modelo gera; a classe mais cara | contexto repetido reaproveitado (Anthropic 0,1x; OpenAI 0,5x) |

Fora da equação: code completions e Next Edit Suggestions seguem incluídos no plano e não consomem AI Credits.

## Como funciona o custo

Cada interação de chat ou agente consome **tokens**. Há três tipos: tokens de **entrada** (o que é enviado ao modelo: seu prompt, o histórico, o conteúdo de arquivos, instruções), tokens de **saída** (o que o modelo gera) e tokens de **cache** (contexto que o modelo reutiliza). Cada token é precificado conforme o modelo usado, e o total é convertido em AI Credits, onde `1 AI Credit = US$ 0,01`.

Logo, o custo de uma interação depende de apenas dois fatores: **qual modelo** e **quantos tokens**. Toda alavanca de otimização ataca um desses dois, ou os dois.

EXEMPLO ILUSTRATIVO

Uma resposta de chat sem controle de formato pode gerar de 400 a 600 tokens de saída (explicação longa, repetição do código). A mesma tarefa, com a saída restrita a diff, gera de 30 a 50 tokens. Como a saída costuma custar cerca de 5x o token de entrada, esse corte de formato é a economia mais barata e imediata que existe.

O QUE NÃO CONSOME AI CREDITS

Code completions (o texto fantasma) e Next Edit Suggestions seguem incluídos no plano e **não consomem AI Credits**. Otimizar o tamanho de identificadores para economizar token de completion é esforço perdido. O consumo real está em **chat** e nas **tarefas agênticas**.

A REGRA QUE ORGANIZA O RUNBOOK

As alavancas não se somam, elas se compõem: cada uma age sobre uma base diferente de tokens. Por isso a ordem importa. Os sete procedimentos da Parte 1 estão em ordem de retorno: comece pelo output e pelo routing, que carregam o resultado; o resto refina.

## Referências oficiais

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 BLOG

**GitHub Copilot is moving to usage-based billing**

https://github.blog/news-insights/company-news/github-copilot-is-moving-to-usage-based-billing/

GITHUB DOCS

**Requests in GitHub Copilot**

https://docs.github.com/en/copilot/concepts/billing/copilot-requests

PARTE 0 · FUNDAMENTOS ADMIN DEV

# Pré-requisitos, papéis e políticas

## Por que isso importa

Metade dos procedimentos depende de plano, papel e política corretos. Recursos como content exclusion e o BYOK governado existem só em Business e Enterprise, e budgets e políticas são de admin. Conferir isso antes evita travar no meio do caminho.

## Checklist de pré-requisitos

1. **Plano.** Confirme Copilot Business ou Enterprise. Content exclusion e o BYOK governado por política são exclusivos desses planos.
2. **Versões.** VS Code `1.113` ou superior e a extensão GitHub Copilot Chat `0.41.0` ou superior. Para modelos locais via Ollama, `Ollama 0.18.3` ou superior.
3. **Papéis.** Identifique o **admin** (organização ou enterprise: define políticas, budgets e content exclusion) e o **desenvolvedor** (aplica instruções, escopo de contexto e roteia o modelo). Vários passos exigem o papel de admin.
4. **Políticas no plano de controle** (GitHub.com, organização ou enterprise, seção Copilot): habilite `Copilot usage metrics`; conceda aos membros a permissão de trocar de modelo; para modelos locais, confirme a política `Bring Your Own Language Model Key in VS Code` (ligada por padrão) e, para certos modelos, `Editor Preview Features`.

> ANTES DE INICIAR
>
> Sem o papel certo, metade do runbook trava. Alinhe com o admin quais políticas e budgets ele vai habilitar, e em qual janela. O desenvolvedor pode aplicar R1, R4, R5 e R6 sozinho; R2, R3 e R7 dependem de política ou papel de admin.

## Aplicabilidade por plano

Nem toda alavanca existe em todo plano. Esta tabela mostra o que se aplica ao indivíduo (Free, Pro, Pro+) e à organização (Business, Enterprise). Code completions e Next Edit Suggestions são incluídos em todos os planos.

| PROCEDIMENTO | INDIVÍDUO (FREE, PRO, PRO+, MAX) | ORGANIZAÇÃO (BUSINESS, ENTERPRISE) |
|---|---|---|
| **R1 Output control** | Sim | Sim |
| **R2 Model routing** | Picker e Auto; sem política de organização | Completo, com política de modelos |
| **R3 Modelos locais** | Sim; o Free já habilita a seleção de modelo | Sim, governado por política de admin |
| **R4 Contexto e content exclusion** | #-mentions sim; content exclusion não | Completo, com content exclusion |
| **R5 Cache e memória** | Cache sim; Copilot Memory em Pro e Pro+ | Sim |
| **R6 Primitivos** | Sim | Sim |
| **R7 Budgets e medição** | Budget pessoal de gasto adicional | Completo: ULB, cost centers, dashboard |

## Referências oficiais

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

> VS CODE DOCS
> **AI language models in VS Code**
> https://code.visualstudio.com/docs/agent-customization/language-models

> GITHUB DOCS
> **Changing the AI model for Copilot Chat**
> https://docs.github.com/en/copilot/how-tos/use-ai-models/change-the-chat-model

PARTE 0 · FUNDAMENTOS ADMIN

# Medir primeiro: a linha de base

## Por que isso importa

Otimização sem linha de base é palpite. Você precisa saber, antes de mexer, quanto cada usuário, modelo e feature consome hoje. Só assim dá para comparar o antes e o depois e tratar o ganho como uma banda que aperta com os dados, não como um número inventado.

## Passos detalhados

1. No plano de controle, habilite a política `Copilot usage metrics` (organização ou enterprise). Sem ela, o dashboard fica vazio.
2. Acesse a enterprise (ou organização) e abra a aba **Insights**. O dashboard de métricas de uso cobre uma janela de 28 dias e atualiza diariamente (o dado pode estar até 3 dias atrás de UTC).
3. Explore os breakdowns: adoção, por feature, por modelo e por linguagem, mais o code generation dashboard.
4. Para dado bruto e série mais longa que 28 dias, use a **exportação NDJSON** e o **CSV de uso**; leve para a sua ferramenta de BI se precisar.
5. Registre a linha de base: créditos por usuário, por modelo e por feature. É contra ela que você compara depois de cada alavanca.

COMO VERIFICAR

O dashboard mostra tendências e o consumo por modelo; o CSV e o NDJSON trazem consumo por usuário, que é o que permite atribuição fina e os overrides de budget no R7.

ERROS COMUNS

- Otimizar antes de medir, e depois não conseguir provar o ganho.
- Esquecer de habilitar a política de métricas e achar que não há dado.
- Confundir o dashboard de 28 dias com a série histórica; para o longo prazo, exporte.

## Referências oficiais

GITHUB DOCS

**GitHub Copilot usage metrics**

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

GITHUB DOCS

**Viewing the Copilot usage metrics dashboard**

https://docs.github.com/en/copilot/how-tos/administer-copilot/view-usage-and-adoption

GITHUB CHANGELOG

**Copilot metrics is now generally available**

https://github.blog/changelog/2026-02-27-copilot-metrics-is-now-generally-available/

PARTE 0 · FUNDAMENTOS

# Os impactos possíveis e a base de evidência

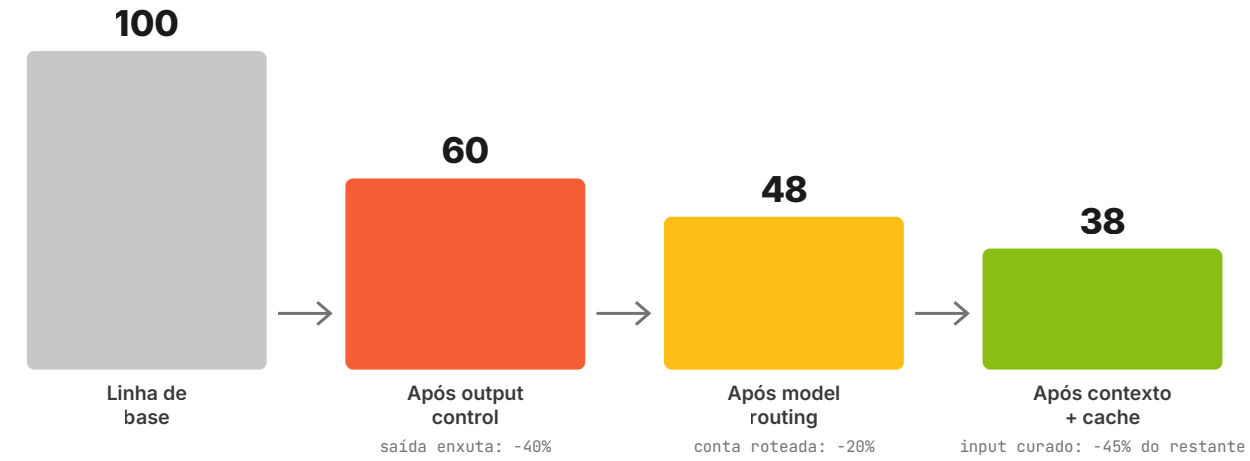
## Como ler as bandas

Todo número de impacto neste runbook é uma banda de planejamento, não uma promessa. As bandas vêm de três tipos de fonte: preço oficial dos provedores (o mais duro), pesquisa publicada e revisada (Stanford, Microsoft Research e benchmarks públicos), e estimativa direcional de campo, sempre rotulada como tal. E elas não se somam: compõem-se multiplicativamente sobre bases de tokens diferentes.

**Como as bandas se compõem**
Uma conta de 100 créditos passando pelas alavancas, na ordem.

A CASCATA DE COMPOSIÇÃO

**Os ganhos não se somam: compõem-se sobre bases diferentes.**




De 100 a ~38: faixa madura de 55 a 70%. Aritmética de composição, compatível com o teto da literatura (FrugalGPT, 50 a 98%).

| ALAVANCA | BANDA USADA | BASE DE EVIDÊNCIA |
|---|---|---|
| **Output control** | 40 a 70% da saída | Razão de preço saída/entrada de ~4 a 5x nas tabelas públicas dos provedores e na tabela por modelo do GitHub. A escada de especificação (400-600 → 30-50 tokens) é estimativa direcional de campo. |
| **Model routing** | 40 a 70% da conta | FrugalGPT (Stanford, TMLR): cascata iguala o melhor modelo com economia de 50 a 98%. RouterBench (2024): roteadores igualam ou superam o melhor modelo único a custo menor. A banda usada é conservadora. |
| **Modelos locais** | rotina a custo zero; um dígito a ~15% da conta | Mecânica oficial: BYOK não consome AI Credits (GitHub Changelog, 2026). A participação no custo total é estimativa direcional, dependente do mix e do hardware. |
| **Escopo de contexto** | 40 a 80% do input | LLMLingua (Microsoft, EMNLP 2023): compressão de até 20x com perda de ~1,5 ponto. Survey de context engineering (2025). A banda usada é conservadora ante o teto da literatura. |
| **Cache** | 30 a 50% do input em loops de agente | Docs oficiais de preço: leitura de cache a 0,1x o input na Anthropic e 50% na OpenAI (automático acima de 1.024 tokens). A banda de loops vem da prática documentada de caching. A literatura de compressão de cache KV (ChunkKV, KVCompose) sustenta compressões de 70 a 90% com perda mínima. |
| **Memória e primitivos** | composto, não quantificado | GitHub Docs (Copilot Memory, custom instructions). Economia por reexplicação evitada; a literatura não publica banda única. |
| **Combinado** | 20 a 30% enxuto; 55 a 70% maduro | Aritmética de composição sobre as bandas acima. Compatível com o teto da literatura (FrugalGPT, 50 a 98%). Banda diretiva, a confirmar contra a sua linha de base. |

A REGRA DE HONESTIDADE

Onde a fonte é preço oficial, o número é fato. Onde é pesquisa, o resultado vale para o cenário do estudo e a banda aqui é deliberadamente mais conservadora. Onde é estimativa direcional, o texto diz isso. Em todos os casos, o número que vale é o seu: compare com a linha de base da seção anterior e deixe a banda apertar com os dados reais.

## Referências oficiais

PESQUISA · STANFORD (TMLR)
**FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance**
https://arxiv.org/abs/2305.05176

PESQUISA · MICROSOFT (EMNLP 2023)
**LLMLingua: Compressing Prompts for Accelerated Inference**
https://arxiv.org/abs/2310.05736

PESQUISA · ROUTERBENCH
**RouterBench: A Benchmark for Multi-LLM Routing Systems**
https://arxiv.org/abs/2403.12031

ANTHROPIC DOCS
**Prompt caching**
https://platform.claude.com/docs/en/build-with-claude/prompt-caching

OPENAI DOCS
**Prompt caching**
https://platform.openai.com/docs/guides/prompt-caching

> GITHUB DOCS
> **AI model comparison (preços por modelo)**
> https://docs.github.com/en/copilot/reference/ai-models/model-comparison

PARTE 1 · POR ONDE COMEÇAR DEV

# R1 · Output control

## Por que isso importa

A saída custa cerca de cinco vezes o input por token. Uma resposta cheia de preâmbulo, explicação e repetição do código inteiro é a forma mais cara e mais comum de desperdício. Controlar o formato da resposta é a maior alavanca isolada de custo, e a mais barata de aplicar.

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 de saída transforma a carta em telegrama: só o que muda, nada do resto.

IMPACTO ESPERADO

40 a 70% da classe de token mais cara. A saída custa ~4 a 5x o input nas tabelas públicas dos provedores; cortar verbosidade é a alavanca mais barata de aplicar.

## Como funciona

O arquivo `.github/copilot-instructions.md` é injetado automaticamente em cada pedido ao Copilot no contexto daquele repositório. Em vez de você repetir a cada prompt o tom e o formato que quer, isso vira regra padrão. Instruções com escopo (`.instructions.md` com campo `applyTo`) aplicam regras só nos arquivos que casam um glob, por exemplo só nos testes.

OBJETIVO

Tornar a saída direta (código ou diff) o padrão do repositório, sem preâmbulo nem explicação desnecessária.

PRÉ-REQUISITOS

Acesso de escrita ao repositório. Vale para qualquer plano.

## Passos detalhados

1. Na raiz do repositório, crie a pasta `.github` se ela ainda não existir.
2. Crie o arquivo `.github/copilot-instructions.md`.
3. Cole as diretivas de saída (bloco abaixo) e faça commit. As instruções passam a entrar automaticamente nos pedidos ao Copilot.
4. Para regras por área do código, crie `.github/instructions/NOME.instructions.md` com o campo `applyTo` apontando o glob dos arquivos.
5. No prompt pontual, reforce o formato: peça só o diff, ou só a função.

.GITHUB/COPILOT-INSTRUCTIONS.MD

```
- Responda com código ou diff.
- Sem preâmbulo, sem resumo.
- Não explique, a menos que eu peça.
- Menor mudança correta; devolva só as linhas alteradas.
```

.GITHUB/INSTRUCTIONS/TESTS.INSTRUCTIONS.MD

```
---
applyTo: "**/*.test.ts"
---
- Gere só o corpo do teste.
- Sem explicar o scaffolding.
```

EXEMPLO: A ESCADA DE SAÍDA

Sem especificação, uma resposta típica gira em 400 a 600 tokens. Com a instrução terse, cai para 100 a 150. Restrita a diff, fica em 30 a 50. São números diretivos, mas mostram a ordem de grandeza: o mesmo resultado por uma fração do custo de saída.

COMO VERIFICAR

No painel de Chat, expanda a lista de **References** no topo da resposta. Se as instruções foram aplicadas, o arquivo `.github/copilot-instructions.md` aparece como referência clicável.

ERROS COMUNS

- Instruções conflitantes: a escolha do Copilot entre conflitos é não determinística.
- Arquivo longo e genérico: mantenha curto e de alto sinal (espaço em branco entre instruções é ignorado).
- Esperar que a regra afete completions: ela molda chat e agente, não o texto fantasma.

## Referências oficiais

GITHUB DOCS

**Adding repository custom 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

VS CODE DOCS

**Customize AI in Visual Studio Code**

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

PARTE 1 · POR ONDE COMEÇAR DEV ADMIN

# R2 · Model routing

## Por que isso importa

O preço por token varia enormemente entre a classe leve e a de fronteira: duas ordens de grandeza, conforme documentado pelo estudo FrugalGPT (Stanford) e visível na tabela de preços por modelo do GitHub. Usar um modelo de fronteira para formatar, renomear ou gerar docs é pagar caro por uma tarefa trivial. O roteamento por tarefa é a alavanca que age sobre a conta inteira.

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.

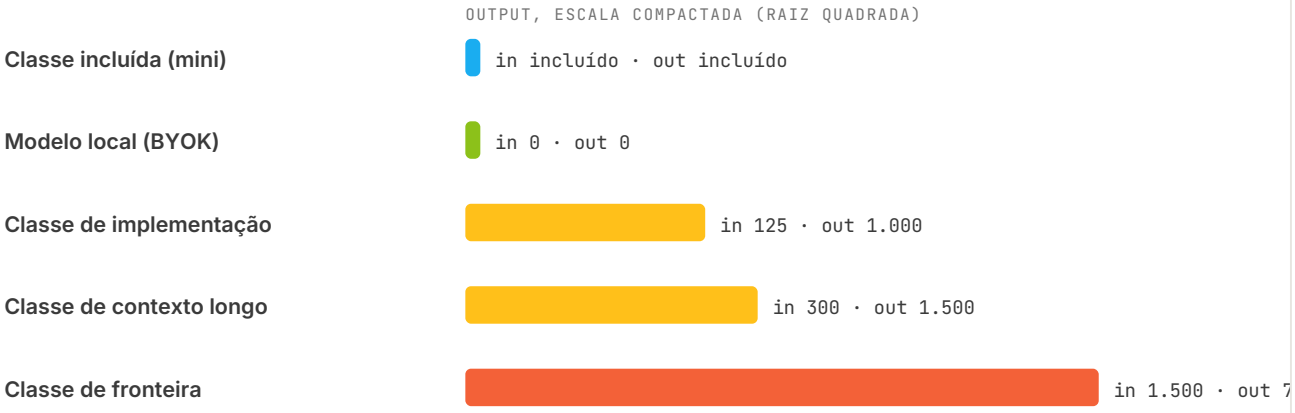
IMPACTO ESPERADO

40 a 70% da conta. FrugalGPT (Stanford) mediu 50 a 98% no cenário do estudo; a banda aqui é deliberadamente conservadora. O Auto ainda dá 10% de desconto no chat.

**A escada de preços por classe de modelo**
O argumento visual do roteamento: a distância entre o trivial e a fronteira.

A ESCADA DE PREÇOS POR CLASSE

**Duas ordens de grandeza entre a classe leve e a fronteira (créditos por 1M de tokens).**




Valores de referência; os modelos e preços mudam. Confira a tabela viva em AI model comparison antes de usar em proposta.

## Como funciona

O modo **Auto** escolhe o modelo por disponibilidade e confiabilidade, e ainda aplica **10% de desconto** nos multiplicadores no chat para planos pagos. Além do modelo do chat, há dois ajustes que cortam consumo silencioso: o **utility model** (modelo leve para tarefas utilitárias como geração de título e mensagem de commit) e a opção de **não enriquecer requests fora do chat com modelos premium**.

OBJETIVO

Padronizar Auto, reservar a fronteira para o que precisa, e cortar o premium onde ele não agrega.

PRÉ-REQUISITOS

Para Business e Enterprise, a organização precisa conceder aos membros a permissão de trocar de modelo (política de modelos).

## Passos detalhados

1. No painel do Copilot Chat, abra o seletor de modelo (canto inferior) e deixe **Auto** como padrão.
2. Reserve modelos de fronteira para o raciocínio difícil; o desenvolvedor troca manualmente só quando justificar.
3. Para cortar consumo fora do chat: no Visual Studio, em `Tools > Options > GitHub > Copilot > Editor`, desligue **Enhance non-chat requests with premium models**; no VS Code, aponte as tarefas utilitárias para um modelo leve nas settings.
4. Aponte o **utility model** (geração de título, detecção de intenção, mensagem de commit) para um modelo leve.
5. Ajuste o **reasoning effort** no seletor conforme a complexidade da tarefa.
6. Na organização, defina a política de quais modelos os membros podem usar.

REGRA DE ROTEAMENTO (APLIQUE NO .AGENT.MD OU NA POLÍTICA)

```
trivial   → leve / incluído, sem reasoning
padrão    → mid-tier
complexo  → fronteira, reasoning alto
nunca     → fronteira p/ formatar, renomear, docs
```

EXEMPLOS DE CLASSE (CONFIRA A LISTA VIVA, OS NOMES MUDAM)

Os nomes de modelo mudam rápido, então pense por classe e confirme na referência AI model comparison. Em junho de 2026, de forma ilustrativa: a classe **econômica**, frequentemente incluída, inclui o modelo padrão do chat (GPT-4.1) e variantes da família mini; a **intermediária** são modelos de uso geral de custo médio; a de **fronteira ou raciocínio** inclui os modelos mais novos das famílias Claude e GPT-5, para refatoração difícil e tarefas multi-passo.

EXEMPLO

Geração de mensagem de commit e títulos de sessão é trabalho de modelo leve. Roteá-los para o utility model leve, e deixar o Auto cuidar do chat, tira esse tráfego dos modelos caros sem o desenvolvedor pensar a respeito. A fronteira fica reservada para refatoração difícil e raciocínio multi-passo.

COMO VERIFICAR

O seletor mostra indicadores de custo por modelo. O dashboard de uso mostra o breakdown por modelo ao longo do tempo, confirmando a migração para Auto e modelos mais leves.

ERROS COMUNS

- Achar que trocar o modelo do chat muda o das inline suggestions: são independentes.
- Deixar o premium ligado para requests fora do chat, pagando caro por trabalho de fundo.
- Esquecer que Extensions podem sobrepor o modelo escolhido.

## Referências oficiais

GITHUB DOCS
**Changing the AI model for Copilot Chat**
https://docs.github.com/en/copilot/how-tos/use-ai-models/change-the-chat-model

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

MICROSOFT LEARN
**GitHub Copilot usage and models**
https://learn.microsoft.com/en-us/visualstudio/ide/copilot-usage-and-models?view=visualstudio

GITHUB DOCS
**Requests in GitHub Copilot**
https://docs.github.com/en/copilot/concepts/billing/copilot-requests

PESQUISA · STANFORD (TMLR)
**FrugalGPT: cascata de modelos iguala o melhor modelo com economia de 50 a 98%**
https://arxiv.org/abs/2305.05176

PESQUISA · ROUTERBENCH
**RouterBench: roteadores multi-modelo igualam ou superam o melhor modelo único a custo médio menor**
https://arxiv.org/abs/2403.12031

PARTE 1 · POR ONDE COMEÇAR DEV ADMIN

# R3 · Modelos locais (Ollama e Foundry Local)

## Por que isso importa

Rodar a rotina em um modelo que executa na sua máquina custa zero por token e não consome AI Credits. É a ponta zero da escada de roteamento do R2, não uma alavanca separada que se soma. O valor está em mover o trabalho repetitivo e barato para fora da conta medida, liberando orçamento para o trabalho de fronteira, e em privacidade. Confirmado para Business e Enterprise no VS Code, por política de admin.

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 do BYOK). Participação no custo total: um dígito a ~15%, estimativa direcional dependente do mix.

## Como funciona: as duas superfícies de BYOK

Há duas superfícies distintas. A primeira é o **BYOK no editor** (política do VS Code), que suporta modelos locais via Ollama e Foundry Local; os modelos ficam disponíveis no chat, no plan agent e em agentes customizados. A segunda é o BYOK administrado por chave de provedor em nuvem. Este procedimento foca o caminho local no editor. Em todos os casos, BYOK **não vale para code completions**, e busca semântica e embeddings ainda exigem conta GitHub.

OBJETIVO

Registrar um modelo local no VS Code (Ollama ou Foundry Local) e roteá-lo para o trabalho de rotina.

PRÉ-REQUISITOS

Política `Bring Your Own Language Model Key in VS Code` habilitada (ligada por padrão; o admin pode desligar). Para certos modelos, `Editor Preview Features` habilitado pelo admin. `VS Code 1.113+`, `Copilot Chat 0.41.0+`. Para agent mode, use modelos com tool calling (Llama 3.1+, Mistral Nemo, Phi-4).

## Caminho A · Ollama

1. Confirme com o admin as políticas (BYOK e, se aplicável, Editor Preview Features).
2. Instale o Ollama (`0.18.3+`) e baixe um modelo com tool calling, por exemplo `llama3.1`.
3. Registre no VS Code: rode `ollama launch vscode`, ou manualmente abra o Copilot Chat, ícone de engrenagem, **Language Models**, **Add Models**, **Ollama** (endpoint `http://localhost:11434`).
4. No editor de Language Models, dê **unhide** nos modelos; filtre por capability (tool calling ou agent) se necessário.
5. Selecione **Local** no rodapé do painel do Copilot Chat e roteie a rotina para o modelo local.

```
TERMINAL · OLLAMA

# requisitos: VS Code 1.113+, Copilot Chat 0.41.0+, Ollama 0.18.3+
ollama pull llama3.1
ollama launch vscode
# manual: Chat: Manage Language Models
#   → Add Models → Ollama → http://localhost:11434
```

## Caminho B · Foundry Local

1. Instale o Foundry Local pelo gerenciador do seu sistema (bloco abaixo) e confirme com `foundry --version`.
2. Liste e baixe um modelo de código; na primeira vez, o Foundry Local baixa os execution providers do seu hardware. `Phi-4` é um bom começo.
3. Rode o modelo com `foundry model run phi-4`; opcionalmente suba o serviço e confira a porta dinâmica com `foundry service status`.
4. No VS Code, instale a extensão **AI Toolkit** (`aka.ms/aitoolkit`) e a extensão GitHub Copilot.
5. No Copilot Chat, abra o model picker, **Add model**, e escolha o provedor **Foundry Local via AI Toolkit**; o AI Toolkit baixa o modelo se preciso. Selecione-o e roteie a rotina.

```
TERMINAL · FOUNDRY LOCAL

# Windows
winget install Microsoft.FoundryLocal
# macOS
brew tap microsoft/foundrylocal && brew install foundrylocal
foundry --version
foundry model list
foundry model run phi-4
foundry service status   # confirma a porta dinamica
```

QUANTO ISSO CORTA DE VERDADE

O impacto é real, mas limitado. Mover a rotina para o local zera o custo dessa fatia, mas o gasto medido se concentra no trabalho agêntico de fronteira, que você não move para o local. Como tarefas de rotina já são baratas por tarefa, a economia em participação do custo total tende a ser de um dígito a cerca de 15%, dependendo do mix e do hardware. Esta participação é uma estimativa direcional de campo, não um benchmark publicado; a mecânica (não consumir AI Credits) é oficial. O maior ganho é privacidade e folga de rate limit, mais orçamento livre para a fronteira.

COMO VERIFICAR

O modelo local aparece no model picker. Com ele selecionado, as respostas são processadas inteiramente na sua máquina. Em agent mode, só aparecem modelos com tool calling.

ERROS COMUNS

- Esperar que BYOK reduza o custo de completions: não, completions seguem incluídas e fora do BYOK.
- Escolher um modelo sem tool calling e ele não aparecer no agent mode.
- Tratar local como alavanca que se soma ao routing: ela é a ponta zero do próprio routing.

## BYOK enterprise e a arquitetura de transbordo

A segunda superfície ganhou plano de controle: desde novembro de 2025 (public preview, ampliado em janeiro de 2026), o admin conecta a key do provedor nas settings da enterprise ou da organização, decide quais organizações enxergam cada modelo, e os modelos ficam disponíveis no Copilot Chat do GitHub.com e nas IDEs suportadas, cobrados direto no provedor e sem consumir AI Credits. É isso que permite desenhar o transbordo: quando o pool acaba, a conta não precisa ir para o metered.

> NÃO EXISTE GATILHO AUTOMÁTICO
>
> Nenhum budget ou cost center redireciona consumo: budgets param, não roteiam, e o fallback automático para modelo mais barato foi descontinuado em 2026-06-01. A troca de faixa é sempre seleção de modelo. A arquitetura abaixo aproxima o comportamento com os controles que existem.

**Duas faixas e um muro**
Pool incluído, hard stop do ULB e a faixa BYOK que sobrevive ao bloqueio.


### As cinco camadas, na ordem de implantação:

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.

COMO VALIDAR ANTES DE PROMETER

Piloto numa organização de teste: budget baixo num usuário, estourar de propósito, confirmar que o modelo BYOK segue respondendo após o bloqueio e que o consumo aparece no cost management do provedor, não no billing do GitHub. Só então levar o desenho para proposta.

## Referências oficiais

VS CODE DOCS
**AI language models in VS Code**
https://code.visualstudio.com/docs/agent-customization/language-models

GITHUB CHANGELOG
**Bring your own language model key in VS Code now available**
https://github.blog/changelog/2026-04-22-bring-your-own-language-model-key-in-vs-code-now-available/

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

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

MICROSOFT FOUNDRY BLOG
**AI-Assisted Development powered by Local Models**
https://devblogs.microsoft.com/foundry/ai-assisted-development-powered-by-local-models/

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: Bedrock, Google AI Studio, Responses API, max context window (2026-01-15)**
https://github.blog/changelog/2026-01-15-github-copilot-bring-your-own-key-byok-enhancements/

GITHUB DOCS
**Copilot SDK, bring your own key**
https://docs.github.com/en/copilot/how-tos/copilot-sdk/authenticate-copilot-sdk/bring-your-own-key

PARTE 1 · POR ONDE COMEÇAR DEV ADMIN

# R4 · Escopo de contexto e content exclusion

## Por que isso importa

O contexto é a maior parcela do input que você paga. Cada arquivo, cada diff, cada trecho de codebase que entra no prompt é token cobrado. Mandar só o que importa, e impedir que ruído e arquivos sensíveis entrem, corta o input de forma confiável.

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) comprimiu prompts em até 20x com perda de ~1,5 ponto; a banda usada é conservadora ante esse teto.

## Como funciona: o contexto é montado

No VS Code, o prompt é montado de várias fontes: sua mensagem, o histórico, o conteúdo de arquivos, a saída de ferramentas e as instruções. O editor já inclui implicitamente o arquivo ativo e a sua seleção. Você controla o resto com **#-mentions**. A sintaxe antiga `@workspace` foi substituída; hoje se usa `#codebase` para busca no workspace.

INVENTÁRIO DE #-MENTIONS

`#file` referencia um arquivo, `#folder` uma pasta, `#sym` um símbolo, `#changes` os diffs em source control, `#codebase` uma busca no workspace, `#terminalSelection` a saída do terminal, `#fetch` o conteúdo de uma URL. Use o específico em vez do amplo sempre que der.

OBJETIVO

Reduzir o input enviado e impedir que arquivos irrelevantes ou sensíveis entrem no contexto.

PRÉ-REQUISITOS

Para content exclusion, plano Business ou Enterprise e papel de admin de repositório, organização ou enterprise.

## Passos detalhados

1. No chat do VS Code, escope com **#-mentions**: prefira `#file` e `#sym` a jogar o `#codebase` inteiro.
2. Habilite a busca de código (setting `github.copilot.chat.codesearch.enabled`) para melhores resultados do `#codebase`.
3. Anexe só os arquivos necessários e remova o que não ajuda; lembre que o arquivo ativo e a seleção já entram sozinhos.
4. Para excluir arquivos do Copilot (sensíveis ou ruído), configure **Content exclusion**: no repositório, `Settings > Copilot > Content exclusion`; ou na organização, `Settings > Copilot > Content exclusion`. Liste os caminhos no formato YAML.

5 Aguarde até 30 minutos para propagar, ou recarregue as settings de content exclusion no IDE. Teste perguntando sobre um arquivo excluído.

```
CONTENT EXCLUSION (PATHS NAS SETTINGS)

"*":
  - "**/.env"
  - "**/secrets/**"
  - "dist/**"
  - "**/*.lock"
```

EXEMPLO

Cortar o contexto de uma consulta de cerca de 100 mil para cerca de 20 mil tokens reduz o input daquela chamada em torno de 80%. Em uma base grande, a diferença entre referenciar dois arquivos com `#file` e despejar o `#codebase` inteiro é a diferença entre uma resposta barata e uma cara.

COMO VERIFICAR

No Chat, pergunte sobre um arquivo excluído: o Copilot responde que não pode por causa de regra de content exclusion. No `#codebase`, o conteúdo excluído não entra na resposta.

ERROS COMUNS

- Usar content exclusion como controle de contexto de agente: ela **não é suportada em Edit e Agent mode**.
- Esquecer que content exclusion **bloqueia completions** nos arquivos afetados.
- Continuar usando `@workspace`: é sintaxe antiga; use `#codebase`.

## Referências oficiais

VS CODE DOCS
**Manage context for AI**
https://code.visualstudio.com/docs/copilot/chat/copilot-chat-context

GITHUB DOCS
**Excluding content from GitHub Copilot**
https://docs.github.com/en/copilot/how-tos/configure-content-exclusion/exclude-content-from-copilot

GITHUB DOCS
**Content exclusion (concept)**
https://docs.github.com/en/copilot/concepts/context/content-exclusion

MICROSOFT LEARN
**Manage content exclusions**
https://learn.microsoft.com/en-us/training/modules/github-copilot-management-and-customizations/4-manage-content-exclusions

PESQUISA · MICROSOFT (EMNLP 2023)
**LLMLingua: compressão de prompt de até 20x com perda de ~1,5 ponto**
https://arxiv.org/abs/2310.05736

PESQUISA · SURVEY
**A Survey of Context Engineering for Large Language Models**
https://arxiv.org/abs/2507.13334

PARTE 1 · POR ONDE COMEÇAR DEV

# R5 · Cache e memória

## Por que isso importa

Token em cache custa de 10 a 50% do token novo, conforme o modelo (na Anthropic a leitura de cache custa 0,1x o input; na OpenAI, 50%, automático acima de 1.024 tokens de prefixo), e a memória para de reexplicar o mesmo contexto a cada sessão. São dois ganhos compostos: o cache barateia o que se repete dentro da sessão; a memória elimina a reexplicação entre sessões.

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. Preço oficial: leitura de cache a 0,1x na Anthropic e 50% na OpenAI; a literatura KV (ChunkKV, KVCompose) sustenta 70 a 90% de compressão.

## Como funciona o cache

A infraestrutura do Copilot Chat cacheia o contexto repetido dentro de uma sessão. O cache acerta quando o **prefixo estável** não muda: se você mantém `copilot-instructions.md` e `AGENTS.md` no topo e não os edita no meio da conversa, eles caem em cache e o custo da repetição despenca. Fragmentar o trabalho em muitas sessões curtas, cada uma reenviando o contexto cheio, joga contra o cache.

OBJETIVO

Estabilizar o contexto para que ele seja cacheado, e usar memória para cortar a reexplicação entre sessões.

PRÉ-REQUISITOS

Copilot Memory (public preview) disponível nos planos pagos; é usado por Copilot cloud agent, code review e CLI. O code review usa apenas fatos de repositório, não preferências de usuário.

## Passos detalhados

1. Coloque o contexto estável (`copilot-instructions.md`, `AGENTS.md`) no topo e não o edite no meio da sessão; thrash quebra o cache.
2. Agrupe perguntas relacionadas na mesma sessão; cada sessão nova reenvia o contexto e perde o cache.
3. Mantenha um `AGENTS.md` enxuto com arquitetura, convenções e comandos de build e test (bloco abaixo).
4. Habilite e cure o **Copilot Memory**; revise periodicamente os fatos e preferências capturados.
5. Não rerode code review nem agentes sobre um diff que não mudou.

AGENTS.MD (ESTÁVEL, NO TOPO, PARA CACHEAR)

```
## Contexto
2 a 3 linhas de arquitetura.
## Convenções
o que o time prefere e evita.
## Build e validação
build: <comando>
test:  <comando>
```

EXEMPLO

Numa sessão longa de refatoração, manter o mesmo prefixo de instruções e AGENTS.md faz com que o segundo, terceiro e enésimo turnos paguem uma fração do input estável (de 10 a 50%, conforme o modelo). Em uma sequência de dez turnos, isso é uma economia grande e invisível, que some se você ficar editando os arquivos no meio do caminho.

COMO VERIFICAR

Respostas repetidas na mesma sessão ficam mais baratas (efeito do cache). Com memória ligada, o Copilot deixa de pedir o mesmo contexto que você já forneceu antes.

ERROS COMUNS

- Editar o prefixo estável no meio da sessão e quebrar o cache sem perceber.
- Fragmentar tudo em sessões curtas, reenviando o contexto cheio toda vez.
- Rerodar code review ou agentes sobre um diff que não mudou.

## Referências oficiais

GITHUB DOCS

**About GitHub Copilot Memory**

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

GITHUB DOCS

**Your first custom instructions (AGENTS.md)**

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

ANTHROPIC DOCS

**Prompt caching (leitura de cache a 0,1x o input)**

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

OPENAI DOCS

**Prompt caching (50% automático acima de 1.024 tokens)**

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

PARTE 1 · POR ONDE COMEÇAR DEV

# R6 · Primitivos de repositório

## Por que isso importa

Os primitivos transformam o Copilot de chat genérico em ferramenta governada. A maioria dos times improvisa, e esse improviso é um grande ofensor de custo. Cada primitivo é, ao mesmo tempo, governança e economia, sem trade-off: fazer o básico bem feito melhora o resultado e reduz a conta ao mesmo tempo.

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. Disciplina individual vira infraestrutura versionada.

IMPACTO ESPERADO

Ganho composto: cada arquivo reduz reexplicação e escopo em todo pedido futuro. É o que transforma os cortes das outras alavancas em padrão permanente.

## Os primitivos e o que cada um economiza

| PRIMITIVO | ONDE FICA | PARA QUE | ECONOMIZA |
|---|---|---|---|
| **Instruções de repositório** | `.github/copilot-instructions.md` | contexto geral, sempre ligado | entrada reexplicada |
| **Instruções com escopo** | `.github/instructions/*.instructions.md` | regras por área (applyTo) | contexto irrelevante |
| **Agente customizado** | `.github/agents/*.agent.md` | um agente, uma tarefa, modelo fixo | agente vagando, modelo errado |
| **Prompt reutilizável** | `.github/prompts/*.prompt.md` | encapsula instrução longa | erro e digitação |
| **Hook** | `pasta hooks/` | impede loops e ações desnecessárias | loops e ações à toa |
| **Skill** | `pasta com SKILL.md` | conhecimento sob demanda | reexplicar como fazer |

OBJETIVO

Versionar instruções, agentes, prompts e hooks no repositório, tratando configuração de IA como infraestrutura, revisada como qualquer mudança.

PRÉ-REQUISITOS

Acesso de escrita ao repositório; VS Code para os slash commands de geração.

## Passos detalhados

1. Comece pelo `copilot-instructions.md` (contexto geral, terse). É o primeiro e mais importante.
2. Adicione instruções com escopo por área em `.github/instructions/*.instructions.md` (campo `applyTo`).

3. Crie agentes customizados em `.github/agents/*.agent.md`: um agente, uma tarefa, com modelo e ferramentas restritos (bloco abaixo).
4. Encapsule comandos longos em prompt files `.github/prompts/*.prompt.md`.
5. Use hooks (pasta `hooks/`) para impedir loops e ações desnecessárias.
6. No VS Code, gere os arquivos com `/create-instruction`, `/create-agent`, `/create-prompt`, `/create-skill`, `/create-hook`.

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

```
---
description: 'Revisa PRs por correção e custo de IA.'
name: 'Cost-Aware Reviewer'
tools: ['read', 'search']
model: 'auto'
target: 'vscode'
---
Revise só o diff referenciado; não expanda o
contexto além dos arquivos alterados.
Responda em lista curta, sem preâmbulo.
```

EXEMPLO: POR QUE O AGENTE ECONOMIZA

Um agente com escopo (revisar só o diff) e modelo fixo (`auto`) evita dois desperdícios: o agente vagar pela base puxando contexto à toa, e a tarefa cair num modelo de fronteira sem necessidade. Escopo e governança de modelo, juntos, no mesmo arquivo versionado.

COMO VERIFICAR

O agente aparece no seletor de agentes do VS Code. As instruções com escopo são aplicadas automaticamente aos arquivos que casam o `applyTo`.

ERROS COMUNS

- Despejar tudo num arquivo gigante em vez de manter uma fonte única e enxuta.
- Snapshot velho: instruções que não acompanham a evolução do repositório.
- Prompt mágico: depender de um prompt longo decorado em vez de um prompt file versionado.

## Referências oficiais

VS CODE DOCS

**Customize AI in Visual Studio Code**

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

GITHUB

**github/awesome-copilot**

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

GITHUB DOCS

**Your first custom instructions**

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

PARTE 1 · POR ONDE COMEÇAR ADMIN

# R7 · Budgets e medição

## Por que isso importa

Caps são guard-rail de variância, não cortam o regime de consumo: eles evitam surpresas, não substituem as alavancas. O budget de usuário universal é o controle mais importante. E a medição contínua é o que transforma a estimativa em uma banda que aperta com os dados reais.

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. ULB em GA desde 2026-06-01; sem ele, um loop de agente pode consumir o pool da unidade sozinho.

**Os quatro níveis de budget**
Do hard stop individual ao failsafe da empresa.

OS QUATRO NÍVEIS DE BUDGET

**Do indivíduo ao failsafe: cada nível segura uma variância diferente.**

NÍVEL 1 **User-level budget (ULB)**
Hard stop por usuário por ciclo. O controle mais importante; em GA desde 2026-06-01.

NÍVEL 2 **Cost center budget**
Limita o metered de uma unidade de negócio após o pool; aloca custo por unidade.

NÍVEL 3 **Organization budget**
Acompanha o gasto da organização ou de repositórios específicos.

NÍVEL 4 **Enterprise spending limit**
Failsafe global: o teto do metered da empresa inteira.

Caps são disjuntor de variância, não economizador de regime.
Ao bloquear, não há fallback automático para modelo mais barato; completions seguem funcionando.

## Os quatro níveis de budget

| CONTROLE | ESCOPO | O QUE FAZ |
|---|---|---|
| **User-level budget (ULB)** | `por usuário, por ciclo` | hard stop; o controle mais importante; ativo no pool e no metered |
| **Cost-center budget** | `grupo de usuários` | limita o metered de uma unidade após o pool |
| **Enterprise spending limit** | `toda a enterprise` | limita o metered total após o pool |
| **Organization budget** | `organização ou repo` | acompanha o gasto da organização |

Cada licença inclui AI Credits agrupados num pool da enterprise: 1.900 créditos por usuário em Business e 3.900 em Enterprise (na promoção até 2026-09-01, 3.000 e 7.000). Quando o pool acaba, o uso adicional é cobrado a `US$ 0,01` por crédito, sujeito aos budgets. Code completions e Next Edit Suggestions seguem funcionando mesmo com budget esgotado, pois não consomem créditos.

OBJETIVO

Pôr budgets antes do uso crescer e medir continuamente para sustentar o rollout.

PRÉ-REQUISITOS

Papel de owner de enterprise ou organização, ou billing manager. Para budgets que bloqueiam o metered, o overage precisa estar habilitado.

## Passos detalhados

1. Vá a `Settings > Billing > Budgets and alerts` e clique em **New budget**.
2. Em Budget Type, escolha **Bundled AI credits budget** (cobre Copilot, Copilot cloud agent e Spark).
3. Defina o **budget de usuário universal (ULB)** acima do valor por licença (`US$ 19` Business, `US$ 39` Enterprise); ele aplica automaticamente a todo usuário licenciado. Os budgets por usuário estão em disponibilidade geral desde 2026-06-01. Com o BYOK enterprise conectado, o bloqueio do ULB vira a fronteira da faixa de transbordo: o desenho completo está em R3.
4. Revise o dashboard de uso e crie **overrides por usuário** para os consumidores pesados, sem afetar os demais.
5. Para múltiplas unidades, use cost centers com budget próprio e um enterprise budget como failsafe.
6. Ative alertas em `75`, `90` e `100%`, e **Stop usage when budget limit is reached** onde fizer sentido.

CONFIGURAÇÃO INICIAL RECOMENDADA

```
ULB universal > valor por licenca (19 / 39 USD)
overrides para consumidores pesados
cost center por unidade (opcional)
enterprise budget como failsafe
alertas em 75 / 90 / 100%
```

EXEMPLO DE DIMENSIONAMENTO

Se o consumo está concentrado num grupo pequeno, um ULB universal alto desperdiça; melhor um ULB moderado mais overrides para os pesados. Se o consumo é distribuído, o ULB universal mais cost centers por unidade é o suficiente. O dashboard de uso e o CSV são as ferramentas para decidir qual caso é o seu.

COMO VERIFICAR

O budget aparece em Budgets and alerts. Ao atingir o limite, o usuário é bloqueado para features que consomem AI Credits (completions seguem funcionando). Alertas chegam por email em 75, 90 e 100%.

ERROS COMUNS E O RELÓGIO DA PROMO

O período promocional (junho a agosto de 2026, com pool maior) termina em **1 de setembro de 2026**; use-o para achar a linha de base, e não o leia como o estado permanente. Ao bloquear, não há fallback automático para modelo mais barato: o usuário simplesmente para.

## Referências oficiais

GITHUB DOCS

**Setting up budgets to control spending**

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

**Budgets for usage-based billing**

https://docs.github.com/en/copilot/concepts/billing/budgets-for-usage-based-billing

GITHUB DOCS

**Usage-based billing for organizations and enterprises**

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

PARTE 2 · OPERAR E ITERAR

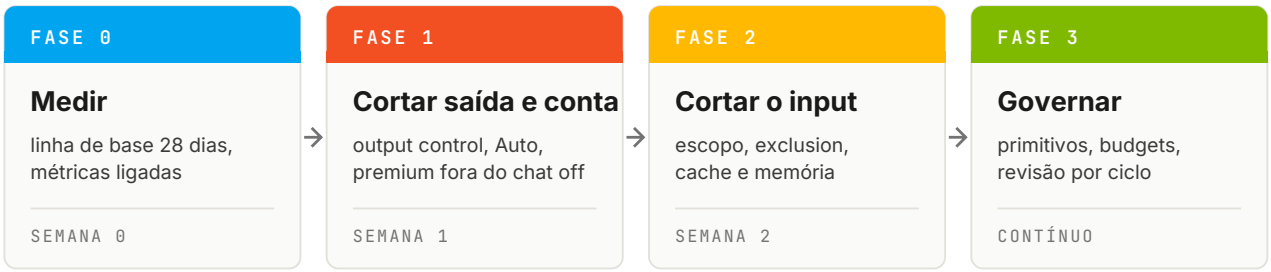
# Sequência de adoção recomendada

A ordem importa porque as alavancas se compõem. Esta é uma sequência de adoção pragmatica, do mais barato e de maior retorno para o mais estrutural. Cada fase deixa um ganho medível antes de passar à próxima.

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

A SEQUÊNCIA DE ADOÇÃO

**Em ordem de retorno: o que corta mais, primeiro.**




Começo enxuto: 20 a 30% já na Fase 1. Programa maduro, com as quatro fases operando: 55 a 70%.

## Fase 0 · Antes de tudo

- Ligue a observabilidade (R0.3) e registre a linha de base por usuário, modelo e feature.
- Alinhe papéis e políticas com o admin (R0.2).

## Fase 1 · Carregar o resultado (dias 1 a 3)

- **R1 Output control:** copilot-instructions.md com saída direta. Maior alavanca, custo quase zero.
- **R2 Model routing:** Auto por padrão, utility model leve, premium fora do chat desligado.

## Fase 2 · Cortar o input (semana 1)

- **R4 Escopo de contexto:** hábito de #-mentions e content exclusion para ruído e sensível.
- **R3 Modelos locais:** mover a rotina para Ollama ou Foundry Local onde fizer sentido.

## Fase 3 · Tornar permanente (semana 2)

- **R5 Cache e memória:** estabilizar o prefixo, ligar memória, higiene de sessão.
- **R6 Primitivos:** versionar instruções, agentes, prompts e hooks.

## Contínuo · Governar

- **R7 Budgets e medição:** ULB universal e alertas desde já; revisão a cada ciclo.
- Compare com a linha de base e ajuste antes do fim da promo, em 1 de setembro de 2026.

POTENCIAL COMBINADO

Os ganhos não se somam, se compõem sobre bases diferentes. Um programa maduro com todas as alavancas tende a uma faixa de 55 a 70%; um começo enxuto, de 20 a 30%. São bandas diretivas, que devem ser confirmadas contra os seus dados, não promessas.

PARTE 2 · OPERAR E ITERAR

# Operar, atribuir e tratar como banda

## O ciclo

A otimização não é um evento, é um ciclo. Depois de aplicar os procedimentos, opere e itere com base no dado real.

1. Releia o dashboard de uso a cada ciclo de cobrança e compare com a linha de base capturada na Parte 0.
2. Atribua o consumo por usuário, modelo e feature usando o CSV e o NDJSON de exportação.
3. Trate o ganho como **banda que aperta com os dados**, não como alvo fixo de ponto único.
4. Ajuste budgets e políticas conforme o consumo real, especialmente antes do fim do período promocional.
5. Reforce o que funciona pelos primitivos versionados; o que vira arquivo no repositório vira permanente.

> SINAL DE SAÚDE
>
> Consumo concentrado em poucos usuários pesados favorece overrides individuais em vez de um ULB universal alto. Consumo distribuído favorece o ULB universal e cost centers por unidade. Deixe o dado decidir.

## Referências oficiais

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

> GITHUB DOCS
>
> **Optimizing your budget configuration**
>
> https://docs.github.com/en/copilot/tutorials/budgets/optimizing-your-budget-configuration

PARTE 2 · OPERAR E ITERAR

# Anti-padrões a evitar

Os erros mais comuns sob o UBB, reunidos. Evitá-los vale tanto quanto aplicar as alavancas.

- **Achar que completions consomem créditos.** Não consomem. Otimizar o tamanho de identificadores para economizar é esforço perdido; o custo está em chat e agente.
- **Ler os planos como ilimitados.** Desde 2026-06-01, Pro inclui 1.500 créditos por mês, Pro+ 7.000 e o novo Copilot Max (US$ 100) 20.000; nenhum é uso ilimitado de fronteira. Uma sessão agêntica longa de fronteira consome uma fatia grande de qualquer um deles.
- **Fragmentar a sessão.** Muitas sessões curtas reenviam o contexto cheio e perdem o desconto de cache; agrupe o trabalho relacionado.
- **Não definir budgets por usuário.** Um único loop de agente acidental pode queimar uma fatia grande do pool; o ULB é um seguro barato.
- **Ler a promo como o normal.** Junho a agosto de 2026 tem pool maior; o regime real vem depois de 1 de setembro de 2026.
- **Rerodar em diff inalterado.** Rodar code review ou agentes sobre o mesmo diff gasta à toa, e code review ainda consome Actions minutes.
- **Despejar o codebase inteiro.** Use #-mentions específicos; o `#codebase` amplo infla o input.
- **Thrash do contexto estável.** Editar copilot-instructions.md ou AGENTS.md no meio da sessão quebra o cache.
- **Instruções conflitantes.** A escolha entre conflitos é não determinística; mantenha uma fonte única e enxuta.
- **Usar @workspace.** É sintaxe antiga; o VS Code atual usa `#codebase`.

## Referências oficiais

GITHUB DOCS
**Requests in GitHub Copilot**
https://docs.github.com/en/copilot/concepts/billing/copilot-requests

GITHUB DOCS
**Budgets for usage-based billing**
https://docs.github.com/en/copilot/concepts/billing/budgets-for-usage-based-billing

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

APÊNDICES

# A1 · Mapa de links oficiais

Todos os links citados ao longo do runbook, agrupados por fonte. Clicáveis no PDF.

## GITHUB DOCS

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

**Setting up budgets to control spending**
https://docs.github.com/en/billing/how-tos/set-up-budgets

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

**Budgets for usage-based billing**
https://docs.github.com/en/copilot/concepts/billing/budgets-for-usage-based-billing

**Optimizing your budget configuration**
https://docs.github.com/en/copilot/tutorials/budgets/optimizing-your-budget-configuration

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

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

**Changing the AI model for Copilot Chat**
https://docs.github.com/en/copilot/how-tos/use-ai-models/change-the-chat-model

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

**Requests in GitHub Copilot**
https://docs.github.com/en/copilot/concepts/billing/copilot-requests

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

**Content exclusion (concept)**
https://docs.github.com/en/copilot/concepts/context/content-exclusion

**About GitHub Copilot Memory**
https://docs.github.com/en/copilot/concepts/agents/copilot-memory

**GitHub Copilot usage metrics**
https://docs.github.com/en/copilot/concepts/copilot-usage-metrics/copilot-metrics

**Viewing the Copilot usage metrics dashboard**
https://docs.github.com/en/copilot/how-tos/administer-copilot/view-usage-and-adoption

## VS CODE DOCS

**Customize AI in Visual Studio Code**
https://code.visualstudio.com/docs/agent-customization/overview

**AI language models in VS Code**
https://code.visualstudio.com/docs/agent-customization/language-models

**Manage context for AI**
https://code.visualstudio.com/docs/copilot/chat/copilot-chat-context

### MICROSOFT LEARN

**GitHub Copilot usage and models**
https://learn.microsoft.com/en-us/visualstudio/ide/copilot-usage-and-models?view=visualstudio

**Manage content exclusions (módulo)**
https://learn.microsoft.com/en-us/training/modules/github-copilot-management-and-customizations/4-manage-content-exclusions

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

### GITHUB CHANGELOG E BLOG

**GitHub Copilot is moving to usage-based billing**
https://github.blog/news-insights/company-news/github-copilot-is-moving-to-usage-based-billing/

**Bring your own language model key in VS Code now available**
https://github.blog/changelog/2026-04-22-bring-your-own-language-model-key-in-vs-code-now-available/

**Updates to GitHub Copilot billing and plans**
https://github.blog/changelog/2026-06-01-updates-to-github-copilot-billing-and-plans/

**Copilot metrics is now generally available**
https://github.blog/changelog/2026-02-27-copilot-metrics-is-now-generally-available/

**AI-Assisted Development powered by Local Models (Microsoft Foundry)**
https://devblogs.microsoft.com/foundry/ai-assisted-development-powered-by-local-models/

### PESQUISA ACADÊMICA E BENCHMARKS

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

**RouterBench: benchmark de roteamento multi-modelo**
https://arxiv.org/abs/2403.12031

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

**A Survey of Context Engineering for LLMs (2025)**
https://arxiv.org/abs/2507.13334

**Peng et al. (2023): impacto do GitHub Copilot na produtividade (55,8% mais rápido)**
https://arxiv.org/abs/2302.06590

**ChunkKV (2025): 10% do cache KV com queda de ~2,3% no LongBench**
https://arxiv.org/abs/2502.00299

**KVCompose (2025): compressão estruturada de 70 a 80% sob tolerância de 10 a 20%**
https://arxiv.org/abs/2509.05165

### PROVEDORES (PREÇO E CACHE)

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

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

### OLLAMA DOCS

> **VS Code integration**
> https://docs.ollama.com/integrations/vscode

APÊNDICES

# A2 · Treinamento para aprofundar

Para aprofundar e formar o time. Conteúdo oficial da Microsoft Learn e do GitHub.

MICROSOFT LEARN

**Get Started with GitHub Copilot (módulo)**

https://learn.microsoft.com/en-us/training/modules/get-started-github-copilot/

MICROSOFT LEARN

**Accelerate app development using GitHub Copilot (trilha)**

https://learn.microsoft.com/en-us/training/paths/accelerate-app-development-using-github-copilot/

MICROSOFT LEARN

**GitHub Copilot Fundamentals (trilha)**

https://learn.microsoft.com/en-us/training/paths/gh-copilot-2/

MICROSOFT LEARN

**Curso GH-300T00: GitHub Copilot**

https://learn.microsoft.com/en-us/training/courses/gh-300t00

MICROSOFT LEARN

**Certificação GitHub Copilot**

https://learn.microsoft.com/en-us/credentials/certifications/github-copilot/

MICROSOFT LEARN

**AI Toolkit para VS Code (Foundry Local)**

https://aka.ms/aitoolkit

GITHUB

**github/awesome-copilot (catálogo validado)**

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

APÊNDICES

# A3 · Glossário

Termos usados neste runbook.

- **AI Credit.** Unidade de cobrança do UBB. 1 AI Credit = US$ 0,01.
- **Token.** Unidade de texto processada. Há tokens de entrada, de saída e de cache; o custo é por token, conforme o modelo.
- **Completions / NES.** Code completions e Next Edit Suggestions; incluídos no plano, não consomem AI Credits.
- **Pool.** Conjunto de AI Credits agrupados na enterprise; quando esgota, o uso adicional é metered.
- **ULB.** User-level budget, o budget de usuário; o controle mais importante, com hard stop por ciclo, ativo no pool e no metered.
- **Cost center.** Agrupamento de usuários com budget próprio, para alocar custo por unidade de negócio.
- **Content exclusion.** Mecanismo de settings (Business e Enterprise) que impede o Copilot de acessar arquivos; não vale em Edit e Agent mode, e bloqueia completions nos arquivos afetados.
- **BYOK.** Bring Your Own Key; usar chave ou modelo próprio (inclusive local) no chat do VS Code; não vale para completions.
- **Primitivo.** Arquivo versionado de configuração: instruções, instruções com escopo, agentes, prompts, hooks e skills.
- **Auto.** Seleção automática de modelo; escolhe por disponibilidade e dá 10% de desconto no chat para planos pagos.
- **Reasoning effort.** Controle no seletor de modelo que equilibra profundidade de raciocínio, latência e custo.
- **Utility model.** Modelo leve para tarefas utilitárias como geração de título, detecção de intenção e mensagem de commit.
- **#-mentions.** Sintaxe atual do VS Code para escopar contexto (#file, #folder, #sym, #changes, #codebase); substitui o antigo @workspace.
- **Cache de prompt.** Reuso do contexto repetido dentro da sessão; o token em cache custa de 10 a 50% do token novo, conforme o modelo (Anthropic 0,1x; OpenAI 50%).
- **Tool calling.** Capacidade de o modelo chamar ferramentas (ler arquivo, rodar comando, buscar). É necessária para o agent mode; modelos sem tool calling não aparecem no picker de agente.
- **Agent mode.** Modo em que o Copilot executa tarefas de forma autônoma no editor, lendo arquivos, rodando comandos e editando código em vários passos.
- **Cloud agent.** O Copilot coding agent que executa na nuvem do GitHub, fora do editor, para tarefas atribuídas; usa o Copilot Memory.
- **Metered.** Fase de cobrança por uso, após o pool esgotar, a US$ 0,01 por crédito adicional, sujeita aos budgets.

Comece medindo, aplique na ordem, e deixe a banda apertar com os dados reais. Otimização sem medição é palpite.

FIM DO RUNBOOK · PAULA SILVA, SOFTWARE GLOBAL BLACK BELT