OPTIMIZACIÓN DE COSTOS · USAGE-BASED BILLING · GITHUB COPILOT

Optimización de costos en GitHub Copilot: siete palancas, cuatro capas.

Bajo Usage-Based Billing, el costo depende de solo dos factores: qué modelo y cuántos tokens. Esta es la referencia de las siete palancas de optimización, la base de evidencia detrás de cada banda, y el orden de adopción que carga la mayor parte del resultado.

AUTORA
Paula Silva

FUNCIÓN
Software Global Black Belt

DURACIÓN
110 a 130 minutos

FECHA
2026-06-14

AGENDA

De la ecuación del costo al guard-rail.

PARTE

I

La ecuación del costo.

Bajo Usage-Based Billing, el consumo se mide en AI Credits, donde 1 AI Credit equivale a US$ 0,01. El costo de cada interacción depende de dos factores, y solo dos: qué modelo y cuántos tokens.

LA ECUACIÓN · LOS TRES TIPOS DE TOKEN

Toda palanca ataca uno de estos dos factores, o ambos.

TOKENS DE ENTRADA

Prompt, historial, archivos e instrucciones. La base que el contexto y el caché atacan.

TOKENS DE SALIDA

Lo que el modelo genera, típicamente 4 a 5x el precio de la entrada por token. La clase más cara, el primer objetivo.

TOKENS EN CACHÉ

Contexto reutilizado, a 10 a 50% de un token nuevo: Anthropic 0,1x, OpenAI 50% automático sobre un prefijo de 1.024 tokens.

COMPLETIONS NO CUESTAN

Code completions y Next Edit Suggestions siguen incluidos en el plan y no consumen AI Credits. Optimizar completions es esfuerzo perdido.

EL CONSUMO REAL

Vive en el chat y en las tareas agénticas. Es allí donde cada palanca de este deck actúa.

LA ECUACIÓN · POR QUÉ NO SE SUMAN

Las bandas se componen multiplicativamente. Nunca se suman.

Cada palanca actúa sobre una base de tokens distinta: el output control sobre la salida, contexto y caché sobre la entrada, el routing sobre toda la cuenta. Por eso sumar 70% + 80% + 50% y prometer 200% es un error de aritmética.

17 a 25%
Comienzo austero

46 a 59%
Programa maduro

EL ORDEN IMPORTA

1. La salida es la clase más cara: el output control es el primer movimiento.
2. La compresión de contexto es el segundo.
3. El caché es el tercero.
4. El routing es la decisión arquitectónica que limita el radio de impacto de todo.

Bandas direccionales, a confirmar contra su línea de base. El número que vale es el suyo, medido.

CONTEXTO · LO QUE CAMBIÓ EL 2026-06-01

VII

La realidad de la factura.

El Usage-Based Billing llegó a GA. Antes, premium requests con multiplicadores. Ahora, AI Credits ligados a tokens. Los agentes dejaron de ser una feature y se volvieron compute.

LA FACTURA · QUÉ CUENTA Y QUÉ NO

El autocompletado sigue gratis. El resto se volvió compute medido.

NO CONSUME CRÉDITOS

- ✓ Code completions, el autocompletado inline, siguen ilimitados en los planes pagos.
- ✓ Next Edit Suggestions (NES), las sugerencias de la próxima edición, también ilimitadas.
- ✓ Modelos vía BYOK y local: facturados por el proveedor, fuera de los AI Credits.

MEDIDO EN AI CREDITS

- Chat, edición multiarchivo y todas las sesiones agénticas.
- GitHub Copilot code review, ahora agéntico, que además consume minutos de GitHub Actions sobre los créditos.
- Análisis de repositorio, workflows multipaso y el coding agent en la nube.

Los planes mensuales migraron automáticamente el 2026-06-01. Los planes anuales siguen en el modelo de premium requests hasta renovar, pero con multiplicadores más altos para modelos de frontera. La regla mental: el autocompletado sigue siendo una feature; un agente es compute.

LA FACTURA · PLANES Y MIGRACIÓN

El mensual migró solo. El anual envejece en el modelo viejo, más caro.

PLANES Y ALLOWANCE INCLUIDO

PLAN	PRECIO	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: solo por upgrade, para quien ya tiene plan GitHub Copilot. Business y Enterprise: créditos por usuario, agrupados en la entidad. Promoción hasta 2026-09-01: Business 3.000, Enterprise 7.000.

PLAN MENSUAL

Migró automáticamente el 2026-06-01 a AI Credits. Sin acción necesaria.

PLAN ANUAL

Sigue en premium requests hasta renovar, con los multiplicadores de frontera que subieron el 2026-06-01:

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

Cada plan incluye base credits, fijos e iguales al precio, más un flex allotment variable que GitHub ajusta según la economía de los modelos. 1 AI Credit = US$ 0,01. Cuando el pool se acaba, el uso continúa a precio por crédito, sujeto a budget, o se bloquea hasta el próximo ciclo.

PARTE

II

La base de evidencia.

Cada banda tiene fuente declarada. Tres tipos: precio oficial de los proveedores (el más duro), investigación publicada y revisada, y estimaciones direccionales de campo, siempre rotuladas como tales.

EVIDENCIA · CADA BANDA, SU FUENTE

La regla de honestidad: donde la fuente es precio oficial, el número es un hecho.

PALANCA	BANDA	FUENTE
Output control	40 a 70% de la salida	Razón salida/entrada de 4 a 5x en las tablas públicas de los proveedores y de GitHub. La escalera de especificación es estimación de campo.
Model routing	40 a 70% de la cuenta	FrugalGPT (Stanford, TMLR): una cascada iguala al mejor modelo con 50 a 98% de ahorro. RouterBench (2024). Banda conservadora.
Modelos locales	un dígito a 15%	Mecánica oficial: BYOK no consume AI Credits (GitHub Changelog, 2026). La participación en el costo total es estimación direccional.
Alcance de contexto	40 a 80% de la entrada	LLMLingua (Microsoft, EMNLP 2023): compresión de hasta 20x con caída de ~1,5 puntos. Survey de context engineering (2025).
Caché	30 a 50% de la entrada	Precio oficial: lectura de caché a 0,1x en Anthropic y 50% en OpenAI. La banda de loops viene de la práctica documentada.
Combinado	**20-30% / 55-70%**	Aritmética de composición sobre las bandas anteriores, consistente con el techo de la literatura (FrugalGPT, 50 a 98%).

Fuentes primarias: FrugalGPT, LLMLingua, RouterBench, prompt caching de Anthropic y OpenAI (docs oficiales), GitHub AI model comparison.

PARTE

III

La arquitectura.

Siete palancas en cuatro capas. Léase de arriba hacia abajo, que es también el orden de adopción. La capa de Salida es donde vive el desperdicio más caro; Gobernanza es lo que vuelve la ganancia permanente.

ARQUITECTURA · CUATRO CAPAS

De la más cara a la estructural.

Salida
La clase de token más cara, ~5x la entrada

Output control · Instrucciones concisas · Solo diff

Cuenta
El modelo correcto por tarea limita el radio de impacto

Model routing · Auto · Modelos locales

Entrada
Enviar solo lo que importa, nunca reprocesar lo repetido

Alcance de contexto · Caché · Memoria

Gobernanza
Lo que se vuelve archivo versionado y guard-rail se vuelve permanente

Primitivos · Budgets · Medición

Dev y admin: Salida y Entrada son del desarrollador. Cuenta es compartida, el picker del dev y la política del admin. Gobernanza es del admin.

EL PRIMER CORTE · EN UNA HORA

Cinco pasos cargan la mayor parte del resultado.

01	**Instalar e iniciar sesión**	VS Code con las extensiones GitHub Copilot y GitHub Copilot Chat, en la cuenta con el plan esperado.
02	**Encender la medición**	Admin habilita la política GitHub Copilot usage metrics y captura la línea de base en Insights (28 días, NDJSON y CSV).
03	**Cortar la salida**	Cree copilot-instructions.md con la directiva de salida directa y haga commit. La mayor palanca.
04	**Estandarizar Auto**	Model picker en Auto (10% de descuento en chat en planes pagos) y apague el premium fuera del chat.
05	**Poner el guard-rail**	Admin crea el user-level budget universal por encima del valor por licencia, con alertas en 75, 90 y 100%.

Los pasos 3 y 4 son las dos palancas más grandes: la salida cuesta ~5x la entrada por token, y el precio entre clases de modelo abarca dos órdenes de magnitud. Haga ambos antes de cualquier refinamiento.

REFERENCIA · PRIMITIVOS DEL REPOSITORIO

Cada primitivo es gobernanza y ahorro al mismo tiempo.

.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

El primero y más importante. Contexto general inyectado en cada pedido. Conciso, de alta señal.

***.instructions.md**

Reglas con alcance vía el campo applyTo (glob). Cura el contexto por área del código.

***.agent.md**

Un agente, una tarea, con modelo y herramientas restringidos. Gobernanza de alcance y modelo en un archivo.

***.prompt.md**

Encapsula una instrucción larga en un comando versionado. Mata el prompt mágico memorizado.

En VS Code, genere los archivos con /create-instruction, /create-agent, /create-prompt, /create-hook. Un primitivo es código: revíselo por PR.

GOBERNANZA · HOOKS COMO GUARD-RAIL

El interruptor diferencial en el nivel de la acción, no solo del presupuesto.

El ULB detiene la factura en el nivel del presupuesto. Los hooks detienen el desperdicio un nivel antes: en la acción del agente. Un hook PreToolUse puede bloquear una llamada de herramienta peligrosa o redundante; un hook Stop encierra un loop antes de que queme el pool. Se generan en VS Code con /create-hook.

`PreToolUse`	Corre antes de una herramienta. Decide pass, block o non_blocking_error.
`Stop`	Encierra la sesión agéntica. El freno del loop largo de frontera.

POR QUÉ IMPORTA EN EL COSTO

El loop de agente es el antipatrón que más quema crédito de una vez. Sin freno, repite llamadas caras hasta que el presupuesto estalla. El hook es el seguro más barato contra el peor caso.

En OpenTelemetry, el span execute_hook registra cada decisión como pass o block: auditas el guard-rail en producción.

Genera con /create-hook, no copies un esquema fijo: el formato evoluciona con la extensión. El hook es un primitivo, entra por PR y se revisa como cualquier otro archivo de configuración.

REFERENCIA · BUDGETS

Cuatro niveles de budget. El user-level es el control más importante.

USER-LEVEL BUDGET (ULB) OBLIGATORIO

Hard stop por ciclo. Configure el universal por encima del valor por licencia (US$ 19 Business, US$ 39 Enterprise) y overrides para los usuarios pesados.

COST-CENTER BUDGET

Limita el gasto metered de una unidad de negocio después del pool. Asigna costo por unidad.

ENTERPRISE SPENDING LIMIT

Failsafe global: limita el gasto metered total después del pool. El último seguro de la enterprise.

ORGANIZATION BUDGET

Acompaña el gasto de una organización o repositorio. Visibilidad, no solo corte.

Allowances desde 2026-06-01: Pro 1.500 créditos, Pro+ 7.000, GitHub Copilot Max 20.000; Business 1.900 y Enterprise 3.900 por usuario, con promoción hasta 2026-09-01. Use la promoción para hallar la línea de base, no la lea como el régimen permanente. Budgets por usuario en GA desde 2026-06-01.

PARTE

IV

Los procedimientos.

R1 a R7, condensados. Cada receta apunta a la palanca, el rol que la ejecuta, y el gesto concreto en el editor.

PROCEDIMIENTOS · R1 Y R2

Output control y model routing: las dos palancas más grandes.

R1 · Output control (Dev)

.github/copilot-instructions.md

```
- Responda con código o diff.
- Sin preámbulo, sin resumen final.
- El menor cambio correcto; solo las líneas alteradas.
```

En el Chat, expanda References: el archivo de instrucciones debe aparecer como referencia aplicada.

R2 · Model routing (Dev + Admin)

Auto como estándar
Model picker en Auto. Frontera solo por cambio manual, para razonamiento difícil.

Cortar el premium silencioso
Visual Studio: apague Enhance non-chat requests with premium models. VS Code: utility model en una clase liviana.

Fijar la política
El admin define qué modelos pueden usar los miembros. Cambiar el modelo del chat no cambia el de las inline suggestions.

PROCEDIMIENTOS · R3 A R5

Modelos locales, alcance de contexto, y el caché que nunca se reprocesa.

R3 · MODELOS LOCALES

BYOK habilitada por defecto. Registre Ollama o Foundry Local; commits, boilerplate y tests van a lo local, a costo cero de créditos.

El agent mode exige modelos con tool calling.

R4 · ALCANCE DE CONTEXTO

Prefiera #file, #sym y #changes a volcar todo el #codebase. Content exclusion (admin) para paths sensibles.

No soportado en Edit y Agent mode.

R5 · CACHÉ Y MEMORIA

Estabilice el prefijo: instrucciones arriba, sin ediciones a mitad de sesión. Agrupe el trabajo relacionado.

El thrash rompe el caché.

R6 Primitivos: el orden de adopción es instrucciones generales, instrucciones con alcance, agentes custom, prompt files, hooks. R7 Budgets: primero el ULB universal, después overrides y cost centers, y medir cada ciclo.

ENTRADA · GITHUB COPILOT MEMORY

La memoria cura el contexto entre sesiones, para que no reexpliques.

GitHub Copilot Memory, en public preview, captura hechos y preferencias de tu trabajo y los reutiliza en las próximas sesiones. Es una palanca de entrada: lo que la memoria guarda, no tienes que mandarlo de nuevo en el prompt cada conversación. Menos entrada repetida, menos tokens.

Habilita y revisa
Mira los hechos y preferencias capturados. La memoria solo ayuda si refleja el proyecto de verdad.

Lo que envejece, sale
La memoria desactualizada se vuelve ruido caro: entra en el contexto en cada pedido. Puedes, quítala.

LA IMAGEN

El cuaderno del proyecto.

Un buen colega anota lo que importa de tu proyecto una vez, y no pregunta de nuevo a la semana siguiente. La memoria es ese cuaderno: mantenida, ahorra; abandonada, estorba.

La memoria complementa los primitivos versionados: el copilot-instructions.md guarda la regla estable y revisada por PR; la memoria guarda lo que se aprende en la sesión. Ambos reducen la entrada, por caminos distintos.

ENTRADA · CACHÉ EN LA PRÁCTICA

El mismo trabajo, con y sin disciplina de caché.

ANTES · EL CACHÉ NUNCA ACIERTA

- Instrucciones editadas a mitad de sesión: el prefijo cambia, el caché se invalida.
- Trabajo fragmentado en sesiones cortas: cada una recomienza de cero.
- Resultado: pagas la entrada entera, a precio lleno, en cada pedido.

100% de la entrada

precio lleno, siempre

DESPUÉS · PREFIJO ESTABLE

- ✓ Instrucciones y AGENTS.md arriba, sin ediciones a mitad: el prefijo queda estable.
- ✓ Trabajo relacionado agrupado en la misma sesión: el contexto se reutiliza.
- ✓ La lectura de caché cuesta una fracción: Anthropic 0,1x, OpenAI 50% sobre el prefijo.

10 a 50%

del token nuevo, en la parte cacheada

La diferencia no es el modelo, es la higiene de sesión. El Cache Explorer, en los Agent Debug Logs, muestra la tasa de acierto y cuántos tokens de entrada se reutilizaron: ahí confirmas si la disciplina funciona.

MODELOS LOCALES · FOUNDRY LOCAL Y AI TOOLKIT

El borde de costo cero: ejecute la rutina en su propia máquina.

BYOK no consume AI Credits: lo factura el proveedor, o es gratis cuando el modelo corre local. El AI Toolkit conecta GitHub Copilot Chat con modelos de Azure AI Foundry o con Foundry Local en su máquina. Cuatro pasos en el editor.

01 Instale las extensiones GitHub Copilot Chat y AI Toolkit en VS Code.

02 En el model picker del Chat, elija Agregar modelo y seleccione Foundry Local via AI Toolkit.

03 El AI Toolkit descarga el modelo si aún no está en la máquina. Sin API key: la credencial se genera localmente.

04 Cambie entre local y nube en el picker, por tarea. La rutina va a lo local, la frontera queda en la nube.

terminal · Foundry Local

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

DOS REQUISITOS

El agent mode exige modelos con tool calling. Y el hardware importa: el GPU define la performance del modelo local. BYOK no aplica a completions.

BYOK está activo por defecto en Business y Enterprise, con modelos de proveedores como Anthropic, Gemini, OpenAI, OpenRouter y Azure, además de Ollama y Foundry Local. Una vez configurado, el modelo aparece en todo el Chat, incluido el Plan agent y los custom agents.

ENRUTAMIENTO · QUÉ TAREA VA A DÓNDE

La regla práctica: el modelo correcto por tipo de tarea.

TAREA	CLASE DE MODELO	DÓNDE
Commits, boilerplate, scaffolding de pruebas	Liviana / local	Foundry Local, costo cero
Explicar código, preguntas factuales, autocompletado	Incluida / liviana	Completions son gratis
Generación de feature estándar, refactoring local	Intermedia	Auto en el picker
Planificación de arquitectura, decisión de diseño	Frontera, reasoning	Plan agent, luego cambia
Refactoring multiarchivo, debug multipaso	Frontera, reasoning	Manual, alcance restringido
Nunca: frontera para tarea trivial	Antipatrón	Un camión para un sobre

El picker muestra el costo en el hover: costo por tipo de token y una etiqueta de tier (Bajo, Medio, Alto). Los custom agents pueden fijar un modelo barato por subtarea: al invocarse como subagente, usan su propio modelo, no el de la sesión.

ENRUTAMIENTO · LOS CONTROLES CAMBIAN POR EDITOR

Conoce los controles de tu editor. No son iguales.

VS CODE

El conjunto más completo: Auto, utility model, Plan agent, /compact, /fork, Configure Tools, OpenTelemetry y los Agent Debug Logs.

Todas las palancas de este deck.

VISUAL STUDIO

El premium silencioso vive aquí: Tools, Options, GitHub, GitHub Copilot, Editor. Apaga Enhance non-chat requests with premium models.

El ajuste que más sorprende en la factura.

JETBRAINS Y OTROS

La telemetría puede ser menos consistente fuera de VS Code. Mantén el IDE en la última versión para que el dashboard refleje el uso real.

Actualiza antes de confiar en los números.

La política de modelo del admin vale en todos los editores, pero los gestos de flujo (nuevo chat, fork, compact) y la observabilidad hoy son más ricos en VS Code. Estandariza el editor donde el programa de FinOps necesita más control.

PARTE

V

Troubleshooting.

Antipatrones: síntoma, causa, solución. Los cuatro problemas que aparecen antes que cualquier otro, y cómo diagnosticarlos.

TROUBLESHOOTING · SÍNTOMA Y SOLUCIÓN

Cuatro antipatrones, cuatro diagnósticos.

La cuenta subió sin cambio de uso	Causa: pedidos fuera del chat mejorados con modelos premium, o el utility model apuntado a uno caro. Solución: apague la mejora fuera del chat y apunte el utility model a la clase liviana.
Un usuario quemó el pool	Causa: un loop de agente o una sesión agéntica larga de frontera. Solución: el ULB universal como seguro, un override individual, límites de loop y compuertas de aprobación.
El caché nunca acierta	Causa: prefijo inestable (instrucciones editadas a mitad de sesión) o trabajo fragmentado en sesiones cortas. Solución: contexto estable arriba, agrupar el trabajo relacionado.
El combinado no coincide con la suma	Síntoma: alguien sumó 70% + 80% + 50% y prometió 200%. Solución: las bandas se componen multiplicativamente. Use 20 a 30% austero, 55 a 70% maduro, y confirme contra la línea de base.

Peligro: sin ULB, un solo loop accidental puede consumir el pool de la unidad. El budget por usuario es un seguro barato; configúrelo antes de que el uso crezca.

MEDICIÓN · LA CAPA QUE FALTABA

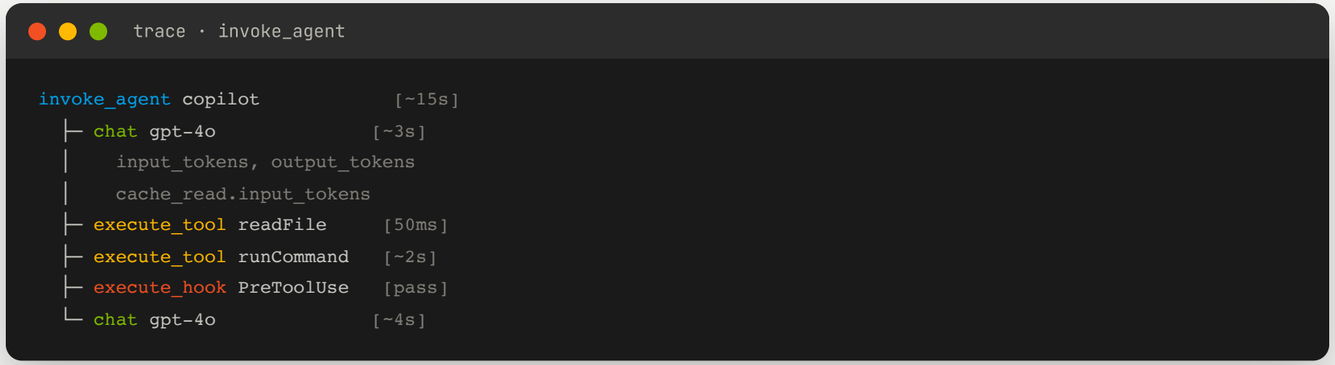
VIII

Observabilidad con OpenTelemetry.

Optimizar sin medir es adivinar. Desde VS Code 1.119, GitHub Copilot Chat y GitHub Copilot CLI exportan traces, métricas y eventos vía OTel, con tokens, costo y caché por sesión. Apagado por defecto, cero overhead hasta que lo enciendas.

OPENTELEMETRY · LO QUE PASAS A VER

Cada interacción se vuelve un árbol de spans, con tokens y caché medidos.



TOKENS POR LLAMADA

gen_ai.usage.input_tokens, output_tokens y cache_read.input_tokens. Ves exactamente a dónde va el crédito.

ATRIBUCIÓN POR EQUIPO

OTEL_RESOURCE_ATTRIBUTES marca team.id y department. Filtra el costo por equipo o squad.

CONVENCIONES GENAI

Todas las señales siguen las OTel GenAI Semantic Conventions: funciona en cualquier backend OTLP.

Las métricas incluyen gen_ai.client.token.usage y costo por modelo. Los eventos cubren aceptación de edición, supervivencia del código y feedback. Por defecto, no se captura contenido de prompt: solo metadatos como modelo, tokens y duración.

OPENTELEMETRY · ENCENDER EN DOS LUGARES

En VS Code y en el GitHub Copilot CLI. Apunta a un backend y ve los traces.

VS Code · settings.json

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

Enciende los traces del agente de primer plano, del GitHub Copilot CLI en background y del Claude agent. El mismo setting cubre los tres.

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
```

El JSONL local alimenta herramientas de costo de la comunidad que leen ~/.copilot/otel y suman tokens por sesión.

Aspire Dashboard, local, cero nube | Jaeger | Grafana + App Insights | Langfuse

Apagado por defecto, sin phone-home: los datos van solo a donde apuntas. Sin captura de contenido por defecto. Grafana Managed tiene un dashboard listo para tokens de entrada y salida, sesiones, tool calls y tiempo de respuesta por modelo.

PALANCAS NATIVAS · EN EL FLUJO DE VS CODE

Seis gestos en el editor que cortan tokens sin tocar configuración.

PLANIFICAR, LUEGO IMPLEMENTAR

Usa el Plan agent (reasoning) para el plan, apruébalo, y pásalo a un agente rápido para ejecutar. Menos ida y vuelta, menos retrabajo.

NUEVO CHAT POR TAREA

Al cambiar de tema, abre una sesión nueva. El historial irrelevante se reprocesa cada turno y quema tokens en vano.

/COMPACT

En una sesión larga, resume las partes viejas y recupera espacio de contexto. Acepta foco: /compact enfócate en las decisiones de API.

/FORK

Para explorar una alternativa, bifurca la conversación en vez de reiniciar. Hereda el contexto, sin reestablecer todo.

DESACTIVAR HERRAMIENTAS

Cada tool call consume contexto. Configure Tools desactiva MCP servers y herramientas que la tarea actual no necesita.

THINKING EFFORT EN EL DEFECTO

Más reasoning genera más thinking tokens. El defecto adaptativo basta en la mayoría; súbelo solo para un problema realmente complejo.

Inspecciona con los Agent Debug Logs: el Summary muestra tokens agregados de la sesión, y el Cache Explorer muestra la tasa de acierto del prompt cache y cuántos tokens de entrada se reutilizaron. Excluir build outputs vía .gitignore quita basura del índice.

MEDICIÓN · LEER EL DASHBOARD

Atribuye por usuario, modelo y feature. Ajusta la banda con el dato real.

01

Captura la línea de base

En la pestaña Insights, ventana de 28 días. Exporta en NDJSON y CSV. Sin línea de base, toda banda es un chiste.

02

Atribuye el consumo

Quiébralo por usuario, modelo y feature. El pico suele estar en un puñado de usuarios o en una feature agéntica.

03

Relee cada ciclo

Ajusta la banda contra el dato real, no contra la expectativa. La meta es tu curva, medida, no la del slide.

> Dos niveles de medición se completan: el dashboard de Insights da la visión de gestión, agregada y por ciclo; el OpenTelemetry da la visión de ingeniería, span a span, con tokens y caché por sesión. Uno responde cuánto y quién; el otro responde dónde y por qué.

La experiencia de preview de la factura, en el Billing Overview, muestra cómo el costo se desplaza en el nuevo modelo antes de volverse cobro. Léela antes de que el uso crezca.

REFERENCIA CONSOLIDADA · TODAS LAS PALANCAS

El mapa completo de impacto, banda y tipo de fuente.

PALANCA	BANDA	SOBRE QUÉ ACTÚA	FUENTE
Output control	40 a 70% salida	La clase de token más cara	Precio oficial
Model routing	40 a 70% cuenta	Toda la cuenta, limita el radio	Investigación
Alcance de contexto	40 a 80% entrada	La entrada enviada por pedido	Investigación
Caché	30 a 50% entrada	La entrada repetida en loops	Precio oficial
Modelos locales	un dígito a 15%	La rutina, a costo cero de créditos	Direccional
Primitivos + gobernanza	compuesto	Vuelve la ganancia permanente	Direccional
Combinado	20-30% / 55-70%	Composición multiplicativa, austero a maduro	Direccional

Más la medición con OTel y las palancas nativas (plan, nuevo chat, /compact, /fork, desactivar herramientas, thinking effort), que cortan tokens por sesión sin entrar en la tabla de bandas. Recuerda: las bandas se componen, nunca se suman.

LA DECISIÓN · COSTO DEL PROGRAMA Y RETORNO

El programa es barato. La ganancia se compone y es permanente.

LO QUE EL PROGRAMA CUESTA

Algunas horas para el primer corte: instrucciones, Auto, ULB.

Disciplina de sesión, que se vuelve hábito en una o dos semanas.

Los primitivos, escritos una vez y revisados por PR como código.

No es un proyecto, es configuración más hábito. El costo es tiempo, no una licencia nueva.

LO QUE EL PROGRAMA DEVUELVE

55 a 70%

de reducción en un programa maduro, mes tras mes

La ganancia no es un corte único, es una pendiente. Cada palanca dobla la curva, y la gobernanza traba el resultado para que no se evapore en la primera semana ocupada.

La asimetría es la tesis: la inversión es pequeña y única; el ahorro es compuesto y recurrente. El payback de cortar la salida y poner el ULB llega en el primer ciclo de factura.

LA DECISIÓN · EL CASO, EN NÚMEROS

Cómo se componen las bandas, paso a paso, en una línea de base ilustrativa.

Línea de base mensual	punto de partida	1000
Output control	× (1 - 0,40)	600
Model routing	× (1 - 0,30)	420
Alcance de contexto	× (1 - 0,15)	357
Caché en loops	× (1 - 0,10)	321

Resultado compuesto

68%

de reducción, dentro de la banda madura de 55 a 70%. No es la suma de 40 más 30 más 15 más 10, que daría 95.

Números ilustrativos, para mostrar la mecánica de la composición. Los porcentajes de cada paso son ejemplos dentro de las bandas; tu resultado es lo que midas contra tu línea de base. La lección es la forma de la cuenta, no los dígitos.

PARTE

VI

El programa en siete imágenes.

Para cualquier audiencia. Cada palanca tiene una metáfora que cabe en una frase y deja el jargón técnico de lado.

METÁFORAS · LAS SIETE IMÁGENES

De la carta al telegrama, del camión al interruptor diferencial.

R1 El telegrama y la carta
La instrucción convierte la carta en telegrama: solo lo que cambia. 40 a 70% de la salida.

R2 El camión y el sobre
El routing elige el vehículo por el tamaño de la carga. 40 a 70% de la cuenta.

R3 Cocinar en casa
La rutina sale de la cuenta en un modelo local. Un dígito a ~15% del total.

R4 La carpeta correcta para la reunión
Llevar el archivo muerto entero confunde y cuesta. 40 a 80% de la entrada.

R5 La credencial de acceso
Después del registro, solo apoye la credencial. 30 a 50% de la entrada en loops.

R6 El pasamanos de la escalera
La disciplina se vuelve infraestructura versionada. Compuesto y permanente.

R7 El interruptor diferencial
No baja la cuenta de luz del mes, evita el incendio del pico. El ahorro viene de las otras palancas; el cap sostiene la varianza. ULB en GA desde 2026-06-01.

RESUMEN

Seis ideas para llevar.

DOS FACTORES

El costo depende de qué modelo y cuántos tokens. Toda palanca ataca uno de ellos, o ambos.

EL ORDEN IMPORTA

Salida primero (la clase más cara), después contexto, después caché. El routing limita el radio de todo.

LAS BANDAS SE COMPONEN

Nunca se suman. 20 a 30% austero, 55 a 70% maduro. Cada cifra tiene fuente declarada.

LOS PRIMITIVOS SON CÓDIGO

Instrucciones, agentes, prompts y hooks versionados. Gobernanza y ahorro en el mismo archivo.

EL ULB ES EL SEGURO

Hard stop por usuario. Configure el universal antes de que el uso crezca; un loop accidental quema el pool.

MIDA PRIMERO

La banda que vale es la suya, contra la línea de base. Optimizar sin medir es adivinar.

CIERRE

Mida primero, aplique en orden, deje que la banda se ajuste con datos reales.

CONTACTO

Paula Silva
Software Global Black Belt
paulasilva@microsoft.com

PRÓXIMO PASO

Capture la línea de base hoy
Insights, 28 días, y haga commit del copilot-instructions.md

LA REGLA DE ROUTING · ACUERDO DEL EQUIPO

```
trivial  -> clase liviana / incluida
estándar -> clase intermedia
complejo -> frontera, reasoning
nunca    -> frontera para tarea trivial
```

Microsoft + GitHub · Deck v2.1.0 · Actualizado el 2026-06-14