OTIMIZAÇÃO DE CUSTOS · USAGE-BASED BILLING · GITHUB COPILOT

Otimização de custos no GitHub Copilot: sete alavancas, quatro camadas.

Sob Usage-Based Billing, o custo depende de apenas dois fatores: qual modelo e quantos tokens. Esta é a referência das sete alavancas de otimização, a base de evidência por trás de cada banda, e a ordem de adoção que carrega a maior parte do resultado.

AUTORA
Paula Silva

FUNÇÃO
Software Global Black Belt

DURAÇÃO
110 a 130 minutos

DATA
2026-06-14

AGENDA

Da equação do custo ao guard-rail.

PARTE

I

A equação do custo.

Sob Usage-Based Billing, o consumo é medido em AI Credits, onde 1 AI Credit equivale a US$ 0,01. O custo de cada interação depende de dois fatores, e só dois: qual modelo e quantos tokens.

A EQUAÇÃO · OS TRÊS TIPOS DE TOKEN

Toda alavanca ataca um destes dois fatores, ou ambos.

TOKENS DE ENTRADA

Prompt, histórico, arquivos e instruções. É a base que o contexto e o caché atacam.

TOKENS DE SAÍDA

O que o modelo gera, tipicamente 4 a 5x o preço da entrada por token. A classe mais cara, o primeiro alvo.

TOKENS EM CACHÉ

Contexto reutilizado, a 10 a 50% de um token novo: Anthropic 0,1x, OpenAI 50% automático sobre prefixo de 1.024 tokens.

COMPLETIONS NÃO CUSTAM

Code completions e Next Edit Suggestions seguem incluídos no plano e não consomem AI Credits. Otimizar completions é esforço perdido.

O CONSUMO REAL

Vive no chat e nas tarefas agênticas. É ali que cada alavanda deste deck atua.

A EQUAÇÃO · POR QUE NÃO SE SOMAM

As bandas se compõem multiplicativamente. Nunca se somam.

Cada alavanca age sobre uma base de tokens distinta: o output control sobre a saída, contexto e caché sobre a entrada, o routing sobre toda a conta. Por isso somar 70% + 80% + 50% e prometer 200% é erro de aritmética.

17 a 25%
Começo austero

46 a 59%
Programa maduro

Bandas direcionais, a confirmar contra a sua linha de base. O número que vale é o seu, medido.

A ORDEM IMPORTA

1. A saída é a classe mais cara: output control é o primeiro movimento.
2. A compressão de contexto é o segundo.
3. O caché é o terceiro.
4. O routing é a decisão arquitetônica que limita o raio de impacto de tudo.

CONTEXTO · O QUE MUDOU EM 2026-06-01

VII

A realidade da fatura.

O Usage-Based Billing entrou em GA. Antes, premium requests com multiplicadores. Agora, AI Credits ligados a tokens. Agentes deixaram de ser feature e passaram a ser compute.

A FATURA · O QUE CONTA E O QUE NÃO CONTA

Autocomplete continua de graça. O resto virou compute medido.

NÃO CONSOME CRÉDITOS

- ✓ Code completions, o autocomplete inline, seguem ilimitados nos planos pagos.
- ✓ Next Edit Suggestions (NES), as sugestões da próxima edição, também ilimitadas.
- ✓ Modelos via BYOK e local: faturados pelo provedor, fora dos AI Credits.

MEDIDO EM AI CREDITS

- Chat, edição multi-arquivo e todas as sessões agênticas.
- GitHub Copilot code review, agora agêntico, que ainda consome minutos de GitHub Actions além dos créditos.
- Análise de repositório, workflows multi-passo e o coding agent na nuvem.

Planos mensais migraram automaticamente em 2026-06-01. Planos anuais seguem no modelo de premium requests até renovar, mas com multiplicadores mais altos para modelos de fronteira. A regra mental: autocomplete ainda é feature; agente é compute.

A FATURA · PLANOS E MIGRAÇÃO

Mensal migrou sozinho. Anual envelhece no modelo antigo, mais caro.

PLANOS E ALLOWANCE INCLUÍDO

PLANO	PREÇO	CRÉDITOS
Pro	US$ 10	1.500 (US$ 15)
Pro+	US$ 39	7.000 (US$ 70)
Max	US$ 100	20.000 (US$ 200)
Business	US$ 19/lic	1.900 (US$ 19)
Enterprise	US$ 39/lic	3.900 (US$ 39)

Max: só por upgrade, para quem já tem plano GitHub Copilot. Business e Enterprise: créditos por usuário, agrupados na entidade. Promoção até 2026-09-01: Business 3.000, Enterprise 7.000.

PLANO MENSAL

Migrou automaticamente em 2026-06-01 para AI Credits. Nada a fazer.

PLANO ANUAL

Segue no premium request até renovar, com os multiplicadores de fronteira que subiram em 2026-06-01:

Claude Opus	27x
GPT-5.4	6x
GPT-5.5	57x

Cada plano inclui base credits, fixos e iguais ao preço, mais um flex allotment variável que o GitHub ajusta conforme a economia dos modelos. 1 AI Credit = US$ 0,01. Quando o pool acaba, o uso continua a preço por crédito, sujeito a budget, ou é bloqueado até o próximo ciclo.

PARTE

II

A base de evidência.

Cada banda tem fonte declarada. Três tipos: preço oficial dos provedores (o mais duro), pesquisa publicada e revisada, e estimativas direcionais de campo, sempre rotuladas como tais.

EVIDÊNCIA · CADA BANDA, SUA FONTE

A regra da honestidade: onde a fonte é preço oficial, o número é um fato.

ALAVANCA	BANDA	FONTE
Output control	40 a 70% da saída	Razão saída/entrada de 4 a 5x nas tabelas públicas dos provedores e do GitHub. A escada de especificação é estimativa de campo.
Model routing	40 a 70% da conta	FrugalGPT (Stanford, TMLR): uma cascata iguala o melhor modelo com 50 a 98% de economia. RouterBench (2024). Banda conservadora.
Modelos locais	um dígito a 15%	Mecânica oficial: BYOK não consome AI Credits (GitHub Changelog, 2026). A participação no custo total é estimativa direcional.
Alcance de contexto	40 a 80% da entrada	LLMLingua (Microsoft, EMNLP 2023): compressão de até 20x com queda de 1,5 pontos. Survey de context engineering (2025).
Caché	30 a 50% da entrada	Preço oficial: leitura de caché a 0,1x na Anthropic e 50% na OpenAI. A banda de loops vem da prática documentada.
Combinado	20-30% / 55-70%	Aritmética de composição sobre as bandas anteriores, consistente com o teto da literatura (FrugalGPT, 50 a 98%).

Fontes primárias: FrugalGPT, LLMLingua, RouterBench, prompt caching de Anthropic e OpenAI (docs oficiais), GitHub AI model comparison.

PARTE

III

A arquitetura.

Sete alavancas em quatro camadas. Leia de cima para baixo, que é também a ordem de adoção. A camada de Saída é onde vive o desperdício mais caro; Governança é o que torna o ganho permanente.

ARQUITETURA · QUATRO CAMADAS

Da mais cara à estrutural.

Saída
A classe de token mais cara, ~5x a entrada

Output control | Instruções concisas | Só diff

Conta
O modelo certo por tarefa limita o raio de impacto

Model routing | Auto | Modelos locais

Entrada
Enviar só o que importa, nunca reprocessar o repetido

Alcance de contexto | Caché | Memória

Governança
O que vira arquivo versionado e guard-rail vira permanente

Primitivos | Budgets | Medição

Dev e admin: Saída e Entrada são do desenvolvedor. Conta é compartilhada, o picker do dev e a política do admin. Governança é do admin.

O PRIMEIRO CORTE · EM UMA HORA

Cinco passos carregam a maior parte do resultado.

01	**Instalar e entrar**	VS Code com as extensões GitHub Copilot e GitHub Copilot Chat, na conta com o plano esperado.
02	**Ligar a medição**	Admin habilita a política GitHub Copilot usage metrics e captura a linha de base em Insights (28 dias, NDJSON e CSV).
03	**Cortar a saída**	Crie o copilot-instructions.md com a diretiva de saída direta e faça commit. A maior alavanca.
04	**Padronizar Auto**	Model picker em Auto (10% de desconto no chat em planos pagos) e desligue o premium fora do chat.
05	**Pôr o guard-rail**	Admin cria o user-level budget universal acima do valor por licença, com alertas em 75, 90 e 100%.

Passos 3 e 4 são as duas maiores alavancas: a saída custa ~5x a entrada por token, e o preço entre classes de modelo abrange duas ordens de magnitude.
Faça ambos antes de qualquer refinamento.

REFERÊNCIA · PRIMITIVOS DO REPOSITÓRIO

Cada primitivo é governança e economia ao mesmo tempo.

.github/ · tree

```
.github/
├── copilot-instructions.md   # sempre ativo
├── instructions/
│   └── tests.instructions.md # applyTo (glob)
├── agents/
│   └── cost-aware-reviewer.agent.md
└── prompts/
    └── release-notes.prompt.md
```

copilot-instructions.md

O primeiro e mais importante. Contexto geral injetado em cada pedido. Conciso, de alto sinal.

***.instructions.md**

Regras com alcance via campo applyTo (glob). Cura o contexto por área do código.

***.agent.md**

Um agente, uma tarefa, com modelo e ferramentas restritos. Governança de alcance e modelo num arquivo.

***.prompt.md**

Encapsula uma instrução longa num comando versionado. Mata o prompt mágico memorizado.

No VS Code, gere os arquivos com /create-instruction, /create-agent, /create-prompt, /create-hook. Um primitivo é código: revise por PR.

GOVERNANÇA · HOOKS COMO GUARD-RAIL

O disjuntor no nível da ação, não só do orçamento.

O ULB para a fatura no nível do orçamento. Os hooks param o desperdício um nível antes: na ação do agente. Um hook PreToolUse pode bloquear uma chamada de ferramenta perigosa ou redundante; um hook Stop encerra um loop antes que ele queime o pool. São gerados no VS Code com /create-hook.

`PreToolUse`	Roda antes de uma ferramenta. Decide pass, block ou non_blocking_error.
`Stop`	Encerra a sessão agêntica. O freio do loop longo de fronteira.

POR QUE IMPORTA NO CUSTO

O loop de agente é o antipadrão que mais queima crédito de uma vez só. Sem freio, ele repete chamadas caras até o orçamento estourar. O hook é o seguro mais barato contra o pior caso.

No OpenTelemetry, o span execute_hook registra cada decisão com pass ou block: você audita o guard-rail em produção.

Gere por /create-hook, não copie um esquema fixo: o formato evolui com a extensão. O hook é um primitivo, entra por PR e é revisado como qualquer outro arquivo de configuração.

REFERÊNCIA · BUDGETS

Quatro níveis de budget. O user-level é o controle mais importante.

USER-LEVEL BUDGET (ULB) OBRIGATÓRIO

Hard stop por ciclo. Configure o universal acima do valor por licença (US$ 19 Business, US$ 39 Enterprise) e overrides para os usuários pesados.

COST-CENTER BUDGET

Limita o gasto metered de uma unidade de negócio depois do pool. Atribui custo por unidade.

ENTERPRISE SPENDING LIMIT

Failsafe global: limita o gasto metered total depois do pool. O último seguro da enterprise.

ORGANIZATION BUDGET

Acompanha o gasto de uma organização ou repositório. Visibilidade, não só corte.

Allowances desde 2026-06-01: Pro 1.500 créditos, Pro+ 7.000, GitHub Copilot Max 20.000; Business 1.900 e Enterprise 3.900 por usuário, com promoção até 2026-09-01. Use a promoção para achar a linha de base, não a leia como o regime permanente. Budgets por usuário em GA desde 2026-06-01.

PARTE

IV

Os procedimentos.

R1 a R7, condensados. Cada receita aponta para a alavanca, o papel que a executa, e o gesto concreto no editor.

PROCEDIMENTOS · R1 E R2

Output control e model routing: as duas maiores alavancas.

R1 · Output control (Dev)

.github/copilot-instructions.md

```
- Responda com código ou diff.
- Sem preâmbulo, sem resumo final.
- A menor mudança correta;
  só as linhas alteradas.
```

No Chat, expanda References: o arquivo de instruções deve aparecer como referência aplicada.

R2 · Model routing (Dev + Admin)

Auto como padrão
Model picker em Auto. Fronteira só por mudança manual, para raciocínio difícil.

Cortar o premium silencioso
Visual Studio: desligue Enhance non-chat requests with premium models. VS Code: utility model numa classe leve.

Fixar a política
O admin define quais modelos os membros podem usar. Mudar o modelo do chat não muda o das inline suggestions.

PROCEDIMENTOS · R3 A R5

Modelos locais, alcance de contexto, e o caché que nunca se reprocessa.

R3 · MODELOS LOCAIS

BYOK habilitada por padrão. Registre Ollama ou Foundry Local; commits, boilerplate e tests vão para o local, a custo zero de créditos.

O agent mode exige modelos com tool calling.

R4 · ALCANCE DE CONTEXTO

Prefira #file, #sym e #changes a despejar todo o #codebase. Content exclusion (admin) para paths sensíveis.

Não suportado em Edit e Agent mode.

R5 · CACHÉ E MEMÓRIA

Estabilize o prefixo: instruções no topo, sem edições no meio da sessão. Agrupe o trabalho relacionado.

O thrash quebra o caché.

R6 Primitivos: ordem de adoção é instruções gerais, instruções com alcance, agentes custom, prompt files, hooks. R7 Budgets: primeiro o ULB universal, depois overrides e cost centers, e medir a cada ciclo.

ENTRADA · GITHUB COPILOT MEMORY

A memória cura o contexto entre sessões, para você não reexplicar.

O GitHub Copilot Memory, em public preview, captura fatos e preferências do seu trabalho e os reaproveita nas próximas sessões. É uma alavanca de entrada: o que a memória guarda, você não precisa mandar de novo no prompt a cada conversa. Menos entrada repetida, menos token.

Habilite e revise
Veja os fatos e preferências capturados. A memória só ajuda se refletir o projeto de verdade.

O que envelhece, sai
Memória desatualizada vira ruído caro: ela entra no contexto a cada pedido. Pode, remova.

A IMAGEM

O caderno do projeto.

Um bom colega anota o que importa do seu projeto uma vez, e não pergunta de novo na semana seguinte. A memória é esse caderno: mantida, ela economiza; abandonada, ela atrapalha.

A memória complementa os primitivos versionados: o copilot-instructions.md guarda a regra estável e revisada por PR; a memória guarda o que é aprendido na sessão. Os dois reduzem a entrada, por caminhos diferentes.

ENTRADA · CACHÉ NA PRÁTICA

O mesmo trabalho, com e sem disciplina de caché.

ANTES · O CACHÉ NUNCA ACERTA

- Instruções editadas no meio da sessão: o prefixo muda, o caché invalida.
- Trabalho fragmentado em sessões curtas: cada uma recomeça do zero.
- Resultado: você paga a entrada inteira, a preço cheio, a cada pedido.

100% da entrada

preço cheio, sempre

DEPOIS · PREFIXO ESTÁVEL

- ✓ Instruções e AGENTS.md no topo, sem edições no meio: o prefixo fica estável.
- ✓ Trabalho relacionado agrupado na mesma sessão: o contexto se reaproveita.
- ✓ A leitura de caché custa uma fração: Anthropic 0,1x, OpenAI 50% sobre o prefixo.

10 a 50%

do token novo, na parte cacheada

A diferença não é o modelo, é a higiene de sessão. O Cache Explorer, no Agent Debug Logs, mostra a taxa de acerto e quantos tokens de entrada foram reaproveitados: é ali que você confirma se a disciplina está funcionando.

MODELOS LOCAIS · FOUNDRY LOCAL E AI TOOLKIT

A borda de custo zero: rode a rotina na sua própria máquina.

O BYOK não consome AI Credits: é faturado pelo provedor, ou gratuito quando o modelo roda local. O AI Toolkit conecta o GitHub Copilot Chat a modelos do Azure AI Foundry ou ao Foundry Local na sua máquina. Quatro passos no editor.

01 Instale as extensões GitHub Copilot Chat e AI Toolkit no VS Code.

02 No model picker do Chat, escolha Adicionar modelo e selecione Foundry Local via AI Toolkit.

03 O AI Toolkit baixa o modelo se ainda não estiver na máquina. Sem API key: a credencial é gerada localmente.

04 Troque entre local e nuvem pelo picker, por tarefa. A rotina vai para o local, a fronteira fica para a nuvem.

terminal · Foundry Local

```
# instalar
winget install Microsoft.FoundryLocal
# rodar um modelo de código
foundry model run phi-4
# Ollama, alternativa
ollama pull llama3.1
```

DOIS REQUISITOS

O agent mode exige modelos com tool calling. E hardware importa: o GPU define a performance do modelo local. BYOK não vale para completions.

BYOK está ativo por padrão em Business e Enterprise, com modelos de provedores como Anthropic, Gemini, OpenAI, OpenRouter e Azure, além de Ollama e Foundry Local. Uma vez configurado, o modelo aparece em todo o Chat, inclusive no Plan agent e nos custom agents.

ROTEAMENTO · QUE TAREFA VAI PARA ONDE

A regra prática: o modelo certo por tipo de tarefa.

TAREFA	CLASSE DE MODELO	ONDE
Commits, boilerplate, scaffolding de testes	Leve / local	Foundry Local, custo zero
Explicar código, dúvidas factuais, autocomplete	Incluída / leve	Completions são grátis
Geração de feature padrão, refactoring local	Intermediária	Auto no picker
Planejamento de arquitetura, decisão de design	Fronteira, reasoning	Plan agent, depois troca
Refactoring multi-arquivo, debug multi-passo	Fronteira, reasoning	Manual, escopo restrito
Nunca: fronteira para tarefa trivial	Antipadrão	Caminhão para um envelope

O picker mostra o custo no hover: custo por tipo de token e um rótulo de tier (Baixo, Médio, Alto). Custom agents podem fixar um modelo barato por subtarefa: ao invocar como subagente, usam o próprio modelo, não o da sessão.

ROTEAMENTO · OS CONTROLES MUDAM POR EDITOR

Conheça os controles do seu editor. Eles não são iguais.

VS CODE

O conjunto mais completo: Auto, utility model, Plan agent, /compact, /fork, Configure Tools, OpenTelemetry e o Agent Debug Logs.

Todas as alavancas deste deck.

VISUAL STUDIO

O premium silencioso vive aqui: Tools, Options, GitHub, GitHub Copilot, Editor. Desligue Enhance non-chat requests with premium models.

O ajuste que mais surpreende na fatura.

JETBRAINS E OUTROS

A telemetria pode ser menos consistente fora do VS Code. Mantenha o IDE na última versão para o dashboard refletir o uso real.

Atualize antes de confiar nos números.

A política de modelo do admin vale em todos os editores, mas os gestos de fluxo (novo chat, fork, compact) e a observabilidade hoje são mais ricos no VS Code. Padronize o editor onde o programa de FinOps precisa de mais controle.

PARTE

V

Troubleshooting.

Antipadrões: sintoma, causa, solução. Os quatro problemas que aparecem antes de qualquer outro, e como diagnosticá-los.

TROUBLESHOOTING · SINTOMA E SOLUÇÃO

Quatro antipadrões, quatro diagnósticos.

A conta subiu sem mudança de uso	Causa: pedidos fora do chat melhorados com modelos premium, ou o utility model apontado para um caro. Solução: desligue a melhoria fora do chat e aponte o utility model para a classe leve.
Um usuário queimou o pool	Causa: um loop de agente ou uma sessão agêntica longa de fronteira. Solução: o ULB universal como seguro, um override individual, limites de loop e comportas de aprovação.
O caché nunca acerta	Causa: prefixo instável (instruções editadas no meio da sessão) ou trabalho fragmentado em sessões curtas. Solução: contexto estável no topo, agrupar o trabalho relacionado.
O combinado não bate com a soma	Sintoma: alguém somou 70% + 80% + 50% e prometeu 200%. Solução: as bandas se compõem multiplicativamente. Use 20 a 30% austero, 55 a 70% maduro, e confirme contra a linha de base.

Perigo: sem ULB, um só loop acidental pode consumir o pool da unidade. O budget por usuário é um seguro barato; configure antes do uso crescer.

MEDIÇÃO · A CAMADA QUE FALTAVA

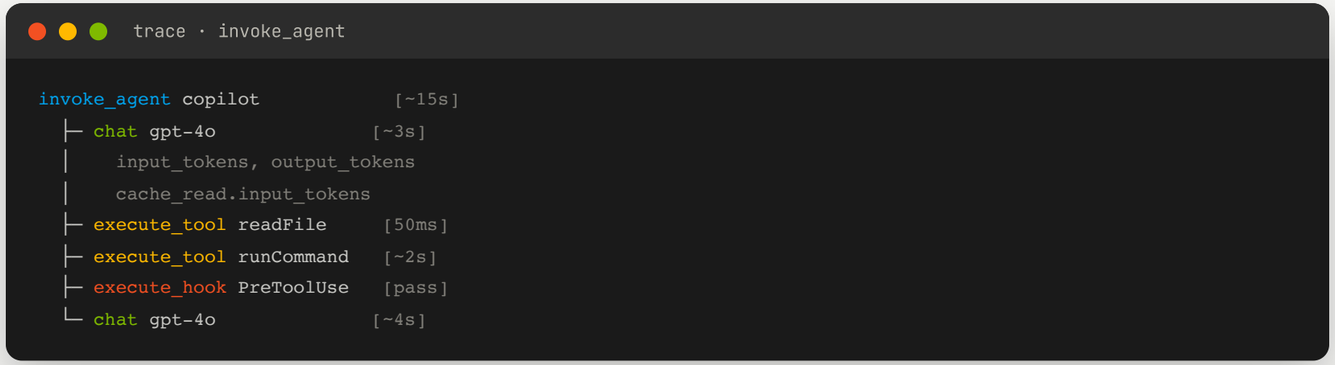
VIII

Observabilidade com OpenTelemetry.

Otimizar sem medir é adivinhar. Desde o VS Code 1.119, o GitHub Copilot Chat e o GitHub Copilot CLI exportam traces, métricas e eventos via OTel, com tokens, custo e cache por sessão. Desligado por padrão, zero overhead até você ligar.

OPENTELEMETRY · O QUE VOCÊ PASSA A VER

Cada interação vira uma árvore de spans, com tokens e cache medidos.



TOKENS POR CHAMADA

gen_ai.usage.input_tokens, output_tokens e cache_read.input_tokens. Você vê exatamente onde o crédito vai.

ATRIBUIÇÃO POR TIME

OTEL_RESOURCE_ATTRIBUTES marca team.id e department. Filtre o custo por equipe ou squad.

CONVENÇÕES GENAI

Todos os sinais seguem as OTel GenAI Semantic Conventions: funciona em qualquer backend OTLP.

Métricas incluem gen_ai.client.token.usage e custo por modelo. Eventos cobrem aceitação de edição, sobrevivência do código e feedback. Por padrão, nenhum conteúdo de prompt é capturado: só metadados como modelo, tokens e duração.

OPENTELEMETRY · LIGAR EM DOIS LUGARES

No VS Code e no GitHub Copilot CLI. Aponte para um backend e veja os traces.

VS Code · settings.json

```
{
  "github.copilot.chat.otel.enabled": true,
  "github.copilot.chat.otel.otlpEndpoint":
    "http://localhost:4318"
}
```

Liga os traces do agente de primeiro plano, do GitHub Copilot CLI em background e do Claude agent. O mesmo setting cobre os três.

GitHub Copilot CLI · terminal

```
# ligar e exportar para arquivo
export COPILOT_OTEL_ENABLED=true
export COPILOT_OTEL_EXPORTER_TYPE=file
export COPILOT_OTEL_FILE_EXPORTER_PATH=\
  ~/.copilot/otel/run.jsonl
```

O JSONL local alimenta ferramentas de custo da comunidade que leem ~/.copilot/otel e somam tokens por sessão.

Aspire Dashboard, local, zero cloud | Jaeger | Grafana + App Insights | Langfuse

Off por padrão, sem phone-home: os dados vão só para onde você aponta. Sem captura de conteúdo por padrão. O Grafana Managed tem um dashboard pronto para tokens de entrada e saída, sessões, tool calls e tempo de resposta por modelo.

ALAVANCAS NATIVAS · NO FLUXO DO VS CODE

Seis gestos no editor que cortam tokens sem tocar em configuração.

PLANEJAR, DEPOIS IMPLEMENTAR

Use o Plan agent (reasoning) para o plano, aprove, e passe para um agente rápido executar. Menos vai e volta, menos retrabalho.

NOVO CHAT POR TAREFA

Ao trocar de assunto, abra uma sessão nova. O histórico irrelevante é reprocessado a cada turno e queima tokens à toa.

/COMPACT

Em sessão longa, resuma as partes antigas e recupere espaço de contexto. Aceita foco: /compact foque nas decisões de API.

/FORK

Para explorar uma alternativa, bifurque a conversa em vez de recomeçar. Herda o contexto, sem reestabelecer tudo do zero.

DESLIGAR FERRAMENTAS

Cada tool call consome contexto. Configure Tools desliga MCP servers e ferramentas que a tarefa atual não precisa.

THINKING EFFORT NO PADRÃO

Mais reasoning gera mais thinking tokens. O padrão adaptativo basta na maioria; suba só para problema complexo de verdade.

Inspecione com o Agent Debug Logs: o Summary mostra tokens agregados da sessão, e o Cache Explorer mostra a taxa de acerto do prompt cache e quantos tokens de entrada foram reaproveitados. Excluir build outputs via .gitignore tira lixo do índice.

MEDIÇÃO · LER O DASHBOARD

Atribua por usuário, modelo e feature. Ajuste a banda com o dado real.

01

Capture a linha de base

Na aba Insights, janela de 28 dias. Exporte em NDJSON e CSV. Sem linha de base, toda banda é chute.

02

Atribua o consumo

Quebre por usuário, modelo e feature. O pico costuma estar num punhado de usuários ou numa feature agêntica.

03

Releia a cada ciclo

Ajuste a banda contra o dado real, não contra a expectativa. A meta é a sua curva, medida, não a do slide.

> Dois níveis de medição se completam: o dashboard de Insights dá a visão de gestão, agregada e por ciclo; o OpenTelemetry dá a visão de engenharia, span a span, com tokens e cache por sessão. Um responde quanto e quem; o outro responde onde e por quê.

A experiência de preview da fatura, no Billing Overview, mostra como o custo se desloca no novo modelo antes de virar cobrança. Leia antes de o uso crescer.

REFERÊNCIA CONSOLIDADA · TODAS AS ALAVANCAS

O mapa completo de impacto, banda e tipo de fonte.

ALAVANCA	BANDA	SOBRE O QUÊ AGE	FONTE
Output control	40 a 70% saída	A classe de token mais cara	Preço oficial
Model routing	40 a 70% conta	Toda a conta, limita o raio	Pesquisa
Alcance de contexto	40 a 80% entrada	A entrada enviada por pedido	Pesquisa
Caché	30 a 50% entrada	A entrada repetida em loops	Preço oficial
Modelos locais	um dígito a 15%	A rotina, a custo zero de créditos	Direcional
Primitivos + governança	composto	Torna o ganho permanente	Direcional
Combinado	**20-30% / 55-70%**	Composição multiplicativa, austero a maduro	Direcional

Mais a medição com OTel e as alavancas nativas (plan, novo chat, /compact, /fork, desligar ferramentas, thinking effort), que reduzem tokens por sessão sem entrar na tabela de bandas. Lembre: as bandas se compõem, nunca somam.

A DECISÃO · CUSTO DO PROGRAMA E RETORNO

O programa é barato. O ganho se compõe e é permanente.

O QUE O PROGRAMA CUSTA

Algumas horas para o primeiro corte: instruções, Auto, ULB.

Disciplina de sessão, que vira hábito em uma ou duas semanas.

Os primitivos, escritos uma vez e revisados por PR como código.

Não é projeto, é configuração mais hábito. O custo é tempo, não licença nova.

O QUE O PROGRAMA DEVOLVE

55 a 70%

de redução num programa maduro, mês após mês

O ganho não é um corte único, é uma inclinação. Cada alavanca dobra a curva, e a governança trava o resultado para que ele não evapore na primeira semana corrida.

A assimetria é a tese: o investimento é pequeno e único; a economia é composta e recorrente. O payback de cortar a saída e pôr o ULB chega no primeiro ciclo de fatura.

A DECISÃO · O CASO, EM NÚMEROS

Como as bandas se compõem, passo a passo, numa linha de base ilustrativa.

Linha de base mensal	ponto de partida	1000
Output control	× (1 - 0,40)	600
Model routing	× (1 - 0,30)	420
Alcance de contexto	× (1 - 0,15)	357
Caché em loops	× (1 - 0,10)	321

Resultado composto

68%

de redução, dentro da banda madura de 55 a 70%. Não é a soma de 40 mais 30 mais 15 mais 10, que daria 95.

Números ilustrativos, para mostrar a mecânica da composição. As porcentagens de cada passo são exemplos dentro das bandas; o seu resultado é o que você medir contra a sua linha de base. A lição é a forma da conta, não os dígitos.

PARTE

VI

O programa em sete imagens.

Para qualquer audiência. Cada alavanca tem uma metáfora que cabe numa frase e dispensa o jargão técnico.

METÁFORAS · AS SETE IMAGENS

Da carta ao telegrama, do caminhão ao interruptor diferencial.

R1 O telegrama e a carta
A instrução converte a carta em telegrama: só o que muda. 40 a 70% da saída.

R2 O caminhão e o envelope
O routing escolhe o veículo pelo tamanho da carga. 40 a 70% da conta.

R3 Cozinhar em casa
A rotina sai da conta num modelo local. Um dígito a ~15% do total.

R4 A pasta certa para a reunião
Levar o arquivo morto inteiro confunde e custa. 40 a 80% da entrada.

R5 A credencial de acesso
Depois do registro, só apoie a credencial. 30 a 50% da entrada em loops.

R6 O corrimão da escada
A disciplina vira infraestrutura versionada. Composto e permanente.

R7 O interruptor diferencial
Não baixa a conta de luz do mês, evita o incêndio do pico. A economia vem das outras alavancas; o cap sustenta a variância. ULB em GA desde 2026-06-01.

RESUMO

Seis ideias para levar.

DOIS FATORES

O custo depende de qual modelo e quantos tokens. Toda alavanca ataca um deles, ou ambos.

A ORDEM IMPORTA

Saída primeiro (a classe mais cara), depois contexto, depois caché. Routing limita o raio de tudo.

BANDAS SE COMPÕEM

Nunca somam. 20 a 30% austero, 55 a 70% maduro. Cada cifra tem fonte declarada.

PRIMITIVOS SÃO CÓDIGO

Instruções, agentes, prompts e hooks versionados. Governança e economia no mesmo arquivo.

O ULB É O SEGURO

Hard stop por usuário. Configure o universal antes do uso crescer; um loop acidental queima o pool.

MEÇA PRIMEIRO

A banda que vale é a sua, contra a linha de base. Otimizar sem medir é adivinhar.

ENCERRAMENTO

Meça primeiro, aplique em ordem, deixe a banda se ajustar com dados reais.

CONTATO

Paula Silva
Software Global Black Belt
paulasilva@microsoft.com

PRÓXIMO PASSO

Capture a linha de base hoje
Insights, 28 dias, e faça commit do copilot-instructions.md

A REGRA DE ROUTING · ACORDO DO TIME

```
trivial  -> classe leve / incluída
padrão -> classe intermediária
complexo -> fronteira, reasoning
nunca    -> fronteira p/ tarefa trivial
```

Microsoft + GitHub · Deck v2.1.0 · Atualizado em 2026-06-14