TOKEN ECONOMICS · GITHUB COPILOT

Getting more from every token. Token optimization and agent quality for the AI Credits era.

An executive and technical briefing on usage-based billing, what it changes for your engineering organization, and how to operate well under it.

AUTHOR
Paula Silva

ROLE
Software Global Black Belt

DURATION
60 to 90 minutes

DATE
2026-06-14

AGENDA

Seven parts, one frame.

PART I	The shift, AI Credits replace per-request billing on June 1, 2026
PART II	Foundations, how LLMs, agents, tokens, and context windows actually work
PART III	Three surfaces, VS Code, GitHub Copilot CLI, and GitHub.com, one wallet
PART IV	Agent quality first, why quality beats counting tokens
PART V	The six levers of token optimization, and how they compound
PART VI	Apply and govern, context engineering, budgets, observability, adoption
PART VII	Annex, caching, the GitHub Copilot prompt cache versus Foundry plus Redis

PART

I

The shift.

On June 1, 2026, GitHub Copilot stops charging per request. Tokens become the unit of cost.

THE CUTOVER DATE

2026-06-01

Premium Request Units retire. GitHub AI Credits take over.

1 AI Credit equals US$ 0.01. Input, output, and cached tokens are metered at each model's published rate. Cost is now tied to the work performed, not to how often you ask.

OLD UNIT		NEW UNIT
1 premium request	→	tokens, 1 cr = US$ 0.01

BEFORE AND AFTER

The unit of cost moved from intent to work.

BEFORE · PREMIUM REQUESTS

Cost per user intent

A one-word prompt and a forty-turn agent session counted the same, one premium request each. Predictable to budget, but disconnected from the compute actually consumed.

AFTER · AI CREDITS

Cost per work performed

Input, output, and cached tokens at each model's rate. A long agentic session is genuinely far more expensive than one chat question. The bill now matches reality.

THE BILLING EQUATION

Two factors, and you control both.

THE EQUATION

cost = model times tokens

cost = input times in-rate, plus cached times cache-rate, plus output times out-rate. 1 AI Credit = US$ 0.01. Every tactic in this deck acts on the model or the tokens.

WHAT CONSUMES CREDITS

Heavy work, not autocomplete

Chat, agents, the coding agent, and premium models consume credits. Code completions and Next Edit Suggestions do not. GitHub Copilot code review costs twice: AI Credits plus GitHub Actions minutes.

WHAT IS NOT CHANGING

Base prices hold, and the everyday experience stays free.

01

Prices stay the same

Pro US$ 10, Pro+ US$ 39, Business US$ 19, Enterprise US$ 39 per user. Each license adds its dollar value as credits to a shared pool.

02

Completions stay free

Code completions and Next Edit Suggestions remain unlimited and consume no AI Credits. The day-to-day inline experience is unchanged.

03

Only heavy work meters

Chat, agent mode, the coding agent, Spaces, and premium models consume credits. That is where the meter actually spins.

FROM BUFFET TO A LA CARTE

Not an unfair charge. The return of a control that never existed.

01

The buffet hid the price

The expensive dish always cost a lot to make. The flat buffet hid the cost of each plate and quietly rewarded waste.

02

The menu has prices now

Usage-based billing hands you the a la carte menu with prices printed. The kitchen did not change. Now you can choose.

03

The market already moved

FinOps 2026: 98 percent of organizations now manage AI spend, up from 31 percent in 2024. This is market maturity, not a GitHub quirk.

WHY PRICING IS EVOLVING

Token-based billing matches how AI compute actually works.

01

Tokens, not requests

Providers bill GitHub in tokens. A request can be a few hundred or tens of thousands. Equal-request pricing never matched the cost.

02

The market moves fast

Labs release models and reprice constantly. A token framework absorbs that without a pricing overhaul each time.

03

It unlocks features

Larger context windows and richer agentic workflows only make economic sense when cost is metered by consumption.

THE TRANSITION TIMELINE

Plan against September, not against today's cushioned bill.

ANNOUNCE AND PREVIEW	Announced April 27. The May billing preview is your window to instrument monitoring and configure budgets before anything bills.	Apr to May
NATIVE CUSHION	Usage-based billing goes live June 1 with extra included headroom that softens the first months. Use it as a learning window for real consumption.	Jun 1 to Aug 31
REAL BILL LANDS	The cushion expires August 31. September is the first un-cushioned month. Re-baseline budgets here, never against promotional spend.	From September

WHY IMPACT VARIES SO WILDLY

The bill is a mirror of platform maturity, not a verdict.

01

Maturity, not size

Two companies feel the same change in opposite ways. The difference is usage maturity, not headcount.

02

The mirror, not a verdict

Teams with governance pay in proportion to value. Teams without it pay for the chaos the flat price used to hide.

03

AI amplifies

DORA 2025 found AI does not fix a team, it amplifies what is already there. Usage billing simply makes that visible on the invoice.

PART

II

Foundations.

How LLMs, agents, tokens, and context windows work, the mental model the rest depends on.

THE MODEL AND THE RULE

A word-probability machine, fed by you.

01

It predicts the next token

An LLM is text in, text out. It cannot tell relevant from irrelevant, and it cannot tell a hallucination from a fact. Both come from the same math.

02

Context balance

The core rule: as little context as possible, but as much as required. Too much biases the answer and costs more, too little invites hallucination.

03

A token is a word fragment

Roughly three quarters of a word. 1M tokens is about the Lord of the Rings trilogy plus The Hobbit. Prompts, files, and replies all consume them.

THE CORE RULE

As little context as possible, but as much as required.

TOO MUCH

Biases the answer

Irrelevant information is weighed, not ignored. It pushes the model toward wrong answers, and you pay for every extra token, every turn.

TOO LITTLE

Invites hallucination

Missing critical context makes the model fill the gap, with no error message. The math makes no distinction between a fact and a fabrication.

WHAT A TOKEN IS

A token is a word fragment, and density varies.

01

Subword tokenization

Roughly three quarters of a word. English averages about 1.3 tokens per word.

02

Language matters

Portuguese and Spanish run 1.5 to 1.7 tokens per word. Code tokenizes at variable density.

03

Do not over-tune it

You have little control over tokenization. Think higher level: prompts, files, and replies all consume, and compound every turn.

WHAT AN AGENT REALLY IS

An agent is just code talking to a model, many times.

01

The harness

An agent is an application: VS Code Chat, GitHub Copilot CLI, the coding agent, even Claude Code or Codex. It talks to the model on your behalf, many times per task.

02

The LLM

The model itself is just GPT, Claude, or Gemini. The interaction is not magic, it is text, and the model is stateless.

03

Your levers

You influence it through three things: your prompt, the files in your project, and the agent configs, instructions, skills, MCP.

THE RE-SEND EFFECT

2M+

The model does not remember. It re-reads, every turn.

A 50,000-token session over 40 turns ships about 2 million input tokens, even if your last prompt was 20 words. Because LLMs are stateless, the whole conversation is re-sent each turn, so context bloat is the single biggest source of waste.

WHERE THE TOKENS ACTUALLY GO

Your prompt is the smallest slice of the bill.

01

FIXED OVERHEAD

System prompt and tool schemas

You cannot change these, but they are why every session starts north of zero.

02

GROWS EVERY TURN

Conversation history

The slice most people never think about, and the one that compounds with session length.

03

THE CARELESS ONE

Tool-call results

File reads, shell output, test runs. One careless read of a large file rides along forever.

04

SMALLEST, STILL BILLED

Your prompt and the output

Your typed prompt is usually the smallest slice. The model response, including reasoning tokens, is also billed.

THREE TOKEN TYPES, THREE BEHAVIORS

Output is the most expensive token. Cached is the cheapest.

01

Input

Everything you send: prompt, history, attachments, system prompt, tool schemas. Re-billed every turn. The baseline rate.

02

Output

What the model generates, including invisible reasoning tokens. Typically 4 to 10 times the input rate. Verbose answers cost real money.

03

Cached

A stable prefix the provider replays from cache. About 10 percent of input, a 90 percent discount. The largest lever for agentic work.

WINDOW VERSUS BILL

A bigger window is not a smaller bill.

THE CONTEXT WINDOW

What the model can hold

1 token is about three quarters of a word. 1M tokens is roughly the Lord of the Rings trilogy plus The Hobbit. The window is the capacity.

YOUR BILL

What you actually put in it

Tokens meter linearly whether you are at 20 or 80 percent. Sit at 500K on a 1M window and you pay for 500K every turn. Compaction saves you from the wall, not the cost of nearing it.

TWO FAILURE MODES OF LONG CONTEXT

Bigger context is both more expensive and lower quality.

LOST IN THE MIDDLE

A U-shaped recall curve

Models use information best at the start and end of the window, worst in the middle. Switch tasks mid-session and the model can snap back to the first instruction. Start a fresh window per task.

CONTEXT ROT

Quality decays before overflow

Chroma tested 18 frontier models. All degrade as input grows, often well before the window is full. Keep the window under roughly 60 to 70 percent.

PART

III

Three surfaces, one wallet.

One bill, three behaviors. Each surface has its own dominant cost driver and its own dials.

THE GITHUB PLATFORM AND THE GITHUB COPILOT FAMILY

One platform, two cost classes, many models.

01

The platform

GitHub is repos, pull requests, issues, and Actions. GitHub Copilot is the AI layer woven through it, reached from VS Code, the CLI, the web, and other IDEs.

02

Two cost classes

Free and unlimited: code completions and Next Edit Suggestions. Metered in AI Credits: chat, agents, the coding agent, Spaces, premium models.

03

Multi-model

The same tiers come from Anthropic, OpenAI, and Google, routed through one control plane and one billing unit, AI Credits.

ONE WALLET, THREE BEHAVIORS

Optimization is surface-specific. The physics is the same.

01

Terminal · CLI

Driver: re-send of the full history every turn. Dials: /clear, /compact, /context, specialist sub-agents.

02

Editor · VS Code

Driver: attachments plus agent tool-call sprawl. Dials: mode choice, hash-file references, instructions, MCP scope.

03

Web · GitHub.com

Driver: coding-agent runs with a large blast radius. Dials: tightly scoped issues, narrow Spaces, PR review.

THE TERMINAL

GitHub Copilot CLI rewards context hygiene.

01

Clear and compact

/clear resets between unrelated tasks, /compact summarizes mid-task, /context shows the breakdown, /usage shows what the session cost.

02

Switch model mid-session

/model lets you plan on a strong model, implement on a cheap one, review on a specialist. Each step on the cheapest tier that finishes it.

03

Specialists save tokens

/explore, /plan, /review, and /delegate run in isolated context. The main session only sees the summary, not the files they read.

THE EDITOR

Pick the cheapest mode that works, pin context narrow.

$

Ask

Read-only. Explanations, design questions, doc lookups. No file writes. The cheapest mode and the right one for most questions.

$

Plan

Spec-first. Drafts a reviewable plan you hand to Agent. The plan narrows what Agent reads and changes, which is where savings come from.

$$$

Agent

Autonomous tool-using loop. Cheapest when driven by a spec. Default to hash-file over hash-codebase, and restart rather than trim.

UNDER THE HOOD IN VS CODE

Two mechanics that quietly shape the bill.

WORKSPACE INDEX

What hash-codebase queries

Hash-codebase queries a semantic index, not a live grep. For GitHub repos it is remote and near-instant since March 2025. A cold index forces broader, costlier reads.

COMPACTION

/compact in editor chat

Summarizes older turns to free context, with a hint about what to keep. Cheaper than starting fresh when continuity matters, far cheaper than overflowing.

THE WEB

Bigger, less frequent operations, larger blast radius.

01

Coding agent

Issue to branch to PR in a sandbox VM, GA since September 2025. Issue quality dominates the bill, so scope tightly with file pointers.

02

GitHub Copilot Spaces

Curated context across repos and docs. Three files beat three repos. One Space per question shape, refresh rather than accumulate.

03

One file, three surfaces

copilot-instructions.md is honored by VS Code, the coding agent, and the CLI. Invest in it once, benefit everywhere.

SAME TASK, THREE SURFACES, THREE COST SHAPES

The cost dial is the same: how narrowly you scope before the model reads.

01

CLI

/explore, /plan, narrow scope. When you are already in the terminal for a quick edit.

02

VS Code

Plan mode drafts a spec, hand it to Agent with hash-file scope. For a mid-size change you review first.

03

GitHub.com

Issue with file pointers, assign @copilot, review the PR. For a bounded change you can step away from.

CODING AGENT, WHEN IT PAYS OFF

Bounded and well-specified, or do not pay for it.

GOOD FIT

Pays off

Bumping deps, adding tests for one module, a repo-wide rename. Issues with file pointers and acceptance criteria, in repos with strong tests and CI.

BAD FIT

Do not pay for this

Vague exploratory work, cross-cutting architecture, greenfield repos, repos without tests, or anything that should have been a ten-line conversation.

GITHUB COPILOT SPACES, CURATED CONTEXT

Three files beat three repos.

01

What is in a Space

Selected repos and specific paths, pasted notes, free-text instructions, external doc links. Shipped into context at query time.

02

Cost levers

Narrow sources, one Space per question shape, refresh rather than accumulate. Stale sources cost the same as fresh ones.

03

History

Spaces launched May 2025 and replaced Knowledge Bases in September 2025. If you still see KB, that is the legacy term.

THE WEB IS SIX PRODUCTS, ONE WALLET

Know the cost shape of each.

01

Chat and Edits

Repo-anchored Q&A and file-scoped edits. The cheapest entry points on the web.

02

PR review

Auto-review scoped to the diff. Now also consumes Actions minutes for its agentic analysis.

03

Spark and coding agent

Generate full apps, or issue-to-PR runs. The heaviest per-run, long-running generation.

CLI COMPACTION MECHANICS

Compaction protects the wall, not the bill.

01

Auto-compact at 80 to 95%

Background near 80, hard near 95, scaled to the model. It does not fire until you are close.

02

Tokens meter linearly

At 500K on a 1M window you pay 500K every turn, long before compaction. Bigger windows are not smaller bills.

03

Context first, resume often

Under 40 percent, do not pay to compact yet. Resume from a saved summary instead of pasting context back.

PART

IV

Agent quality first.

Why optimizing for quality beats optimizing for cost, and why they move in the same direction.

THE REFRAME

Stop gambling. Send fewer agents that land.

THE WRONG QUESTION

Agent gambling

Like firing 20 cheap rockets at the Moon and hoping one lands. Little context, a lazy prompt, send it, and try again if it fails. Unsustainable when every developer dispatches dozens of agents a day.

THE RIGHT QUESTION

Make every token count

Raise the value and quality of each agent before you send it. Fewer, better agents automatically means fewer tokens. Optimize for return on investment per agent.

QUALITY DRIVES VALUE DRIVES ROI

Optimizing cost when value is zero is useless work.

THE FORMULA

Agent ROI

ROI = value of output minus token cost, divided by token cost. You cannot compute it cleanly, but it still guides: driving cost to zero on a worthless output is not a win.

THE DYNAMIC

Cutting context does both

Raising an agent's value is often achieved by sending fewer tokens. Trimming irrelevant context is one lever that improves quality and lowers cost at once.

THE COMPOUNDING ERROR PROBLEM

Small per-step errors multiply across a workflow.

PER STEP

99%

Optimistic accuracy on a single step

COMPOUNDS OVER

50 steps

TO OVERALL SUCCESS

AFTER 50 STEPS

61%

Even at 99 percent per step. At 95 percent you land near 8 percent.

SHIFT-LEFT, FOR AGENTS

Deterministic controls restart the accuracy clock.

01

Tests are deterministic

A test passes or fails, no probability. A failing test stops a drifting agent and forces it to rebuild on a stable base.

02

53 percent is tests

The GitHub Copilot CLI team ships about 500 PRs a week. Their number-one practice is tests, 53 percent of the codebase.

03

Beyond tests

Linters, type checkers, and security scanners are guardrails the agent executes. Without them, it stacks buggy change on buggy change.

THE THESIS

Cost and quality move in the same direction.

COUNTING TOKENS

The trap

Squeeze tokens so hard the agent produces worse output and you have not saved money, you have paid for a worse result and the rework that follows.

MAKING THEM COUNT

The lever

Cutting irrelevant context raises quality and lowers cost at the same time. Match your effort to your maturity: the more agents you run, the more this pays.

MATCH EFFORT TO MATURITY

How hard you optimize depends on how many agents you run.

AI-ASSISTED ENGINEER

One agent, mostly sync

Around 10 agents a day, AI as an assistant. Even 50 percent savings on a 20-dollar month is only 10 dollars. Learn the fundamentals, apply the top few habits.

AI ENGINEER

Orchestrator of many

Dozens or hundreds of async agents. Every percent of tokens and quality, remember the compounding math, is multiplied across the fleet. The full framework earns its keep.

LONG-TERM TRAITS

What makes you valuable in agentic development.

01

Analytical skills

Coding was never the value, analysis was. Telling an agent precisely what to do, in the domain's language, becomes the most valuable skill.

02

Good architecture

Domain-driven design, hexagonal, CQRS, event-driven. Clean architecture gives agents guardrails and reduces misses.

03

Iterate on configs

You are a context engineer now. Treat agent misses like incidents, keep configs fresh, recreate them quarterly.

PART

V

The six levers.

Six compounding levers, each fixing one driver of token spend. Field models land the combined effect in the 37 to 44 percent band.

SIX DRIVERS OF TOKEN SPEND

Diagnose the cause before you apply the fix.

01

Context, 30 to 50%

Open files, indexing, and history sent every turn.

02

Model, 10 to 24x

Frontier versus lightweight tier, the Opus to mini spread.

03

Instructions, 10x uncached

The same project context re-sent every turn without caching.

04

Reasoning, 10 to 80x

High and max thinking traces, billed as output tokens.

05

Agents and tools, 4 to 30x

Agent loops re-bill the work, MCP schemas re-sent every turn.

06

Sessions, up to 8x

History compounds across turns, turn 30 unmanaged.

THE FRAMEWORK

Six drivers, six levers, one compounding effect.

01

Context discipline

Attach the narrowest scope that lets the model answer. The cheapest token is the one you never send.

02

Model routing

Match the model to the task. A 10 to 24 times spread across tiers is real money. Default to Auto.

03

Instruction engineering

Write team standards once into cacheable files, then reuse them at roughly 90 percent off.

04

Reasoning depth

Thinking is billed as output. Default to medium, escalate only for genuinely hard turns.

05

Agents and tools

Trim MCP overhead, specialize roles, delegate discovery to subagents. Govern the loop or the bill explodes.

06

Session lifecycle

History compounds every turn. One topic, one session. Reset, do not compound.

THE COMPOUND EFFECT

~41%

The levers stack. The same license does far more useful work.

Applied together, the levers land a roughly 41 percent token reduction with no measurable productivity loss, inside a 37 to 44 percent target band. The figures are illustrative, measure your own baseline.

TOKEN REDUCTION		USEFUL WORK PER LICENSE
37 to 44 percent	→	about 1.6 to 1.8 times

LEVERS 1 AND 2

Send less, and send it to the right model.

LEVER 1 · CONTEXT DISCIPLINE

15 times less input

Naming the 3 files that matter instead of searching the whole codebase cut a real refactor from about 25,000 to 1,700 input tokens per turn, and gave a sharper answer.

LEVER 2 · MODEL ROUTING

6 times cheaper, same feature

Plan on a powerful model, implement on a lightweight one, review on a versatile one. One OAuth feature dropped from about US$ 13.75 to US$ 2.30, same code quality.

PER-MODEL RATES, AI CREDITS PER 1M TOKENS

Model choice is a budgeting decision, not an aesthetic one.

MODEL	INPUT	OUTPUT	BEST FIT
GPT-5 mini	25 cr	200 cr	Daily chat, routine completions
GPT-5	125 cr	1,000 cr	Mid-complexity reasoning and code
Claude Sonnet 4.6	300 cr	1,500 cr	Long context, sustained code work
Claude Opus 4.7	1,500 cr	7,500 cr	Hard agent tasks, reserved use

No model is free under usage-based billing: every model, including the lightweight tier, consumes credits per token. Only code completions and Next Edit Suggestions consume no credits. Rates and model names are illustrative as of mid 2026, always validate against the official GitHub Copilot models and pricing rate card before committing budgets.

MATCH THE MODEL TO THE TASK

Three tiers, a 10 to 24 times spread.

01

Lightweight

Simple Q&A, formatting, boilerplate, simple refactors. The cheap default for routine work.

02

Versatile

Everyday coding, feature work, review, debugging. The workhorse where most spend lives.

03

Powerful

Architecture, hard debugging, security review, novel logic. Reserve it, the multiplier is steep.

LEVERS 3 AND 4

Cache the recurring prefix, pay for the thinking the task needs.

LEVER 3 · INSTRUCTION ENGINEERING

Pay once, reuse at 90 percent off

Keep team files stable so the cache holds. Append new rules at the bottom, never reorder. Scaled to 50 developers this saves on the order of US$ 32,000 a year on the instruction prefix alone.

LEVER 4 · REASONING DEPTH

Default to medium

On reasoning models, high or max effort can multiply the bill 10 to 80 times. Decompose: one medium plan, many cheap low steps, one medium review. About 7 times cheaper per task.

REASONING EFFORT IS BILLED AS OUTPUT

LOW to MAX can be up to 80 times the bill.

1x

LOW

1x. About 200 to 800 thinking tokens. Fine for most steps.

2-4x

MEDIUM

2 to 4x. About 1K to 4K. The right default for most coding.

10-25x

HIGH

10 to 25x. About 5K to 20K. Save for genuinely hard turns.

50-80x

MAX

50 to 80x. Up to 64K thinking tokens. Architectural design and subtle bugs only.

LEVERS 5 AND 6

Govern the agent loop, reset the session.

LEVER 5 · AGENTS AND TOOLS

Trim, specialize, delegate

MCP trims what is sent, hooks trim what is stored, skills skip exploration. Specialize roles and delegate discovery to subagents. A 30-turn debug went from about US$ 45 to US$ 6.

LEVER 6 · SESSION LIFECYCLE

One topic, one session

By turn 30 the history slice is about 90 percent of every turn's bill. New topic, new chat. Compact with focus, fork to explore, archive when done.

HABITS THAT STACK ON TOP

Constrain output, use Ask mode, adopt the starter kit.

01

Constrain output

Output is the priciest token. Bound it: one sentence, three bullets, code only. Code-only cuts output 40 to 70 percent.

02

Use Ask mode

Mode sets the number of calls: Ask is one, Agent is 5 to 25. Do not pay agent-loop overhead for a bounded question.

03

The starter kit

Pick three: /clear, /model, /usage. Context control, cost control, cost visibility, one for each dimension.

LEVER 5A, TRIM TOOL OVERHEAD

Three sources of bloat, three levers.

01

MCP trims what is sent

Disable servers you do not need today, use a CLI for one-offs. About 5K to 1K per request.

02

Hooks trim what is stored

A local hook filters noisy output before it enters history. About 10K raw to 200 filtered.

03

Skills skip exploration

A SKILL.md describes the structure once instead of reading 5+ files. About 5K to 500.

LEVER 5B, HANDOFF AGENTS

Pin a tier per role, pass a small payload.

01

Planner, powerful

Read-only tools, produces a 5-step plan and design notes. The frontier rate, paid once.

02

Implementer, lightweight

Edit tools, executes one plan step per turn. The bulk of the work, on the cheap tier.

03

Reviewer, versatile

Read-only, diff and tests only. Naive 10 Opus turns about $15, handoff about $2.11, roughly 7x cheaper.

LEVER 5C, SUBAGENTS

Pay for discovery once, not every turn.

WITHOUT SUBAGENT

Re-billed every turn

The parent reloads the same file content each turn. A 20-file refactor over 4 turns ships 86K input, paid four times.

WITH SUBAGENT

Read once, summarized

The subagent reads the files in isolated context and returns a 1K summary. Parent input stays flat, about 8K total, roughly 10 times less.

SUBAGENT VERSUS HANDOFF

Pick by task shape, not preference.

5.1

Subagent (5.1)

Bounded discovery, file or context volume the bottleneck, one answer, parallel fan-out, parent stays. Discovery goes to a subagent.

5.3

Handoff (5.3)

Multi-phase build, reasoning depth the bottleneck, three-plus phases, sequential, the baton passes forward. A build goes to handoff.

PART

VI

Apply, govern, measure.

Context engineering, budgets and controls, observability, and a sequenced adoption playbook.

CONTEXT ENGINEERING

You are now a context engineer.

01

Persistent instructions

copilot-instructions.md rides every session. Keep it tiny, write it yourself, log recurring agent misses, recreate it quarterly.

02

Skills and scoped rules

Path-scoped instructions and skills load only when relevant. Prompt files and chat modes cap the toolset and the blast radius.

03

Custom agents and subagents

Pin model and tools per role to prevent wrong paths. Subagents read files once and return only a summary, keeping the parent lean.

YOUR PROMPT, ALWAYS-ON

Steer from the start, do not shave words.

01

Be precise

Not fix the bug, but issue 45 describes X, fix it. Point at the exact symptom and file.

02

Add stop signals

Once the bug is fixed and tests pass, stop. Keeps the agent from wandering into unrequested work.

03

Front-load known context

If you know the files, docs, or skill, name them. Every discovery loop you save is tokens you do not pay.

RESEARCH, PLAN, IMPLEMENT

Work in phases, with a fresh window between them.

RESEARCH	Loads many files, most irrelevant to implementation. Do it in its own window or a subagent.	Optional
PLAN	A reasoning model produces a precise spec, a detailed to-do that does the thinking up front.	Once
IMPLEMENT	Execute the spec on a cheaper model with only relevant context. With a clear spec, parallel agents become possible.	Per step

DETERMINISTIC CONTROLS

Tests restart the accuracy clock.

01

Tests are deterministic

Pass or fail, no probability. A red test stops a drifting agent and forces a rebuild on a stable base.

02

53 percent is tests

The GitHub Copilot CLI team ships about 500 PRs a week. Their number-one practice is tests, 53 percent of the codebase.

03

Beyond tests

Linters, type checkers, security scanners. Any guardrail you can express as code, the agent will execute.

SCOPED AND REUSABLE CONFIGS

Load guidance only when it is relevant.

01

Scoped instructions

applyTo path globs load only when matching files are attached, keeping the always-on file small.

02

Prompt files

Versioned, manually invoked prompts. Kill the every-dev-has-a-pet-prompt drift.

03

Chat modes

A tight tool allowlist literally cannot enter an Agent loop, so the bill is bounded by design.

SKILLS VERSUS MCP

Offer behavior on demand, do not advertise it every turn.

SKILLS · PROGRESSIVE LOADING

1 to 2 KB until one matches

A skill exposes only its name and description until your prompt matches. Install 20 skills and you add a couple of KB, not the full bodies. An open standard across VS Code, CLI, and the cloud agent.

MCP · ALWAYS ADVERTISED

10 to 15 KB every turn

Every enabled MCP tool ships its schema on every turn, used or not. Scope per workspace, prune to weekly-used, and prefer a CLI like gh for read-heavy work.

PERSISTENT INSTRUCTIONS

Keep it tiny, write it yourself, recreate it quarterly.

KEEP

Rules the model cannot infer

Non-obvious hazards, the required test framework and build command, explicit do-not rules for files and patterns. Roughly 150 tokens, paid once and then cached.

CUT

Anything the code already says

Onboarding essays, architecture as ASCII art, full style guides, duplicated rules. 2026 research: generated context files add 20 to 23 percent tokens for about minus 2 percent correctness.

COMPRESSED INSTRUCTIONS

Same rules, about 64 percent fewer tokens.

BEFORE, ~120 TOKENS

Prose

Always use pnpm, never npm. Tests live in spec/. Do not modify files in generated/. Document exports with jsdoc, and so on.

AFTER, ~50 TOKENS

Terse key-value

pkg: pnpm. tests: spec/. no-edit: generated/. docs: jsdoc on exports. Same meaning, paid on every turn, 64 percent lighter.

MCP HYGIENE

Per-workspace beats global, prune the tool list.

SCOPE PER WORKSPACE

.vscode/mcp.json

Lives in git, reviewable, scoped to the repo that needs it. One bad global server pollutes every project.

PRUNE WHAT IS SENT

Every tool ships its schema

A 40-tool set advertises 10 to 15 KB per turn, used or not. Trim to weekly-used, and prefer the gh CLI for read-heavy work.

AGENTRC

Stop hand-writing the instructions stack. Generate it.

01

Readiness

Scores the repo across 9 pillars and a 5-level maturity model. Use it as a CI gate that fails below level 3.

02

Instructions

Generates copilot-instructions.md, the MCP config, settings, and AGENTS.md from the codebase itself.

03

Eval

Re-runs saved cases to catch instruction rot in CI. Experimental, so pin a version.

ROLES AND ISOLATION

Pin the role, isolate the discovery.

CUSTOM AGENTS

Prevent wrong paths

Pin model and tools per role: Planner, Implementer, Reviewer. The real win is not token savings, it is not handing the agent a tool you do not want it to use.

SUBAGENTS

Pay for discovery once

A subagent reads many files in isolated context and returns only a summary. The parent stays lean across a long session, often about 10 times less re-billed input.

POWER-USER TERRITORY

Advanced tactics, for when you orchestrate many agents.

01

Think in code

Write a script to filter output before the model sees it. Filter an API response to the fields that matter, do not dump the whole payload.

02

CLI over MCP

For read-heavy work, a known CLI like gh is leaner than an MCP server's tool surface. Fewer static tokens, same answer.

03

Chronicle and RTK

Run chronicle to analyze your own session logs for improvement areas, and tools like RTK to trim long shell output to what the agent needs.

THE OTHER CONFIGS

Scoped, conditional, and automatic guidance.

01

Scoped instructions

Conditional, by file path. Offered to the agent like skills. Start static, move to scoped only when the file grows.

02

Prompt files

Manually invoked, reusable prompts. A common starting point to kick off skills or custom agents.

03

GitHub Copilot memory

Small, automatic, always-on guidance learned from your behavior, applied across surfaces. Check it periodically.

GOVERNANCE · THE SHARED POOL

Every license contributes credits to one shared pool.

BEFORE · PER-SEAT

Stranded allowances

A light user's unused requests could not help a heavy user, who hit overage while others sat idle.

AFTER · SHARED POOL

Credits flex to need

All users draw from one pool first. Only spend beyond the pool is additional spend, governed by the budget layers. User-level budgets cap any one person.

POOLING IN PRACTICE

Included usage flexes across the enterprise.

PRU MODEL

Stranded allowances

A light user's unused requests could not help a heavy one, who hit overage while others sat idle. One user could not drain another.

SHARED POOL

Less waste, uneven use

Users draw from one pool by real need, so unused value is not stranded. The trade-off, consumption is uneven, which is why user-level budgets exist.

THE FOUR BUDGET LAYERS

Guardrails, not handcuffs. Any budget at zero stops usage.

04

LAYER · GLOBAL CEILING

Enterprise budget

Caps total additional spend beyond the pool. Alerts at 75, 90, and 100 percent. Cost centers can be excluded so a funded unit keeps working.

03

LAYER · PER BUSINESS UNIT

Cost-center budget

Allocates additional spend to an org or user group. Can map to an Azure subscription per cost center.

02

LAYER · FAIRNESS DEFAULT

Universal user budget

One default cap on total consumption per user. The easy way to stop one person from draining the shared pool.

01

LAYER · THE POWER-USER OVERRIDE

Individual user budget

Overrides the universal default for specific users. Max 10,000 budgets across the enterprise, so use overrides sparingly.

THE DAY-0 ASK

150%

Set the user-level budget to 150 percent, and two more moves.

150 percent is high enough never to block normal work and low enough to bound a runaway loop to about 12 hours. A zero budget blocks access entirely, there is no fallback to a cheaper model.

THREE MOVES, TODAY

`ULB 150% · Auto default · instructions file` → RESULT `about 1.6 to 1.8x throughput, same license`

BUDGET SCENARIOS

Three shapes the four layers take in practice.

01

Manage shared usage

A universal user budget sets the default, individual overrides lift specific power users. Baseline control with per-person flexibility.

02

By business unit

Cost-center budgets per org, with the enterprise budget as the global ceiling and a failsafe for anyone outside a cost center.

03

Power users in a unit

All four layers together: universal default, individual overrides inside a cost center, the unit budget, and the enterprise ceiling, optionally excluding the unit.

CREATING THE RIGHT GUARDRAILS

Four questions before you set any budget.

01

Enterprise budget

How much is the business willing to spend on AI services?

02

Universal user budget

How much should any single engineer be able to spend?

03

Cost-center budget

What is the maximum each business unit or team can spend?

04

Individual override

Who needs an exception to those budgets?

BUDGET NOTIFICATIONS

Alerts before the wall, a hard stop at it.

75%

Admins get a warning

At 75 percent, an email lands. Start monitoring closely.

90%

Decide at 90

Raise the budget, or let it cap. A conscious choice, not a surprise.

100%

Blocked at 100

Users stop until the next cycle or an admin raises the cap, which takes seconds.

CONTENT EXCLUSION, THE ADMIN BASELINE

Block files from context before any tuning.

01

Secrets and env

.env, .env.*, secrets/*, *.pem, *.key. Never let these flow into context.

02

Generated and vendored

dist/, build/, node_modules/, vendor/. Noise the model should never read.

03

Sensitive data

compliance/, customer-data/. Globs per repo, org, or enterprise, propagate in about 30 minutes.

MODEL POLICY AND AUTO

Make the efficient choice the default.

AUTO BY DEFAULT

A free 10 percent win

Auto picks an appropriate model and earns a roughly 10 percent paid-user discount. Make it the org default.

RESTRICT PREMIUM

Approve in stages

Cap premium models for routine work in Organization Settings, GitHub Copilot, Policies. Approve by workflow, team, and measurable need, not the honor system.

OBSERVABILITY

You cannot optimize what you cannot see.

04
LAYER · PER TURN
OTel plus model badge
VS Code 1.119 emits OpenTelemetry spans with a token and cache breakdown per turn, plus the resolved model and multiplier inline. Free, local, today.

03
LAYER · PER USER
/usage and /context
In-session and end-of-session cost signals before the invoice. Run /usage at the end of your next three sessions to learn your baseline.

02
LAYER · PER ORG
GitHub Copilot Metrics API
A published dataset, now including CLI activity, exported to CSV for trend analysis and monthly budget re-tiering.

01
LAYER · DASHBOARDS
Viewer and Grafana
The open-source Metrics Viewer and Microsoft's Grafana dashboards turn token waste into a shareable chart. Track cache hit rate above 60 percent.

VS CODE 1.119 TELEMETRY

Three hooks that turn cost into data.

01

OpenTelemetry tracing

GenAI spans with token and cache breakdown per turn, exported to any OTLP backend.

02

Model and multiplier badge

Shows the model Auto resolved and its multiplier inline. On by default.

03

Background todo agent

Offloads to-do bookkeeping to a lightweight model so the main model does not pay for it. Off by default, flip it on.

MEASURE THE DELTA YOURSELF

Run the same task two ways and watch the bill.

FLOW A · NAIVE

Agent, unscoped

Open Agent mode, point it at hash-codebase, iterate 8 to 12 turns in one chat, accept whatever it produced. The control case.

FLOW B · ENGINEERED

Plan, then scope

Plan mode drafts a spec, you review it, hand it to Agent with tight hash-file scope, repo instructions guarantee conventions. Measure OTel tokens, turn count, time-to-mergeable.

NUMBERS TO WATCH

The steady-state signals that keep the program honest.

01

Cache hit above 60 percent

Confirms the instruction prefix is stable. A falling rate means someone is editing team files mid-day and losing the discount.

02

Overage 10 to 15 percent

Down from 30 percent plus in uncontrolled baselines. Review monthly averages, not daily spikes.

03

Weekly top-10 review

Look at the ten heaviest sessions each week. They reveal a fixable pattern: an always-on MCP, a never-cleared session, a premium model pinned for triage.

THE 30/60/90-DAY PLAYBOOK

Do them in order. Each phase makes the next cheaper.

STABILIZE	Instructions on top repos, content exclusion at the org level, Auto as the default, OTel wired for high-volume teams.	First 30 days
STANDARDIZE	Generate the instructions stack with AgentRC, scope MCP per repo, convert prompts to skills, stand up the Metrics export and the 75 percent alert.	Days 31 to 60
SCALE AND GOVERN	File coding-agent-ready issues, add an AgentRC readiness gate in CI, re-tier budgets quarterly from real telemetry.	Days 61 to 90

THE 30-MINUTE MONDAY CHECKLIST

Six items, each under five minutes, on a real repo.

01

Add copilot-instructions.md

Five lines, conventions only, compressed style.

02

Add one scoped instruction

Your highest-traffic directory, for example src/api/.

03

Move MCP to .vscode/mcp.json

Per-workspace beats global, prune to weekly-used.

04

Use a skill for a recurring task

PR review, triage, regression check.

05

Build one GitHub Copilot Space

Your number-one cross-cutting question, three sources max.

06

File one coding-agent issue

Bounded scope, file pointers, acceptance criteria.

THE THREE-PHASE ROADMAP

Pre-enablement, optimization, governance.

PRE-ENABLEMENT	About two hours. Set ULB 150 percent, configure the overage cap with an 80 percent alert, define cost centers, enable Auto org-wide, commit a starter instructions file.	Day 0
OPTIMIZATION	Train on context discipline and hash-mentions, audit MCP per team, set effort to medium, roll out path-specific instructions, start a weekly top-10 outlier review.	Weeks 1-4
GOVERNANCE	Export the usage CSV, review monthly averages, adjust ULB for justified power users, check cache hit rates above 60 percent, refine instructions and toolsets.	Monthly

THE SIX-MONTH UBB TRANSITION

Diagnose, rescue, anchor to the decision points.

MONTH 1 · JULY	Are the budgets and policies still appropriate? Confirm or adjust early, while the cushion is still active.	Confirm
MONTH 3 · SEPTEMBER	What is the realistic budget at the standard allocation? Re-baseline and communicate. The most important checkpoint.	Re-baseline
MONTH 6 · DECEMBER	Is the tier still optimal? Are there renegotiation levers? Renew, change tier, or renegotiate.	Decide

SIX MATURITY DIMENSIONS, WEIGHTED BY COST LEVERAGE

Diagnose before you prescribe. Model governance leads.

25

Model governance

Weight 25. The single biggest cost lever, the right model per task.

20

Platform foundation

Weight 20. The root of runaway cost when absent.

15

Context discipline

Weight 15. Attacks the tokens factor directly.

15

Agent scope

Weight 15. Controls uncontrolled spend.

15

Repository primitives

Weight 15. The start of structural discipline.

10

Budget control

Weight 10. The guardrail against horror bills.

TWO HORIZONS

Rescue fits the window, transformation realizes the value.

RESCUE, JUN TO DEC

Four fronts

Model governance, budgets as guardrails, context curation, repository primitives. All act on the model-times-tokens equation.

TRANSFORMATION

Platform by design

A sedimented platform, a context and orchestration layer, a control center in Microsoft Foundry. Efficiency becomes a property of the infrastructure, not of individual discipline.

THE ADVISORY MODEL

Clear roles, fortnightly cadence.

01

GBB is the brain

Strategy, diagnosis, governance, architecture, the what and the why.

02

Partners are the arms

They integrate and validate in the customer's environment, the how.

03

Customer owns the result

Data, participation, implementation. A checkpoint every 15 days proves the commitment is alive.

THE COMMERCIAL LAYER, PRE-PURCHASE PLANS

Predictability, sized by contract band.

THE THREE P3 PLANS

GitHub, AI Credit, Agent Factory

Pre-paid, one-year, via Azure subscriptions. Shared tiers: 20k = US$ 19k (5%), 100k = US$ 90k (10%), 500k = US$ 425k (15%).

FIT BY ACD BAND

Confirm before proposing

Below 15 percent ACD the P3 wins, 15 to 25 only AI Credit P3 with Deal Desk, 25 percent plus the ACD wins. Validate against the P3 FAQ.

ANTI-PATTERNS TO AVOID

Naming the traps protects credibility.

01

Bill equals incompetence

Wrong. The bill is a mirror of platform maturity, not a verdict. Heavy workloads can be legitimate.

02

Budget against promo spend

The number changes in September. Always budget against the real allocation.

03

Promise the extension as fact

It is conditional. And never measure individual productivity, measure organizational delivery.

LATEST NEWS AND ROADMAP

More of this becomes automatic and policy-enforceable.

01

Auto and Hydra

Auto routes to the cheapest capable model at a paid-user discount. Hydra, a small task-intent classifier, picks the fit at no extra cost and switches only at cache boundaries.

02

Coding agent and Spaces

The coding agent is GA, Spaces replaced Knowledge Bases, agent skills are an open standard, and Grafana dashboards ship for GitHub Copilot, Claude Code, and Codex.

03

What is coming

Token-aware routing, tiers of Auto (Eco, Fast, Balanced, Max), and admin policies to enforce Auto. The model catalog keeps turning over, so treat rates as snapshots.

HOW AUTO WORKS

Two systems pick the model for you.

HYDRA

Task-intent classifier

A small ModernBERT classifier scores prompt complexity across reasoning, debugging, code-gen, and tool needs, and picks the cheapest best fit. It adds no extra cost.

DYNAMIC SELECTION

Real-time routing

Accounts for capacity, latency, and errors. Auto switches models only at cache boundaries, the start of a conversation and after compaction, so it never blows your cache.

FOUR LEVERS OF COST CONFIDENCE

No single feature, a roadmap that works together.

01

Route tasks smarter

Task intent matches work to a model path, so users need not weigh trade-offs.

02

Govern by team

Enterprise Teams and cost centers assign budgets where usage happens.

03

Message the roadmap

Be confident but cautious on timing and future routing.

04

The outcome

More usage, fewer surprises, better controls.

RECENT MILESTONES

Where the platform has moved, so you can ground the timeline.

01

MAR 2025

Semantic index GA

Instant semantic code-search indexing went generally available, making hash-codebase fast and accurate on GitHub-hosted repos.

02

SEP 2025

Coding agent GA, Spaces

The async coding agent reached general availability, and GitHub Copilot Spaces replaced Knowledge Bases.

03

JAN 2026

CLI enhanced

GitHub Copilot CLI gained richer agents and context management, and folded CLI activity into the usage Metrics API.

04

MAY 2026

VS Code 1.119

OpenTelemetry tracing, the model and multiplier badge, and the background todo agent landed, with UBB UI pre-staged.

THE FORWARD ROADMAP

More of what you optimize by hand becomes automatic.

JUNE	Auto rolls out to more clients and becomes the only option for Free and EDU, with UX to explain which model was chosen and cache optimizations.	Now
JULY AND BEYOND	Token-aware routing and tiers of Auto: Eco, Fast, Balanced, Max. A single dial for your cost-versus-capability posture.	Next
DIRECTION	Admin policies to enforce Auto and pin its version. The efficient choice becomes the enforceable default. Treat model rates as snapshots.	Ahead

ANNEX

VII

Caching.

Two caches people conflate. One you influence, the other you host. And whether Foundry plus Redis sits behind VS Code and the CLI.

TWO CACHES, ONE WORD

Prompt cache reuses the same input. Semantic cache reuses a similar answer.

CACHE A · GITHUB COPILOT PROMPT CACHE

You influence it

The provider replays a stable prefix across turns, the cached tokens on your bill, about 90 percent off. VS Code reached over 93 percent cache reuse per request. There is no knob, you keep the prefix stable.

CACHE B · SEMANTIC CACHE

You host it

Returns a stored answer when a new prompt is semantically similar to a prior one. You build it for your own apps and agents on Azure, it does not lower your GitHub Copilot subscription bill.

ADMINISTERING CACHE A

In VS Code and the CLI, you protect the cache, you do not configure it.

01

Keep it warm

Hold instruction files and the MCP set stable during a session, append new rules at the bottom, and let Auto switch models at cache boundaries.

02

What breaks it

Editing instructions mid-day, switching models by hand, or changing the tool set rewrites the prefix and you re-pay it at full input price.

03

Measure it

VS Code 1.119 OTel shows cache read and creation per turn, /context shows the composition. Target a cache hit rate above 60 percent.

CACHE B, THE SEMANTIC CACHE YOU HOST

For your own agents on Azure, not for GitHub Copilot.

01

Azure Managed Redis

With the RediSearch module, the external cache and vector store. Enable RediSearch at creation.

02

API Management gateway

The GenAI gateway in front of your models, also available built-in from Foundry.

03

Lookup and store policies

llm-semantic-cache-lookup and store compare prompts by vector proximity, returning similar answers.

DECISION GUIDE

Which cache, when.

01

In VS Code or the CLI

You are on Cache A, the prompt cache. Protect it with prefix discipline. Nothing to deploy.

02

Building your own agent

Use Cache B, the semantic cache, on Foundry plus Redis. A real, governable cost lever.

03

Both at once

Common in mature platforms. They coexist and never overlap, different layers entirely.

FOUNDRY PLUS AZURE MANAGED REDIS

Right layer for your agents, not for GitHub Copilot in the IDE.

GITHUB COPILOT IN VS CODE AND CLI

No

GitHub Copilot is a managed product. You cannot insert your own Redis or Foundry gateway into its cache or billing path. The only lever is prefix discipline.

YOUR OWN AGENTS ON FOUNDRY

Yes

API Management as a GenAI gateway with the semantic-cache policies plus Azure Managed Redis and RediSearch cuts tokens on the models you call directly. The transformation-horizon platform.

TAKEAWAYS

Three pillars, one mental model.

01

Cost is context

On every surface, what you ship into the chat is what you pay for. Prune, scope, restart.

02

Specific beats broad

Hash-file over hash-codebase, tight issue over vague, focused Space over kitchen-sink.

03

Persistence over prompts

Instructions, chat modes, and Spaces in git outlast any one prompt.

GLOSSARY, PART 1

The terms that run through the deck.

AI Credit

The currency of usage-based billing. 1 credit = US$ 0.01, metered per token.

Token

A word fragment, about three quarters of a word. Input, output, or cached.

Re-send effect

The harness re-sends the whole conversation every turn, so context compounds.

Context window

What the model can hold. Distinct from what you put in it, which is what you pay.

Prompt cache

Provider replay of a stable prefix, about 90 percent off. The cached tokens on your bill.

Context rot

Quality decays as input grows, often before the window is full.

GLOSSARY, PART 2

The configs and controls.

Auto and Hydra

Auto routes to the cheapest capable model, with a paid-user discount. Hydra is its task-intent classifier.

Coding agent

Issue to branch to PR in a sandbox. GA since September 2025.

Skill

A portable, on-demand capability. Loads only when its description matches.

MCP

External tools for agents. Each advertises its schema every turn.

ULB

User-level budget, a cap on total consumption. Recommended at 150 percent.

Content exclusion

Admin control that blocks files from context across all surfaces.

REFERENCES

Validate the figures against primary sources.

01

GitHub

The usage-based billing blog post, the models and pricing rate card, budgets and Auto docs, the coding agent and Spaces changelogs.

02

Microsoft Learn

VS Code 1.119 notes, semantic caching and AI gateway in API Management, Azure Managed Redis, Grafana for AI agents.

03

Research

Liu et al. Lost in the Middle, Chroma Context Rot, the 2025 DORA report, the FinOps 2026 survey.

CLOSE

Instead of counting tokens, make every token count.

Pioneering software development with AI and Agentic DevOps.

Contact

Paula Silva
Software Global Black Belt
paulasilva@microsoft.com

Next step

Run /usage at the end of your next three sessions to learn your baseline, set the user-level budget to 150 percent, enable Auto as the default, and commit a small instructions file on your most active repo.