COST OPTIMIZATION · USAGE-BASED BILLING · GITHUB COPILOT

Cost optimization in GitHub Copilot: seven levers, four layers.

Under Usage-Based Billing, cost depends on just two factors: which model and how many tokens. This is the reference for the seven optimization levers, the evidence base behind each band, and the adoption order that carries most of the result.

AUTHOR
Paula Silva

ROLE
Software Global Black Belt

DURATION
110 to 130 minutes

DATE
2026-06-14

AGENDA

From the cost equation to the guard-rail.

01 The cost equation, the three token types, and why the bands do not add up

02 The evidence base, each band with its declared source

03 The architecture: seven levers in four layers, from the most expensive to the structural

04 Versioned primitives, budgets, and procedures R1 to R7

05 Local models with Foundry Local, task routing and troubleshooting

06 Measurement with OpenTelemetry, native editor levers and the impact map

07 The program in seven images, for any audience

PART 01

The cost equation.

Under Usage-Based Billing, consumption is measured in AI Credits, where 1 AI Credit equals US$ 0.01.
The cost of each interaction depends on two factors, and only two: which model and how many tokens.

THE EQUATION · THE THREE TOKEN TYPES

Every lever attacks one of these two factors, or both.

INPUT TOKENS

Prompt, history, files and instructions. The base that context and cache attack.

OUTPUT TOKENS

What the model generates, typically 4 to 5x the input price per token. The most expensive class, the first target.

CACHED TOKENS

Reused context, at 10 to 50% of a new token: Anthropic 0.1x, OpenAI 50% automatic over a 1,024-token prefix.

COMPLETIONS ARE FREE

Code completions and Next Edit Suggestions remain included in the plan and consume no AI Credits. Optimizing completions is wasted effort.

THE REAL CONSUMPTION

Lives in chat and agentic tasks. That is where every lever in this deck acts.

THE EQUATION · WHY THEY DO NOT ADD UP

The bands compose multiplicatively. They never add.

Each lever acts on a different token base: output control on the output, context and cache on the input, routing on the whole account. That is why adding 70% + 80% + 50% and promising 200% is an arithmetic error.

17 a 25%
Austere start

46 a 58%
Mature program

ORDER MATTERS

1. Output is the most expensive class: output control is the first move.
2. Context compression is second.
3. Cache is third.
4. Routing is the architectural decision that limits the blast radius of everything.

Directional bands, to be confirmed against your baseline. The number that counts is yours, measured.

CONTEXT · WHAT CHANGED ON 2026-06-01

VII

The reality of the bill.

Usage-Based Billing reached GA. Before, premium requests with multipliers. Now, AI Credits tied to tokens. Agents stopped being a feature and became compute.

THE BILL · WHAT COUNTS AND WHAT DOESN'T

Autocomplete stays free. Everything else became metered compute.

CONSUMES NO CREDITS

- ✓ Code completions, the inline autocomplete, remain unlimited on paid plans.
- ✓ Next Edit Suggestions (NES), the next-edit hints, also unlimited.
- ✓ BYOK and local models: billed by the provider, outside AI Credits.

METERED IN AI CREDITS

- Chat, multi-file editing and all agentic sessions.
- GitHub Copilot code review, now agentic, which also consumes GitHub Actions minutes on top of credits.
- Repository analysis, multi-step workflows and the cloud coding agent.

Monthly plans migrated automatically on 2026-06-01. Annual plans stay on the premium-request model until renewal, but with higher multipliers for frontier models. The mental rule: autocomplete is still a feature; an agent is compute.

THE BILL · PLANS AND MIGRATION

Monthly migrated on its own. Annual ages on the old, pricier model.

PLANS AND INCLUDED ALLOWANCE

PLAN	PRICE	CREDITS
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: upgrade only, for existing GitHub Copilot users. Business and Enterprise: per-user credits, pooled at the entity. Promotion until 2026-09-01: Business 3,000, Enterprise 7,000.

MONTHLY PLAN

Migrated automatically on 2026-06-01 to AI Credits. No action needed.

ANNUAL PLAN

Stays on premium requests until renewal, with the frontier multipliers that rose on 2026-06-01:

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

Every plan includes base credits, fixed and equal to the price, plus a variable flex allotment that GitHub adjusts as model economics change. 1 AI Credit = US$ 0.01. When the pool runs out, usage continues at the per-credit price, subject to budget, or is blocked until the next cycle.

PART 02

II

The evidence base.

Each band has a declared source. Three kinds: official provider pricing (the hardest), published and reviewed research, and directional field estimates, always labeled as such.

EVIDENCE · EACH BAND, ITS SOURCE

The honesty rule: where the source is official pricing, the number is a fact.

LEVER	BAND	SOURCE
Output control	40 to 70% of output	Output/input ratio of 4 to 5x in the public provider tables and GitHub's. The specification ladder is a field estimate.
Model routing	40 to 70% of account	FrugalGPT (Stanford, TMLR): a cascade matches the best model with 50 to 98% savings. RouterBench (2024). Conservative band.
Local models	single digit to 15%	Official mechanics: BYOK consumes no AI Credits (GitHub Changelog, 2026). The share of total cost is a directional estimate.
Context scope	40 to 80% of input	LLMLingua (Microsoft, EMNLP 2023): up to 20x compression with a ~1.5 point drop. Context engineering survey (2025).
Cache	30 to 50% of input	Official pricing: cache reads at 0.1x on Anthropic and 50% on OpenAI. The loops band comes from documented practice.
Combined	20-30% / 55-70%	Composition arithmetic over the prior bands, consistent with the literature ceiling (FrugalGPT, 50 to 98%).

Primary sources: FrugalGPT, LLMLingua, RouterBench, Anthropic and OpenAI prompt caching (official docs), GitHub AI model comparison.

PART 03

III

The architecture.

Seven levers in four layers. Read top to bottom, which is also the adoption order. The Output layer is where the most expensive waste lives; Governance is what makes the gain permanent.

ARCHITECTURE · FOUR LAYERS

From the most expensive to the structural.

Output
The most expensive token class, ~5x the input

Output control · Concise instructions · Diff only

Account
The right model per task limits the blast radius

Model routing · Auto · Local models

Input
Send only what matters, never reprocess the repeated

Context scope · Cache · Memory

Governance
What becomes a versioned file and guard-rail becomes permanent

Primitives · Budgets · Measurement

Dev and admin: Output and Input belong to the developer. Account is shared, the dev's picker and the admin's policy. Governance belongs to the admin.

THE FIRST CUT · IN ONE HOUR

Five steps carry most of the result.

01	**Install and sign in**	VS Code with the GitHub Copilot and GitHub Copilot Chat extensions, on the account with the expected plan.
02	**Turn on measurement**	Admin enables the GitHub Copilot usage metrics policy and captures the baseline in Insights (28 days, NDJSON and CSV).
03	**Cut the output**	Create copilot-instructions.md with the direct-output directive and commit. The biggest lever.
04	**Standardize Auto**	Model picker on Auto (10% chat discount on paid plans) and turn off premium outside chat.
05	**Set the guard-rail**	Admin creates the universal user-level budget above the per-license value, with alerts at 75, 90 and 100%.

Steps 3 and 4 are the two biggest levers: output costs ~5x the input per token, and the price between model classes spans two orders of magnitude. Do both before any refinement.

REFERENCE · REPOSITORY PRIMITIVES

Each primitive is governance and savings at once.

.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

The first and most important. General context injected into every request. Concise, high-signal.

***.instructions.md**

Scoped rules via the applyTo field (glob). Curates context per code area.

***.agent.md**

One agent, one task, with restricted model and tools. Scope and model governance in one file.

***.prompt.md**

Encapsulates a long instruction in a versioned command. Kills the memorized magic prompt.

In VS Code, generate the files with /create-instruction, /create-agent, /create-prompt, /create-hook. A primitive is code: review it by PR.

GOVERNANCE · HOOKS AS A GUARD-RAIL

The circuit breaker at the action level, not just the budget.

The ULB stops the bill at the budget level. Hooks stop the waste one level earlier: at the agent's action. A PreToolUse hook can block a dangerous or redundant tool call; a Stop hook ends a loop before it burns the pool. They are generated in VS Code with /create-hook.

`PreToolUse`	Runs before a tool. Decides pass, block or non_blocking_error.
`Stop`	Ends the agentic session. The brake on the long frontier loop.

WHY IT MATTERS FOR COST

The agent loop is the antipattern that burns the most credit at once. With no brake, it repeats expensive calls until the budget bursts. The hook is the cheapest insurance against the worst case.

In OpenTelemetry, the execute_hook span records each decision as pass or block: you audit the guard-rail in production.

Generate via /create-hook, don't copy a fixed schema: the format evolves with the extension. The hook is a primitive, it comes in by PR and is reviewed like any other config file.

REFERENCE · BUDGETS

Four budget levels. The user-level is the most important control.

USER-LEVEL BUDGET (ULB) REQUIRED

Hard stop per cycle. Set the universal one above the per-license value (US$ 19 Business, US$ 39 Enterprise) and overrides for heavy users.

COST-CENTER BUDGET

Caps the metered spend of a business unit after the pool. Assigns cost per unit.

ENTERPRISE SPENDING LIMIT

Global failsafe: caps total metered spend after the pool. The enterprise's last safeguard.

ORGANIZATION BUDGET

Tracks the spend of an organization or repository. Visibility, not just a cap.

Allowances since 2026-06-01: Pro 1,500 credits, Pro+ 7,000, GitHub Copilot Max 20,000; Business 1,900 and Enterprise 3,900 per user, with a promotion until 2026-09-01. Use the promotion to find the baseline, do not read it as the permanent regime. Per-user budgets in GA since 2026-06-01.

PART 05

IV

The procedures.

R1 to R7, condensed. Each recipe points to the lever, the role that runs it, and the concrete gesture in the editor.

PROCEDURES · R1 AND R2

Output control and model routing: the two biggest levers.

R1 · Output control (Dev)

.github/copilot-instructions.md

```
- Respond with code or diff.
- No preamble, no closing summary.
- The smallest correct change; only the altered lines.
```

In Chat, expand References: the instructions file should appear as an applied reference.

R2 · Model routing (Dev + Admin)

Auto as the default
Model picker on Auto. Frontier only by manual change, for hard reasoning.

Cut the silent premium
Visual Studio: turn off Enhance non-chat requests with premium models. VS Code: utility model on a light class.

Pin the policy
The admin defines which models members can use. Changing the chat model does not change the inline suggestions model.

PROCEDURES · R3 TO R5

Local models, context scope, and the cache that never reprocesses.

R3 · LOCAL MODELS

BYOK enabled by default. Register Ollama or Foundry Local; commits, boilerplate and tests go local, at zero credit cost.

`Agent mode requires models with tool calling.`

R4 · CONTEXT SCOPE

Prefer #file, #sym and #changes to dumping the whole #codebase. Content exclusion (admin) for sensitive paths.

`Not supported in Edit and Agent mode.`

R5 · CACHE AND MEMORY

Stabilize the prefix: instructions on top, no mid-session edits. Group related work.

`Thrash breaks the cache.`

R6 Primitives: adoption order is general instructions, scoped instructions, custom agents, prompt files, hooks. R7 Budgets: the universal ULB first, then overrides and cost centers, and measure every cycle.

INPUT · GITHUB COPILOT MEMORY

Memory curates context across sessions, so you don't re-explain.

GitHub Copilot Memory, in public preview, captures facts and preferences from your work and reuses them in later sessions. It is an input lever: what memory holds, you don't have to send again in the prompt every conversation. Less repeated input, fewer tokens.

Enable and review
See the captured facts and preferences. Memory only helps if it reflects the real project.

What ages out, goes
Stale memory becomes expensive noise: it enters the context on every request. You can, remove it.

THE IMAGE

The project notebook.

A good colleague notes what matters about your project once, and doesn't ask again next week. Memory is that notebook: maintained, it saves; abandoned, it gets in the way.

Memory complements the versioned primitives: copilot-instructions.md holds the stable, PR-reviewed rule; memory holds what is learned in the session.
Both reduce the input, by different paths.

INPUT · CACHE IN PRACTICE

The same work, with and without cache discipline.

BEFORE · THE CACHE NEVER HITS

- Instructions edited mid-session: the prefix changes, the cache invalidates.
- Work fragmented across short sessions: each one starts over.
- Result: you pay the whole input, full price, on every request.

100% of input

full price, always

AFTER · STABLE PREFIX

- ✓ Instructions and AGENTS.md on top, no mid edits: the prefix stays stable.
- ✓ Related work grouped in the same session: the context is reused.
- ✓ The cache read costs a fraction: Anthropic 0.1x, OpenAI 50% over the prefix.

10 a 50%

of a new token, on the cached part

The difference is not the model, it's session hygiene. The Cache Explorer, in the Agent Debug Logs, shows the hit rate and how many input tokens were reused: that's where you confirm the discipline is working.

LOCAL MODELS · FOUNDRY LOCAL AND AI TOOLKIT

The zero-cost edge: run the routine on your own machine.

BYOK consumes no AI Credits: it is billed by the provider, or free when the model runs local. The AI Toolkit connects GitHub Copilot Chat to Azure AI Foundry models or to Foundry Local on your machine. Four steps in the editor.

01 Install the GitHub Copilot Chat and AI Toolkit extensions in VS Code.

02 In the Chat model picker, choose Add model and select Foundry Local via AI Toolkit.

03 The AI Toolkit downloads the model if it is not on the machine yet. No API key: the credential is generated locally.

04 Switch between local and cloud in the picker, per task. The routine goes local, the frontier stays in the cloud.

terminal · Foundry Local

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

TWO REQUIREMENTS

Agent mode requires models with tool calling. And hardware matters: the GPU defines local model performance. BYOK does not apply to completions.

BYOK is on by default in Business and Enterprise, with models from providers like Anthropic, Gemini, OpenAI, OpenRouter and Azure, plus Ollama and Foundry Local. Once configured, the model appears everywhere in Chat, including the Plan agent and custom agents.

ROUTING · WHICH TASK GOES WHERE

The practical rule: the right model per task type.

TASK	MODEL CLASS	WHERE
Commits, boilerplate, test scaffolding	`Light / local`	Foundry Local, zero cost
Explain code, factual questions, autocomplete	`Included / light`	Completions are free
Standard feature generation, local refactoring	`Intermediate`	Auto in the picker
Architecture planning, design decisions	`Frontier, reasoning`	Plan agent, then switch
Multi-file refactoring, multi-step debugging	`Frontier, reasoning`	Manual, restricted scope
Never: frontier for a trivial task	`Antipattern`	A truck for an envelope

The picker shows cost on hover: cost per token type and a tier label (Low, Medium, High). Custom agents can pin a cheap model per subtask: when invoked as a subagent, they use their own model, not the session's.

ROUTING · CONTROLS DIFFER BY EDITOR

Know your editor's controls. They are not the same.

VS CODE

The most complete set: Auto, utility model, Plan agent, /compact, /fork, Configure Tools, OpenTelemetry and the Agent Debug Logs.

All the levers in this deck.

VISUAL STUDIO

The silent premium lives here: Tools, Options, GitHub, GitHub Copilot, Editor. Turn off Enhance non-chat requests with premium models.

The setting that surprises the bill most.

JETBRAINS AND OTHERS

Telemetry may be less consistent outside VS Code. Keep the IDE on the latest version so the dashboard reflects real usage.

Update before you trust the numbers.

The admin's model policy applies in all editors, but the flow gestures (new chat, fork, compact) and observability are richer today in VS Code. Standardize the editor where the FinOps program needs the most control.

PART 06

V

Troubleshooting.

Antipatterns: symptom, cause, solution. The four problems that show up before any other, and how to diagnose them.

TROUBLESHOOTING · SYMPTOM AND SOLUTION

Four antipatterns, four diagnoses.

The bill rose with no usage change	Cause: out-of-chat requests enhanced with premium models, or the utility model pointed at an expensive one. Solution: turn off the out-of-chat enhancement and point the utility model at the light class.
One user burned the pool	Cause: an agent loop or a long frontier agentic session. Solution: the universal ULB as insurance, an individual override, loop limits and approval gates.
The cache never hits	Cause: an unstable prefix (instructions edited mid-session) or work fragmented across short sessions. Solution: stable context on top, group related work.
Combined does not match the sum	Symptom: someone added 70% + 80% + 50% and promised 200%. Solution: the bands compose multiplicatively. Use 20 to 30% austere, 55 to 70% mature, and confirm against the baseline.

Danger: without a ULB, a single accidental loop can consume the unit's pool. The per-user budget is cheap insurance; set it before usage grows.

MEASUREMENT · THE MISSING LAYER

VIII

Observability with OpenTelemetry.

Optimizing without measuring is guessing. Since VS Code 1.119, GitHub Copilot Chat and GitHub Copilot CLI export traces, metrics and events via OTel, with tokens, cost and cache per session. Off by default, zero overhead until you turn it on.

OPENTELEMETRY · WHAT YOU START TO SEE

Each interaction becomes a span tree, with tokens and cache measured.

trace · invoke_agent

```
invoke_agent copilot            [~15s]
  ├─ chat gpt-4o              [~3s]
  │    input_tokens, output_tokens
  │    cache_read.input_tokens
  ├─ execute_tool readFile     [50ms]
  ├─ execute_tool runCommand   [~2s]
  ├─ execute_hook PreToolUse   [pass]
  └─ chat gpt-4o              [~4s]
```

TOKENS PER CALL

gen_ai.usage.input_tokens, output_tokens and cache_read.input_tokens. You see exactly where the credit goes.

ATTRIBUTION BY TEAM

OTEL_RESOURCE_ATTRIBUTES tags team.id and department. Filter cost by team or squad.

GENAI CONVENTIONS

All signals follow the OTel GenAI Semantic Conventions: it works on any OTLP backend.

Metrics include gen_ai.client.token.usage and cost per model. Events cover edit acceptance, code survival and feedback. By default, no prompt content is captured: only metadata like model, tokens and duration.

OPENTELEMETRY · TURN IT ON IN TWO PLACES

In VS Code and in the GitHub Copilot CLI. Point at a backend and see the traces.

VS Code · settings.json

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

Turns on traces for the foreground agent, the background GitHub Copilot CLI and the Claude agent. The same setting covers all three.

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
```

The local JSONL feeds community cost tools that read ~/.copilot/otel and sum tokens per session.

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

Off by default, no phone-home: data goes only where you point it. No content capture by default. Grafana Managed has a ready dashboard for input and output tokens, sessions, tool calls and response time per model.

NATIVE LEVERS · IN THE VS CODE FLOW

Six editor gestures that cut tokens without touching configuration.

PLAN, THEN IMPLEMENT

Use the Plan agent (reasoning) for the plan, approve it, and hand off to a fast agent to execute. Less back-and-forth, less rework.

NEW CHAT PER TASK

When you switch topics, open a new session. Irrelevant history is reprocessed every turn and burns tokens for nothing.

/COMPACT

In a long session, summarize the old parts and reclaim context space. Accepts focus: /compact focus on the API decisions.

/FORK

To explore an alternative, fork the conversation instead of restarting. It inherits the context, no need to re-establish everything.

DISABLE TOOLS

Each tool call consumes context. Configure Tools turns off MCP servers and tools the current task does not need.

THINKING EFFORT AT DEFAULT

More reasoning generates more thinking tokens. The adaptive default is enough for most; raise it only for a genuinely complex problem.

Inspect with the Agent Debug Logs: the Summary shows aggregate session tokens, and the Cache Explorer shows the prompt cache hit rate and how many input tokens were reused. Excluding build outputs via .gitignore removes junk from the index.

MEASUREMENT · READ THE DASHBOARD

Attribute by user, model and feature. Adjust the band with real data.

01

Capture the baseline

In the Insights tab, a 28-day window. Export in NDJSON and CSV. Without a baseline, every band is a guess.

02

Attribute the consumption

Break it down by user, model and feature. The spike is usually in a handful of users or one agentic feature.

03

Re-read every cycle

Adjust the band against real data, not against expectation. The target is your curve, measured, not the slide's.

> Two levels of measurement complete each other: the Insights dashboard gives the management view, aggregate and per cycle; OpenTelemetry gives the engineering view, span by span, with tokens and cache per session. One answers how much and who; the other answers where and why.

The preview bill experience, in the Billing Overview, shows how cost shifts under the new model before it becomes a charge. Read it before usage grows.

CONSOLIDATED REFERENCE · ALL THE LEVERS

The complete map of impact, band and source type.

LEVER	BAND	ACTS ON	SOURCE
Output control	40 to 70% output	The most expensive token class	Official price
Model routing	40 to 70% account	The whole account, limits the radius	Research
Context scope	40 to 80% input	The input sent per request	Research
Cache	30 to 50% input	The input repeated in loops	Official price
Local models	single digit to 15%	The routine, at zero credit cost	Directional
Primitives + governance	compound	Makes the gain permanent	Directional
Combined	20-30% / 55-70%	Multiplicative composition, austere to mature	Directional

Plus OTel measurement and the native levers (plan, new chat, /compact, /fork, disable tools, thinking effort), which cut tokens per session without entering the band table. Remember: the bands compose, never add.

THE DECISION · PROGRAM COST AND RETURN

The program is cheap. The gain compounds and is permanent.

WHAT THE PROGRAM COSTS

A few hours for the first cut: instructions, Auto, ULB.

Session discipline, which becomes habit in a week or two.

The primitives, written once and reviewed by PR like code.

It's not a project, it's configuration plus habit. The cost is time, not a new license.

WHAT THE PROGRAM RETURNS

55 a 70%

reduction in a mature program, month after month

The gain is not a one-time cut, it's a slope. Each lever bends the curve, and governance locks the result so it doesn't evaporate in the first busy week.

The asymmetry is the thesis: the investment is small and one-time; the savings are compound and recurring. The payback of cutting output and setting the ULB arrives in the first billing cycle.

THE DECISION · THE CASE, IN NUMBERS

How the bands compose, step by step, on an illustrative baseline.

Monthly baseline	starting point	1000
Output control	× (1 - 0,40)	600
Model routing	× (1 - 0,30)	420
Context scope	× (1 - 0,15)	357
Cache in loops	× (1 - 0,10)	321

Composed result

68%

reduction, within the mature 55 to 70% band. It is not the sum of 40 plus 30 plus 15 plus 10, which would give 95.

Illustrative numbers, to show the mechanics of composition. Each step's percentages are examples within the bands; your result is what you measure against your baseline. The lesson is the shape of the math, not the digits.

PART 08

VI

The program in seven images.

For any audience. Each lever has a metaphor that fits in one sentence and drops the technical jargon.

METAPHORS · THE SEVEN IMAGES

From letter to telegram, from truck to circuit breaker.

R1 **The telegram and the letter**
The instruction turns the letter into a telegram: only what changes. 40 to 70% of output.

R2 **The truck and the envelope**
Routing picks the vehicle by the load size. 40 to 70% of the account.

R3 **Cooking at home**
The routine leaves the bill on a local model. Single digit to ~15% of the total.

R4 **The right folder for the meeting**
Bringing the whole dead archive confuses and costs. 40 to 80% of input.

R5 **The access badge**
After check-in, just tap the badge. 30 to 50% of input in loops.

R6 **The handrail on the stairs**
Discipline becomes versioned infrastructure. Compound and permanent.

R7 **The circuit breaker**
It does not lower the month's electric bill, it prevents the fire at the spike. The saving comes from the other levers; the cap holds the variance. ULB in GA since 2026-06-01.

SUMMARY

Six ideas to take home.

TWO FACTORS

Cost depends on which model and how many tokens. Every lever attacks one of them, or both.

ORDER MATTERS

Output first (the most expensive class), then context, then cache. Routing limits the radius of everything.

BANDS COMPOSE

They never add. 20 to 30% austere, 55 to 70% mature. Each figure has a declared source.

PRIMITIVES ARE CODE

Versioned instructions, agents, prompts and hooks. Governance and savings in the same file.

THE ULB IS THE INSURANCE

Hard stop per user. Set the universal one before usage grows; an accidental loop burns the pool.

MEASURE FIRST

The band that counts is yours, against the baseline. Optimizing without measuring is guessing.

CLOSING

Measure first, apply in order, let the band adjust with real data.

CONTACT

Paula Silva
Software Global Black Belt
paulasilva@microsoft.com

NEXT STEP

Capture the baseline today
Insights, 28 days, and commit copilot-instructions.md

THE ROUTING RULE · TEAM AGREEMENT

```
trivial  -> light / included class
standard -> intermediate class
complex  -> frontier, reasoning
never    -> frontier for a trivial task
```

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