GUIA DE OTIMIZAÇÃO · USAGE-BASED BILLING · GITHUB COPILOT

Otimização de custo no GitHub Copilot, as formas e o potencial.

Um guia das formas de reduzir o custo de token no modelo Usage-Based Billing do GitHub Copilot. O que cada forma faz, o máximo que alcança, por que funciona, e quanto dá para alcançar quando elas se compõem. O custo não se prevê com exatidão, ele se controla.

ESCOPO: 12 FORMAS DE OTIMIZAÇÃO DE TOKEN
BASE: DOCUMENTAÇÃO OFICIAL E LITERATURA DE CAMPO ·
ATUALIZADO: 2026-06-08

Paula Silva
SOFTWARE GLOBAL BLACK BELT
PAULASILVA@MICROSOFT.COM

DO CAMPO, SOFTWARE GLOBAL BLACK BELT

Um guia das formas de baixar o custo de token no GitHub Copilot, e do potencial que elas alcançam combinadas.

O guia organiza o material em dez capítulos mais um apêndice. Cada forma traz o máximo que alcança, sobre o que age e por que funciona, com as fontes citadas. Os números são tetos por classe de token, não somas. A equação que governa tudo: custo é igual a modelo escolhido vezes tokens consumidos.

NESTE GUIA

COMO LER ESTE GUIA

Comece pela equação no capítulo 01. O 02 explica por que as formas não se somam. O 03 lista o catálogo completo e o 04 o potencial combinado. Os capítulos 05 a 09 detalham cada grupo de formas, e o 10 traz a sequência de implementação.

A IDEIA CENTRAL

O custo depende do uso, então não se prevê com exatidão. A conversa certa é controlar o modelo e os tokens, com uma banda que aperta conforme os dados reais chegam.

CAPÍTULO 01 A EQUAÇÃO E A VIRADA

O custo mudou de assento fixo para uso medido.

A EQUAÇÃO QUE GOVERNA TUDO

CUSTO = modelo × tokens

LEITURA

Toda forma de otimização mexe em um destes dois fatores: troca o modelo por um mais barato e suficiente, ou reduz os tokens de entrada e de saída. Não há terceiro caminho.

O QUE MUDA NO UBB

Paga-se pelo uso

O custo passa a ser proporcional ao modelo escolhido e ao volume de tokens, não a um assento fixo.

POR QUE NÃO DÁ PRA PREVER

Depende do uso

O consumo varia com a forma como o time trabalha. A conversa certa não é prever o número exato, é controlar o que entra na conta.

A VIRADA

Controlar, não prever

Sai o orçamento fixo, entra o controle contínuo. As formas deste guia atuam nos dois fatores da equação.

POR QUE ISSO IMPORTA

Sem entender a equação, toda discussão de custo vira palpite. Com ela, cada forma de otimização ganha um lugar claro: ou ataca o preço do modelo, ou ataca o volume de tokens. O resto do guia organiza as formas por esses dois eixos.

CAPÍTULO 02 POR QUE NÃO SE SOMAM

Cada forma age sobre uma base diferente da equação.

% DA CONTA TODA

Model routing

É a única forma expressa sobre a fatura inteira, porque troca o preço do modelo em qualquer tarefa.

% DA SAÍDA

Output control

Age sobre os tokens de saída, a classe mais cara, cerca de 5x o input por token.

% DA ENTRADA

Contexto e cache

Escopo de contexto, cache e memória agem sobre os tokens de entrada. Outra base, outro peso.

COMPORTAMENTAL

Spending caps

Limites de gasto não cortam regime; cortam o pico do loop descontrolado e impõem disciplina.

POR QUE NÃO SE SOMAM

Cada forma incide sobre uma base diferente, e várias se sobrepõem. A composição é multiplicativa, não aditiva. Somar os máximos passaria de 100% e não se sustenta. O capítulo 04 mostra como compor de forma honesta.

CAPÍTULO 03 CATÁLOGO COMPLETO DE FORMAS

Toda forma, o máximo que alcança, e por que funciona.

O quadro abaixo é o universo das formas de otimização de token no GitHub Copilot: o máximo de cada uma, sobre o que age, e por que funciona.

FORMA	ATUA SOBRE	MÁXIMO	POR QUE FUNCIONA
Output control	saída	**40 a 70%**	saída custa ~5x o input; corta o token mais caro primeiro
Model routing	conta toda	**40 a 70% (60 a 80%/ agente)**	preço por token varia até 125x entre leve e fronteira
Modelos locais (Ollama, Foundry Local)	rotina	**custo zero**	ponta zero da escada de routing; via BYOK, não consome AI Credits
Scope de contexto	entrada	**40 a 80%**	contexto é 60%+ do input; mande só o arquivo relevante
Cache e semantic cache	entrada	**30 a 50%**	prefixo estável reusado não é reprocessado
Memory, instructions, AGENTS.md	entrada	**composto**	o que foi aprendido não é reexplicado a cada sessão
Prompt engineering	entrada e saída	**50 a 70% (saída)**	anti-padrões inflam token sem agregar sinal
Primitivos de repositório	entrada e conta	**governança = economia**	injeta contexto controlado e fixa o modelo certo
Planejador-executor	conta	**parte do 60 a 80%/ agente**	fronteira planeja 1x, modelo leve executa os passos
Higiene de sessão	entrada	**evita inflar**	histórico de chat longo é reenviado a cada turno
Content exclusion, .gitignore	entrada	**corte confiável**	build, gerados e logs não são contexto útil
Spending caps	comportamental	**sem % de literatura**	corta o pico do loop, não o regime; controla variância

COMO LER OS MÁXIMOS

Cada máximo é o teto da forma sobre a sua própria classe de token. Routing é da conta toda; output e prompt são da saída; contexto, cache e memória são da entrada. Por isso não se somam: agem em bases diferentes e várias se sobrepõem.

FONTES

GitHub Docs, Improving agent quality to optimize AI usage. RouterBench. Literatura de campo de 2026 sobre economia de token. Copilot Memory. FinOps Foundation e Datawiza para caps. Lista completa no apêndice.

CAPÍTULO 04 POTENCIAL COMBINADO

A composição multiplicativa, e a banda honesta que dá pra alcançar.

POTENCIAL DE UM PROGRAMA MADURO

55 a 70%

LEITURA

Redução total direcional, dependente de uso. Um começo enxuto, com poucas alavancas, já entrega da ordem de 20 a 30%; o catálogo completo, maduro, chega a 55 a 70%. Quanto maior a participação de modelos de fronteira no seu mix, mais o routing carrega o resultado. O teto da literatura de campo fica perto de 90%.

- A conta começa em **100**, a linha de base.
- **Routing** puxa o mix para modelos suficientes: corte típico de ~40% quando há bastante fronteira. A conta vai a **~60**.
- **Output control e prompt** cortam a saída, a classe mais cara: ~50% da saída. A conta vai a **~48**.
- **Contexto, cache e memória** cortam o input restante: ~45%. A conta vai a **~38**.
- Composto: **55 a 70%** num programa maduro; um começo enxuto entrega **~20 a 30%**. Aritmética de composição, compatível com o teto da literatura (FrugalGPT, 50 a 98%).

POR QUE MULTIPLICATIVO, NÃO ADITIVO

Cada forma age sobre o que sobrou da anterior, não sobre a conta cheia. Somar os máximos (que daria mais de 100%) é erro de leitura. A composição é uma cascata de fatores, e várias formas se sobrepõem, então o resultado fica abaixo da soma ingênua.

BANDA, NÃO PREVISÃO

O número é uma faixa direcional, não um alvo fixo. O custo depende do uso, então a régua honesta é uma banda que aperta conforme os dados reais de consumo chegam e a disciplina amadurece.

CAPÍTULO 05 OUTPUT, A MAIOR ALAVANCA

A classe de token mais cara, cortada primeiro.

A MAIOR ALAVANCA ISOLADA

Output primeiro

Saída é a classe de token mais cara, cerca de 5x o input. Cortar a explicação que ninguém vai ler remove o token mais caro primeiro.

COMO APLICAR

Code-only por padrão

No copilot-instructions.md, defina respostas diretas, sem preâmbulo nem explicação desnecessária. Vale por projeto e por prompt.

A ESCADA DE OUTPUT SPEC

- Sem especificação: **400 a 600** tokens de saída por resposta.
- Com 'seja direto, sem explicação': **100 a 150**.
- Com 'retorne só o código ou o diff': **30 a 50**.

REGRA PRÁTICA

Se você não vai ler a explicação, não peça. Cada ponto cortado na saída vale mais em dólar do que um ponto no input, porque saída é a classe mais cara da conta.

FONTE

GitHub Docs, Improving agent quality to optimize AI usage: escolha de modelo e arquivos de instrução reduzem token desperdiçado; code only, no explanation como default de projeto.

CAPÍTULO 06 MODEL ROUTING

Modelo certo para a tarefa certa.

DIFERENÇA DE PREÇO POR TOKEN

até 125x

LEITURA

Entre a classe mais leve e a de fronteira. Casar a tarefa com o modelo suficiente torna o custo proporcional ao valor, não ao hábito de sempre pegar o modelo mais forte.

O PRINCÍPIO

Modelo certo, tarefa certa

Reserve fronteira para o raciocínio difícil. Mande o trabalho mecânico para modelos leves ou incluídos. O dev troca manualmente quando justificar.

QUANTO RENDE

40 a 70% da conta

No nível da organização, 40 a 70%. Por agente, 60 a 80%. Um agente de CI movido para um modelo mini chega a 80 a 95%.

DEPENDE DO SEU MIX

O ganho do routing é proporcional à participação de fronteira no seu consumo. Quanto mais concentrado em modelos caros hoje, maior o espaço. RouterBench mostra variação de custo de 2x a 5x para performance comparável. A escada de preços por modelo está no apêndice. A ponta mais barata da escada é o modelo local, no próximo capítulo.

CAPÍTULO 07 MODELOS LOCAIS PARA ROTINA

Tarefa de rotina em modelo local, a custo zero de token.

CUSTO DE UMA TAREFA DE ROTINA EM MODELO LOCAL

0 crédito

LEITURA

Modelo local via BYOK no VS Code não consome AI Credits e não conta contra as cotas do GitHub Copilot. A cobrança, quando existe, é do seu provedor; local é zero por token. É a ponta mais barata da escada de routing: mande a rotina para o local e reserve a conta para a fronteira.

ONDE RODA

Ollama e Foundry Local

Aparecem direto no model picker do VS Code Chat, no plan agent e em agentes customizados. Confirmado para Business e Enterprise.

O QUE MANDAR PRA LÁ

Tarefas de rotina

Commit, boilerplate, perguntas simples, testes, refactor pequeno. BYOK cobre chat e utility tasks. Em agent mode, o modelo local precisa suportar tool calling.

O QUE FICA DE FORA

Completions e busca

Code completions seguem no backend do GitHub, incluídas e não afetadas. Busca semântica e embeddings ainda exigem conta GitHub.

QUANTO ISSO CORTA DE VERDADE

Local zera o custo do que você move pra ele, mas o dinheiro se concentra na fronteira, que você não move. Direcionalmente, mandar a rotina pro local corta da ordem de dígitos únicos a 15% do gasto faturável, conforme o mix e o hardware. Não é um corte de 40 a 70% à parte: é a ponta zero do routing, já dentro da banda do capítulo 04. O ganho maior costuma ser privacidade e liberar orçamento pra fronteira.

GATED PELO ADMIN

Em Business e Enterprise, a política de organização Bring Your Own Language Model Key in VS Code decide se isso é permitido. Vem habilitada por padrão e o admin pode desligar. Sem a política ligada, o caminho local não aparece para o desenvolvedor.

DUAS SUPERFÍCIES DE BYOK

Não confundir. No editor: a política do VS Code, que suporta modelo local, Ollama e Foundry Local, e é a que importa para rotina local. No enterprise: use your own API keys, em que o admin pluga chaves de provedores de nuvem; esse caminho exige a OpenAI Completions API, não a Responses API.

CAPÍTULO 08 CONTEXTO, CACHE E MEMÓRIA

Cortar o input que você paga a cada turno.

SCOPE DE CONTEXTO

40 a 80% do input

Contexto costuma ser 60%+ do input por turno. Mandar só o arquivo ou o intervalo relevante, em vez de todo o workspace, corta o que o modelo lê e você paga.

CACHE E SEMANTIC CACHE

30 a 50% do input

Prefixos estáveis reusados não são reprocessados. Em loops de agente e RAG, o token em cache custa fração do token novo: de 10 a 50%, conforme o modelo (Anthropic 0,1x; OpenAI 50%).

MEMÓRIA E INSTRUÇÕES

Ganho composto

copilot-instructions, AGENTS.md e memória param de reexplicar o mesmo contexto a cada sessão. O ganho cresce e vira permanente.

HIGIENE DE SESSÃO

Histórico de chat longo é reenviado a cada turno e chega a 20 a 50 mil tokens por turno só de histórico. Recomeçar a sessão e destacar conversas longas zera esse custo. E não rerode um agente sobre um diff que não mudou.

CAPÍTULO 09 PROMPT, PRIMITIVOS, GOVERNANÇA

O refino e os guard-rails que sustentam o regime.

PROMPT ENGINEERING

50 a 70% na saída

Anti-padrões como preâmbulo, repetição e pedir explicação passo a passo inflam token sem agregar sinal. Output spec corta direto.

PRIMITIVOS DE REPOSITÓRIO

Governança = economia

Instruções com escopo, agentes customizados, prompt files e hooks injetam contexto controlado e fixam o modelo certo, evitando o agente vagar.

SPENDING CAPS

Controle de variância

Limites por time não reduzem o regime, cortam o pico do loop descontrolado e impõem disciplina. É guard-rail de risco.

EVITAR O DOUBLE-COUNT

Várias dessas formas se sobrepõem: um copilot-instructions enxuto é, ao mesmo tempo, code-only e prompt engineering. Por isso os ganhos não se somam linearmente. Componha, não some, e atribua cada economia a uma fonte só.

CAPÍTULO 10 POR ONDE COMEÇAR

A sequência de maior retorno, do output aos caps.

- **1. Output primeiro.** Defina code-only e respostas diretas no copilot-instructions. Maior retorno, menor esforço.
- **2. Routing.** Reserve fronteira para o difícil e mande o resto para modelos suficientes. Se a política da org permitir, mande a rotina para um modelo local (Ollama, Foundry Local) a custo zero.
- **3. Contexto.** Escope arquivos, use .copilotignore, evite o workspace inteiro.
- **4. Instruções e memória.** Fixe o que se repete para parar de reexplicar.
- **5. Cache.** Para loops de agente e RAG, mantenha o contexto estável no topo.
- **6. Caps e medição.** Coloque limites como guard-rail e meça antes e depois.

MEÇA ANTES E DEPOIS

Otimização sem medição é palpite. Ligue a observabilidade de uso, registre o consumo por time e por modelo, e trate o resultado como uma banda que aperta com os dados reais, não como um alvo fixo.

A ORDEM IMPORTA

Output e routing são as alavancas que carregam o resultado. As demais refinam. Comece pelas duas de maior retorno e adicione disciplina por cima, na ordem acima.

EXECUTIVO KIT E DECISÕES

O kit pronto e as três decisões que destravam tudo.

O kit de artefatos (no playbook companheiro, prontos para colar)

- **copilot-instructions.md** com o bloco de saída: a maior alavanca isolada, um arquivo de texto commitado.
- **Instruções por área** (testes, docs) com escopo via applyTo, **AGENTS.md** como memória de processo cacheável.
- **Agente cost-aware-reviewer** com modelo e ferramentas restritos, **prompt de release notes** versionado.
- **Content exclusion** com os paths sensíveis e a **regra de roteamento** para o acordo do time.

As três decisões do admin

- **Políticas:** troca de modelo habilitada com a política de modelos fixada; BYOK confirmado; métricas de uso ligadas.
- **Guard-rail:** ULB universal acima do valor da licença (em GA desde 2026-06-01), overrides para os pesados, cost centers por unidade, failsafe da enterprise.
- **Cadência:** linha de base de 28 dias exportada e revisão por ciclo (usuário x modelo x feature) na agenda, comparando o real com a banda.

A faixa de transbordo (BYOK enterprise)

- **Key da empresa conectada** nas settings da enterprise (public preview): modelos do provedor no Chat do GitHub.com e nas IDEs, cobrados no contrato próprio, sem consumir AI Credits.
- **O ULB vira o muro:** ao bloquear, completions, modelos locais e BYOK seguem funcionando; o transbordo cai na taxa negociada do provedor em vez do metered de tabela.
- **Sem gatilho automático:** budgets param, não roteiam; a troca é seleção de modelo, fixável por agente. Medição em dois painéis: billing do GitHub + cost management do provedor.

> O PEDIDO PARA O TIME
>
> Fase 1 nesta semana: instruções commitadas, Auto padronizado, premium fora do chat desligado. É o suficiente para sentir a conta cair antes do fim do período promocional (2026-09-01), e a evidência que justifica as fases seguintes. Começo enxuto: 20 a 30%; programa maduro: 55 a 70%.

APÊNDICE PREÇOS E FONTES

A escada de preços por classe e as fontes citáveis.

Preços de roteamento, valores de referência

MODELO	INPUT (CR/1M)	OUTPUT (CR/1M)	MELHOR USO
Classe incluída (mini)	incluído	incluído	chat diário, completions
Modelo local (BYOK)	0	0	tarefas de rotina, sem AI Credits
Classe de implementação	125	1.000	código de média complexidade
Classe de contexto longo	300	1.500	contexto longo, trabalho sustentado
Classe de fronteira	1.500	7.500	agêntico difícil, uso reservado

SOBRE OS PREÇOS

Valores de referência em créditos por 1M de tokens, por classe de modelo. Os preços e os modelos exatos mudam ao longo do tempo; confirme os valores vigentes na documentação oficial do GitHub antes de usar em proposta.

Fontes

- **GitHub Docs**, Improving agent quality to optimize AI usage. Fonte primária para routing e copilot-instructions.
- **GitHub Changelog** (abril de 2026) e doc AI language models in VS Code. BYOK e modelos locais (Ollama, Foundry Local) em Business e Enterprise, governados pela política de organização; BYOK não se aplica a completions.
- **Pesquisa publicada**: FrugalGPT (Stanford, TMLR, arxiv.org/abs/2305.05176), cascata iguala o melhor modelo com economia de 50 a 98%; RouterBench (arxiv.org/abs/2403.12031); LLMLingua (Microsoft, EMNLP 2023, arxiv.org/abs/2310.05736), compressão de até 20x.
- **Preço oficial dos provedores**: leitura de cache a 0,1x na Anthropic e 50% na OpenAI; razão saída/entrada de ~4 a 5x; tabela por modelo do GitHub (AI model comparison).
- **FinOps Foundation** e Datawiza. Caps e controles de uso como guard-rail de risco.

MICROSOFT + GITHUB

O custo não se prevê com exatidão, ele se controla. Modelo certo, tokens sob controle, e o resto é disciplina.

FIM DO GUIA