TOKEN MANAGEMENT AND AGENT DESIGN · PLAYBOOK

Token-based billing rewards disciplined engineering.

On June 1, 2026 GitHub Copilot retired PRUs and moved to GitHub AI Credits. Teams that treat tokens as a finite resource, apply context engineering, and instrument FinOps pay 25 to 45 percent less than teams that use Copilot as a black box. The difference is not talent. It is practice.

AUTHOR

Paula Silva

ROLE

Software Global Black Belt

CONTACT

paulasilva@microsoft.com

READING TIME

~ 45 minutes, end to end

5

PLAYBOOK PARTS

6

CHAPTERS

8

APPLIED PATTERNS

25-45%

COST REDUCTION TARGET

CHAPTERS

Chapter 00

Foundations: tokens, consumption, models

How tokens are counted, where they go inside a session, and how the Copilot model picker shapes the bill. Without this, every estimate is a guess.

Foundations →

Chapter 01

Technical disciplines

Context engineering, prompt caching, agent design, and token-efficient strategies. The levers that stack savings before you touch a budget.

Disciplines →

Chapter 02

The June 2026 change

PRUs retired, GitHub AI Credits in. What changed, the three paths for annual plans, and how to migrate a team without runaway overage.

Change →

Chapter 03

Governance and FinOps

Budgets, alerts, and attribution per session, agent, and repo. Azure AI Foundry economics at enterprise scale.

Governance →

Chapter 04

Applied patterns and anti-patterns

Eight applied optimization patterns and the anti-pattern decision tree. 40 to 70 percent reduction in agentic workloads.

Application →

Chapter 05

Appendices: reference material

Optimized prompt principles, the pricing reference at mid-2026 list prices, FinOps report templates, and where the complete code lives.

Reference →

READING TIPS

KEYBOARD SHORTCUTS

Press `/` to search. Use `j` and `k` to move between chapters. Press `t` to toggle theme. Press `g` to go to top.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PART I, FOUNDATIONS

Foundations: tokens, consumption, and the model ladder.

The model never sees text. It operates on sequences of integers, and every integer has a price. How text becomes tokens, where tokens go, and which model multiplies them: the foundation everything else in this playbook builds on.

The token is the unit of work, and of billing

A token is the atomic unit a language model processes: a positive integer that indexes an entry in a fixed vocabulary, typically between 32,000 and 200,000 entries depending on the model. Most modern LLMs use Byte Pair Encoding (BPE), a compression algorithm designed by Philip Gage in 1994 and adapted for NLP by Sennrich et al. in 2016, which empirically discovers sub-word units from the training corpus. Sub-word tokenization solves three problems at once: open vocabulary, since never-before-seen words decompose into known sub-tokens and the model never hits an unknown token, efficient compression, and multilinguality, because byte-level BPE operates on the 256 possible UTF-8 bytes and works in any language, any code, any emoji.

Because each provider discovers its own vocabulary, the same text tokenizes differently across models, but rule-of-thumb equivalences hold. General English runs about 0.75 words per token; Brazilian Portuguese is roughly 25% denser in tokens than English; code consumes more tokens per line than prose; structured JSON is heavier still, and rare Unicode or emojis can cost 2 to 4 tokens each. In practice, a 1,000-line Python file is about 12,000 tokens, a 2,000-word Portuguese markdown document about 3,300 tokens, and a 500-word English chat response about 670 tokens. With these equivalences you can estimate any session before running it; without them, every cost estimate is a guess.

Input, output, and cache: three prices for three kinds of work

Every LLM call has three token categories, each with a distinct cost profile. Input tokens, the system prompt, tool definitions, conversation history, context, and the current message, are processed in parallel by the attention mechanism; the work is memory-bound and this is the cheapest category. Output tokens are generated one at a time in autoregressive mode, and each new token requires a pass over everything that came before, which makes output typically 3 to 5 times more expensive than input. The ratio is not arbitrary; it is the real computational cost of sequential generation.

Cached tokens are the third category. The provider stores the internal state (the KV cache) of a stable prefix and reuses it across requests: cache reads cost 10% of the regular input price, and a 5-minute cache write costs 25% extra over regular input. In conversational and agentic workloads, where system prompt, tools, and attached documents repeat on every call, systematic caching cuts input cost by up to 90% and total cost by 30 to 50% without changing anything else. It is the most underutilized lever among teams that move to token billing without discipline.

One cost hides inside output: reasoning tokens. In models with extended thinking, the internal reasoning generated before the visible answer is billed as output even though the user never sees it. A visible 500-token response may have consumed 5,000 invisible reasoning tokens, costing 10 times more than it appeared, and teams that enable thinking without this awareness get bills 5 to 10 times higher than expected. Enable extended thinking only where the extra quality justifies the cost.

Where tokens go: the anatomy of consumption

Tokens are not consumed uniformly. The Tokenomics study (arXiv 2601.14470) measured token consumption in multi-agent software development and found implementation dominating at 30 to 50% of the total, which is legitimate work, while refinement takes 15 to 30% and is the usual villain: iterations that better planning, explicit stop criteria, or the right model would have avoided. Understanding the task is cheap, 5 to 10% when done once, verification and testing take 15 to 25%, planning

5 to 15%, and inter-agent communication consumes another 5 to 15% wherever an orchestrator hands work between agents.

The most underestimated effect is the agentic multiplier. In a multi-turn session the entire history is re-sent as input on every turn, so cumulative input grows quadratically with the number of turns. Sessions with more than 10 turns start feeling it; past 20 turns, cost is dominated by accumulated input and cache becomes the only practical lever. Four strategies tame the multiplier, in order of impact: prompt caching; structured compaction, which compresses history into a summary and reduces tokens by 22 to 57% with no loss of accuracy; tool result clearing, which can free more than 50K tokens in long sessions; and subagent fan-out, where subagents explore with isolated context and return only summaries, cutting 70 to 90% in exploration workflows.

> DEFINITION
>
> The agentic multiplier is the reason long sessions surprise budgets: input grows quadratically with the number of turns, not linearly. Treat every long-running agent as a compounding cost object. Cache the stable prefix, compact the history, clear stale tool results, and stop when the task is done.

Models and multipliers in the GitHub Copilot model picker

Model choice is the first-order cost lever: applied well, it reduces average cost by 50 to 70% with no perceived quality loss. In the GitHub Copilot model picker, the Claude family alone spans an order of magnitude at mid-2026 list prices: Haiku 4.5 at $1 per million input tokens and $5 per million output, Sonnet 5 at $3 and $15 (with introductory pricing of $2 and $10 until August 2026), Opus 4.8 at $5 and $25, and Claude Fable 5 at $10 and $50. Opus costs roughly 1.7 times Sonnet, and Haiku is 5 times cheaper than Opus. Bundled models are included in paid plans without consuming additional credits.

The working heuristic has five levels. Bundled models cover about 70% of daily work: completions, quick questions, simple refactorings. Cheap models, the Haiku

4.5 class, take 10 to 15%: classification, extraction, boilerplate at scale. The workhorse tier, Sonnet 5, takes another 10 to 15%: non-trivial debugging, multi-file refactorings, deep code review, the recommended default when bundled is not enough. Premium models, Opus 4.8 and Claude Fable 5, should be 1 to 3% of work, reserved for architecture, distributed debugging, and irreversible decisions, and only after the workhorse proved insufficient. Long-context models enter when the problem genuinely needs more than 100K tokens of context.

Since the June 1, 2026 transition, GitHub Copilot bills this consumption in GitHub AI Credits, where one credit corresponds to roughly one dollar of usage at list price. The arithmetic is direct: a call with 8K input and 1K output on Sonnet 5 costs $0.039; a team of 50 developers making 1,000 such calls per day spends about $1,950 a day, $42,900 a month over 22 business days, close to $515K a year. Under flat per-request billing all those calls cost the same; under tokens they reflect real work, which is exactly why the foundations in this chapter are now a budget skill, not a curiosity.

References

Primary sources for tokenization, consumption anatomy, and model pricing.

- Neural Machine Translation of Rare Words with Subword Units, Sennrich, Haddow and Birch, ACL 2016, the paper that adapted BPE to NLP and solved the open-vocabulary problem.
- Tokenomics, token consumption distribution in multi-agent SDLC systems, arXiv 2601.14470, the empirical distribution behind the anatomy of consumption.
- GitHub Docs, Available Copilot models, the authoritative list of models in the GitHub Copilot model picker.
- Anthropic, Models and pricing, list prices for the Claude family: input, output, and cache.
- Anthropic Docs, Prompt caching, the mechanics of cache reads and writes behind the 10% read price.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PART II, TECHNICAL DISCIPLINES

Technical disciplines: context, cache, agents, and efficiency.

Four disciplines separate teams that pay full price from teams that pay a fraction for the same work: context engineering, prompt caching, agent design, and token-efficient strategies, applied through the GitHub Copilot lens.

Context engineering: curate with surgery

Context engineering is the discipline of selecting the smallest set of high-signal tokens that maximizes the probability of the desired outcome. The common intuition, that more context always helps, is wrong. The curve relating context volume to response quality is concave with a peak: irrelevant tokens first have a neutral effect, then a negative one, a failure mode called context rot. Teams that master the discipline pay 30 to 50% less for the same work, with equal or better quality.

Four techniques make it operational. Compaction, where the model compresses its own history into a structured summary, delivered a 22.7% average token reduction on SWE-bench Lite with Claude Haiku 4.5, with 0% accuracy loss. Tool result clearing targets the largest consumer in agentic workflows: reading one 2,000-line file can inject 30,000 tokens, so keep only the last 5 to 10 results. Persistent memory moves stable facts out of the window, with 11x compression demonstrated over 4,182 conversations. Subagents with fan-out and central synthesis cut exploration workflows by 70% or more. In Copilot this lands in AGENTS.md and .github/copilot-instructions.md: a human-curated AGENTS.md reduced runtime by 28.64% and tokens by 16.58%, while an LLM-generated one performed 3% worse than having no file.

Prompt caching: the 90% lever

Prompt caching stores the attention state computed over your prompt so that subsequent requests with the same token prefix skip that work. The cache operates by prefix hash, byte by byte: any change in any byte invalidates it from that point on. The rule is architectural: stable content first, dynamic content last. System prompt, tool definitions, and static context such as AGENTS.md content go at the top with cache_control; conversation history and the current message stay at the end, never cached. Minimums apply, 1,024 tokens of cacheable prefix on Sonnet and Opus class models and 2,048 on Haiku 4.5.

The economics are simple. A cache read costs 10% of regular input. A cache write with the default 5-minute TTL costs 25% more, and each request reactivates the TTL while the session stays active. In conversational workloads, with typical hit rates of 70 to 85% after the first turn, input savings of up to 90% are achievable without changing a single line of prompt. GitHub Copilot in VS Code manages this cache automatically; you help it by keeping .github/copilot-instructions.md and AGENTS.md stable, changed in pull requests rather than ad hoc.

Anti-patterns that break the cache

Five recurring mistakes destroy the benefit. A timestamp inside the cached prefix changes the hash on every request, so you pay the 25% write premium every time and never read. cache_control on a block that changes, such as the user message, does the same. Reordering blocks between sessions produces a complete miss. Cosmetic variation, a double space or a swapped word in a prompt assembled inline, silently invalidates everything, so keep system prompts in versioned files. And a prefix below the minimum size is silently ignored.

> DEFINITION
>
> Cache read costs 10% of regular input; a 5-minute cache write costs 25% more. With a stable prefix, up to 90% of input cost disappears without touching a single prompt line. The prerequisite is discipline: stable first, dynamic last, nothing volatile inside the cached prefix.

Agent design: default to workflow

The Anthropic taxonomy separates workflows, where code decides the sequence and the LLM does parts of the work, from agents, where the model directs its own process and tool usage. A workflow has predictable cost; an agent can cost from $0.30 to more than $5 on the same task. About 70% of enterprise cases are solved by workflows at a fraction of the cost, so the default is workflow and agency must be justified. Five composable patterns cover the space: prompt chaining, routing, parallelization, orchestrator-workers, and evaluator-optimizer. Reserve true agents, in Copilot terms agent mode and the Coding Agent, for wide solution spaces where the sequence genuinely varies per problem.

When agency is justified, constrain it. Custom agents defined in .agents/ files should follow single responsibility, be read-only by default, carry explicit tool allowlists, and declare budgets such as max_iterations and max_tokens_per_session. Apply the same treatment to the cloud-side GitHub Coding Agent: cap tokens and iterations per task, plan with a stronger model and execute with a cheaper one, and require human approval on every PR. MCP servers are among the largest sources of token overhead; a lean tool definition of about 80 tokens beats a verbose one of 500. Hooks around tool calls, caching repeated results and truncating large outputs, can reduce agentic workflow cost by 30 to 50%.

Token-efficient strategies and the model picker

Less is more, measurably. Cutting a skill body by 39% and its descriptions by 48% improved functional quality by 2.8%: compact prompts concentrate the signal that attention actually reaches. Progressive disclosure applies the same idea structurally: a small always-loaded core, with detailed checklists and examples fetched on demand. Lean tool design completes the set: short names, one-line descriptions, few parameters, ranged results instead of whole-file dumps, dense JSON instead of prose. Every evaluator-optimizer loop needs explicit stop criteria: a maximum iteration count, a score threshold, a minimum improvement delta, and a token budget.

The last discipline is routing by complexity, and in Copilot it lives in the model picker. Defaulting to a premium model overpays 3 to 5x on average; a classification step

costing about $0.00005 per call reduces average cost by 50 to 70% with no perceived quality loss. Map the tiers to mid-2026 pricing: routine work to Haiku 4.5 at $1 input and $5 output per million tokens, complex work to Sonnet 5 at $3 and $15, expert work to Opus 4.8 at $5 and $25, and only frontier reasoning to Fable 5 at $10 and $50. Every premium choice consumes AI Credits at its own multiplier, so the picker is a budget instrument, not a preference. Encode the policy in .github/prompts/ custom prompts, with an explicit model and applyTo patterns.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PART III, THE CHANGE

The June 2026 change: from premium requests to GitHub AI Credits.

PRUs are retired, usage-based billing is live, and one dollar of credits buys one dollar of work at list price. What changed, what to decide, and how to move a team through it.

What changed on June 1, 2026

On June 1, 2026, GitHub Copilot retired Premium Request Units, the per-request billing unit for premium models, and moved every paid plan to GitHub AI Credits, a usage-based currency in which one dollar of credits corresponds to roughly one dollar of usage at the published API list price of each model. One AI Credit equals US$ 0.01, and each premium model call consumes credits in proportion to the input, output, and cached tokens it actually processes. The structural shift fits in one sentence: under PRUs, cost was per intent; under tokens, cost is per work performed. Mario Rodriguez, GitHub CPO, framed the cause as architectural, not commercial: a quick question and a multi-hour autonomous coding session could cost the user the same, and the implicit subsidy behind long agentic sessions was no longer sustainable.

Plan base prices did not change. Copilot Pro stays at US$ 10 per month and now includes 1,000 AI Credits, Pro+ stays at US$ 39 with 3,900 credits, Business at US$ 19 per user with 1,900 credits per user, and Enterprise at US$ 39 per user with 3,900 credits per user. Inline code completions, Next Edit Suggestions, and Copilot Chat with bundled models such as GPT-4.1 and GPT-5 mini remain included and consume no credits. Two changes deserve special attention. First, the automatic fallback is gone: when credits reach zero and the paid usage policy is disabled, premium models stop working, which turns budget into a continuity control, not just a cost control. Second, Copilot Code Review now bills in two dimensions, AI Credits for the

model and GitHub Actions minutes for the runner. Credits also reset on day 1 of each month at 00:00 UTC and do not carry over: use it or lose it.

> DEFINITION
>
> The primary anchor of this chapter: on June 1, 2026, GitHub Copilot retired PRUs and moved paid plans to GitHub AI Credits, announced on the official GitHub blog as GitHub Copilot is moving to usage-based billing. One AI Credit equals US$ 0.01, so one dollar of credits buys approximately one dollar of model usage at list price. Everything else in this chapter follows from that single fact.

Three paths for annual Pro and Pro+ subscribers

Users who subscribed to Pro or Pro+ on an annual plan before June 1, 2026 got a differentiated transition. They stay on PRUs until the plan expires, but the multipliers increased sharply: Claude Opus 4.7 went from 7.5x to 27x and Claude Sonnet 4.6 from 1.25x to 4.5x, both a 3.6x increase, while GPT-5.3 Codex and Gemini 3.1 Pro went from 1x to 6x. Real capacity falls in the same proportion: whoever ran 100 Opus sessions per month now runs about 28, and 100 Codex sessions become about 17. GitHub offered three explicit paths, each with a deadline that mattered in May.

Path A, keep the annual plan until expiration

Zero action required and the annual price stays as paid, but real premium capacity falls 3.6x to 6x and there is no granular cost visibility, because billing is still in PRUs. This path makes sense for developers who work mostly in bundled models like GPT-4.1 and GPT-5 mini, use premium models only occasionally, or hold an annual plan expiring in less than three months.

Path B, convert to monthly

GitHub offers prorated credits for the remaining time on the annual plan. A Pro+ annual subscriber who paid US$ 390 with four months remaining receives US$ 130 as 13,000 bonus AI Credits, enters June with the monthly Pro+ plan active plus the bonus, 16,900 credits in total, and from July onward runs on the standard 3,900

credits per month. The conversion deadline was May 25. This is the right path for power users who rely on premium models frequently and want token-level visibility right away.

Path C, cancel with a prorated refund

Cancelling before May 20 triggered a refund proportional to unused time, US$ 130 in the same four months remaining example. It recovers value and leaves the user free to choose another provider, at the cost of losing Copilot access unless they re-subscribe monthly. It fits users leaving the Copilot ecosystem entirely or losing access through a company change. The persona mapping is direct: individual devs who live in bundled models take Path A, premium power users take Path B, and anyone moving away takes Path C.

Migrating a team: the 16-week window

For organizations, the transition is not a one-day event but a roughly 16-week process, from the bill preview GitHub made available in early May 2026 to steady-state costs in September. May was discovery and preparation: analyze the bill preview in Billing Overview, identify the top consumers, decide the fate of annual plans before the May 20 and May 25 deadlines, activate the premium request paid usage policy with a cap set 20 to 30 percent above historical consumption, configure alerts at 50, 75, 90, and 100 percent of budget, designate an explicit FinOps owner, and train tech leads before June 1. The paid usage policy is the single most important setting: with it enabled, work continues past the included credits as controlled overage; without it, premium models stop the moment credits hit zero.

June is D-Day plus stabilization: monitor consumption daily in the first week, validate alerts, then apply the quick wins, complexity-based model routing and prompt caching on the heaviest workloads. July is systematic optimization, with weekly showback to tech leads, custom prompts, and enterprise-grade observability. August is maturation, and September delivers the first complete month of realistic billing. The bill preview tells each organization how urgent this is. A projection equal to or below the current PRU bill means discipline already exists. An increase of 30 to 100 percent signals premium models inside long agentic workflows and calls for immediate routing and caching. Above 100 percent usually means Opus in agent

mode without discipline and demands corrective action before it becomes a serious operational risk. Teams that applied this discipline captured a 25 to 45 percent reduction against undisciplined usage. Those who acted early captured savings; those who waited paid overage.

References

Sources for the June 2026 change, GitHub AI Credits, and the migration paths.

- GitHub Copilot is moving to usage-based billing, GitHub Blog, April 2026, the official announcement that retired PRUs and introduced GitHub AI Credits.
- About billing for GitHub Copilot, GitHub Docs, plan prices, included AI Credits, and the paid usage policy.
- GitHub Copilot documentation, GitHub Docs, per-model pricing tables and annual plan multipliers, always check for updated values.
- FinOps Foundation, the framework behind the migration timeline, showback, and budget ownership practices in this chapter.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PART IV

Governance and FinOps: run AI spend like a cloud resource.

Budgets, alerts, and cost attribution by session, agent, and repository for GitHub Copilot, plus the Azure AI Foundry economics that make governance affordable at enterprise scale.

Inform, Optimize, Operate: the FinOps cycle applied to Copilot

Without data, optimization is guessing. The FinOps Foundation defines three iterative phases. Inform is granular visibility: who uses AI, how much, with which model, in which feature. The Microsoft stack here is concrete: applications with Copilot emit telemetry via OpenTelemetry into Application Insights, Azure Cost Management exports FOCUS-compliant data, the open source Microsoft FinOps Hub processes it, and Power BI or Grafana visualize it. Optimize turns that visibility into opportunities with estimable payback: inefficient prompts, poorly chosen models, missing cache, untrained teams. Operate institutionalizes the cycle with recurring FinOps meetings, live KPI dashboards, and policy and budget adjustments based on data.

Maturity is measured in months, not days. Month 1 establishes the baseline: how much do we consume today, and where. Months 2 and 3 deliver the first savings from quick wins. Months 4 to 6 institutionalize continuous improvement. From month 6 onward, showback evolves to chargeback and model decisions become data-driven decisions.

Five metrics to instrument from day one

Cost per 1K tokens tells you whether the average price is fair for your model mix; the enterprise target is a decreasing trend, driven by more cache, since cache reads cost 10% of the input price, by routing toward bundled models, and by context engineering. Cost per feature tells you whether an AI capability returns its ROI. Cost per user exposes outliers: the top 10% of users typically consumes 40 to 50% of

total spend, and the top 1% alone can take 10 to 15%. Cache hit rate should exceed 60% in conversational workloads. Output/input ratio is healthy between 0.1 and 0.3; above 0.5 is a red flag.

Budgets, alerts, and attribution by session, agent, and repository

There are four levels of control, from softest to most invasive. Budget alerts notify at 50, 75, and 90% of the monthly budget; they are a signal to react, not a stop. Budget caps interrupt premium models at 100% while bundled models keep working: with the GitHub Copilot premium request policy set to a monthly cap, for example $5,000, developers lose Sonnet, Opus, and Gemini Pro until the next month but keep GPT-4.1 and GPT-5 mini. Hard limits stop everything and should be rare: fixed POC budgets or client contracts where overage cannot happen at all. Anomaly detection is the proactive level: algorithms compare current consumption with a rolling 30 day history and flag any day above 3 sigma. A typical drill-down goes from an anomalous day to the top users, then to one user's sessions, and finds a Coding Agent that ran 50 times in a loop because of a bug.

None of this works unless every AI Credit is attributable. Each request should carry the model, the mode (ask, edit, agent, or plan), tokens by type including cached reads, cost in USD, AI Credits consumed, and identifiers for user, team, feature, session, and organization or repository. That attribution feeds showback, which shows each team its cost without charging, and later chargeback, which actually debits the cost center. Sequence the rollout: showback for tech leads in months 1 to 3, for engineering managers in months 4 to 6, public between teams after month 6. Move to chargeback only with 6 or more months of reliable showback and consistent tagging, then phase it in: 25, 50, 75, and 100% of consumption across four months.

DEFINITION

Behavior changes when the number is seen. Showback creates visibility with zero financial friction; chargeback turns model choice into an economic decision. The readiness signals are explicit: fewer than 3 months of showback, more than 10% of calls untagged, or teams still resisting showback all mean chargeback is premature, whatever the pressure from finance.

Azure AI Foundry economics at enterprise scale

GitHub Copilot solves coding in the IDE; Azure AI Foundry solves production workloads and regulatory governance. It is a complement, not an alternative, and the choice is per workload, not per organization. Foundry brings a catalog of 1,800+ models from Anthropic, OpenAI, Meta, Mistral, and others, Microsoft Entra ID authentication, built-in RBAC, audit logs in Azure Monitor, network isolation with private endpoints, and certifications including HIPAA, SOC 2 Type II, ISO 27001, FedRAMP High, GDPR, and LGPD. IDE coding stays on Copilot, regulated workloads and RAG over sensitive data go to Foundry, and quick POCs can use the direct API.

PTU versus PAYG: the billing decision with the largest impact

PAYG charges per token, with no minimum and no commitment, and suits variable or spiky load. PTU, Provisioned Throughput Units, reserves inference capacity: you pay for capacity, not per token, with guaranteed throughput and consistent latency. The general rule: below $5K per month in one model, PAYG always; between $5K and $30K, it depends on latency requirements; above $30K, seriously evaluate PTU; above $100K, PTU almost always wins. Scale changes everything: 50M tokens per month on GPT-5 costs about $500 per month on PAYG, and PTU is not even a conversation. At 5B tokens per month, the PAYG bill reaches $50,000 per month, and reserved PTUs with a 30% annual discount come out 75% cheaper.

Governance has a price, and it is small

Azure AI Content Safety filters are not free, but they are cheap relative to the risk: text moderation at $0.40 per 1,000 calls, prompt shields at $0.30, protected

materials at $0.60, and groundedness detection at $0.75. For 100K calls per month with all filters enabled, that is roughly $200 per month, a rounding error next to one compliance incident. The line item to watch is diagnostic logging of full prompts and completions, which can be expensive; set a retention policy. Finally, the FOCUS export lands GitHub Copilot billing and Foundry consumption in the same Azure Cost Management data, so the CFO reads one Power BI dashboard, not two.

References

Sources for the FinOps cycle, the control levels, and the Foundry economics.

- FinOps Framework, FinOps Foundation, the Inform, Optimize, Operate phases.
- FOCUS Specification, FinOps Foundation, the open cost and usage schema behind the unified export.
- Microsoft FinOps Toolkit, Microsoft, the open source FinOps Hub with Power BI templates and queries.
- Azure Cost Management Documentation, Microsoft, budgets, alerts, and FOCUS exports.
- Azure AI Foundry Documentation, Microsoft, model catalog, content safety, and enterprise governance.
- Provisioned Throughput Concepts, Microsoft Learn, PTU sizing and pricing model.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PART V, PRACTICAL APPLICATION

Applied patterns and anti-patterns.

Eight patterns you can implement tomorrow, the anti-patterns that silently undo the savings, and the decision trees that turn token discipline into reflex.

Eight patterns, one goal

Theory without implementation does not save tokens. This part of the playbook lands the previous chapters as eight concrete patterns for GitHub Copilot and the agentic workloads around it. Each pattern solves a specific cost problem, each has measurable impact, and they compose: routing decides which model answers, caching decides how much of the input gets billed again, agent design decides how much context each step carries, and tool design decides the fixed overhead of every turn.

Route by complexity, plan with the premium model

Pattern 1 is dynamic routing. Teams set the premium model as the Copilot default and pay premium prices for questions a bundled or cheap model would answer. The solution is a two-stage classifier: free static heuristics by keyword first, then a cheap model as dynamic fallback when the heuristics cannot decide. Impact: 40-70% reduction in average cost without perceived quality loss. Pattern 4 is plan and execute: plan the large task with the premium model, then execute with the workhorse. In the source scenario this comes out 64% cheaper than running everything on the premium model, with premium plan quality.

Cache in layers, exact and semantic

Pattern 2 is the hierarchical cache in three layers: an exact cache keyed by hash, a semantic cache that matches by embedding similarity, and the provider prompt cache underneath. Exact caches miss on cosmetic variation; the semantic layer catches questions that mean the same thing. Combined hit rate reaches 60-90%,

cutting input cost by 50-80%. Pattern 5 is the standalone semantic cache for when the full stack is not needed: on FAQs and recurring questions it adds 20-40% hit rate on top of an exact cache.

Isolate context, specify before you generate

Pattern 3 is subagent fan-out. A lead agent exploring a large codebase accumulates 800K+ tokens of context. Instead, it dispatches subagents with isolated context and read-only tools, each returning a structured summary of at most 2K tokens, and the lead synthesizes only the summaries. Impact: 70-90% reduction in exploration workflows. Pattern 8 is the custom agent with Spec-Kit: a Copilot custom agent in .agents/ that reads a CONSTITUTION, a SPECIFICATION in EARS notation, and an IMPLEMENTATION_PLAN before writing any code. Structure replaces rework: 40-60% fewer refinement iterations.

Compose tools, keep MCP lean

Pattern 6 is tool composition. Ten granular tools at roughly 200 tokens of definition each impose 2,000 tokens of overhead on every single turn; three composed tools with sensible defaults cover the same space and cut system prompt overhead by 60-70%. Pattern 7 extends the idea to the MCP servers Copilot connects to: short tool names, descriptions of 5-10 words, 2-4 essential parameters, paginated results, and lazy-loaded resources. A lean server fits its entire definition in about 250 tokens, against 1000+ in verbose servers.

> DEFINITION
>
> The number to remember: combined in enterprise agentic workloads, these eight patterns deliver a 40-70% reduction in total cost. Not from one silver bullet, but from routing, caching, context isolation, and tool discipline compounding on every request Copilot makes.

The anti-patterns that undo the savings

The source catalog lists ten recurring anti-patterns from enterprise pilots. The agent graveyard: agents created as proofs of concept, never retired, still consuming

tokens. Everything is an agent: deterministic flows implemented as agents for future flexibility, when workflow wins in 70% of cases and converting back typically saves 50-80%. Premium as default: the bias that premium is always better. The monolithic system prompt: 30K tokens of every company convention, which also breaks caching; the fix is progressive disclosure plus a curated AGENTS.md under 500 lines. And tool result blowup: sessions accumulating 600K tokens of raw file reads because tools return entire files.

The second half: absence of stop criteria, instead of combining max iterations, score target, improvement delta, and token budget. The cache that never caches, hit rate under 10% because a timestamp or a reordered block breaks the stable prefix. Long context misuse, confusing what fits with what is worth it, when a 1M token call costs $10. Multi-agent without coordination, where inter-agent communication consumes 30% of the tokens. And no instrumentation, so the real cost is a surprise at the end of the month. GitHub Copilot adds its own variants: Agent mode for simple questions that Ask mode with a bundled model answers at 5-10x less, no committed workspace settings so cost varies 3-5x across teams, and useful prompts trapped in local snippets instead of versioned in .github/prompts/.

The mode matrix in VS Code

The practical antidote is a mode matrix every developer internalizes: Ask with a bundled model for questions, Edit for local changes like docstrings and renames, Agent with the workhorse for complete features and multi-file migrations, Plan with the premium model for architecture and irreversible tradeoffs. Plan makes few calls, so it costs little even on the premium model; execution makes many, which is why it deserves the workhorse.

Three decision frameworks and the rollout

Framework 1, which model: bundled models for autocomplete and short questions; the long-context model only when context genuinely exceeds 100K tokens; the cheap tier for classification and extraction; the workhorse as the default for refactoring and debugging; the premium model only when the workhorse failed or the problem is genuinely complex. The rule of thumb: the workhorse covers 70% of non-bundled cases, premium is for the 5-10% genuinely difficult. Framework 2,

workflow or agent: if the sequence of steps is known and stable, use a workflow; add agency only when decomposition truly varies by input.

Framework 3 is optimization order by ROI: complexity routing first, then prompt caching, subagent fan-out, tool result clearing, lean tool design, system prompt audit, stop criteria, and structured compaction last. After 8 weeks of that sequencing, the source reports a typical 40-70% reduction over the pre-optimization baseline. The rollout runs in four phases: discovery to measure the current state; quick wins such as the workhorse as workspace default and spending caps with alerts, worth an expected 25-40% reduction; technical disciplines for an additional 10-20%; then continuous FinOps operation with dashboards and anomaly detection.

References

Sources for the eight patterns, the anti-pattern catalog, and the decision frameworks.

- Building effective agents, Anthropic, the workflow versus agent taxonomy behind Framework 2.
- Effective context engineering, Anthropic, the context isolation rationale behind subagent fan-out.
- Spec-Kit, Spec-Driven Development, GitHub, the CONSTITUTION, SPECIFICATION, and IMPLEMENTATION_PLAN structure of Pattern 8.
- Model Context Protocol Specification, the protocol behind the lean MCP server template of Pattern 7.
- GitHub Docs, Best practices for Copilot at scale, the operational guidance behind the rollout phases.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 APPENDICES

Appendices, the reference material.

Four appendices back this playbook: optimized prompts, a pricing reference card, FinOps report templates, and the complete code. This chapter condenses each one into what you need at the moment of use.

Appendix A, optimized prompts

Appendix A collects the prompt templates tested in the 2025 and 2026 Latin America pilots. The design principles are compact and repeatable: state the role in one line, enumerate severities or steps instead of describing them, fix the output format with a JSON schema or an instruction such as just the docstring, and delegate project conventions to AGENTS.md instead of repeating them in every prompt. A code reviewer written this way fits in roughly 50 tokens and replaces the verbose 500 plus token versions that teams write by instinct, with no loss of review quality.

The same appendix ships lean tool definitions for read, search and run_tests, plus custom prompts for .github/prompts/ whose frontmatter pins the model in the GitHub Copilot model picker per task: deep code review on a workhorse model, architecture design on a premium model with extended thinking, code explanation on a bundled model that consumes no AI Credits. Completing the set are preToolUse and postToolUse hook templates, which add cache checks, a budget gate, path validation, secret redaction and result truncation, and an AGENTS.md template that tells agents where things live and what not to touch.

Appendix B, pricing reference card

The reference card uses the mid 2026 Anthropic list prices per million tokens, the same ones behind the Copilot model picker options. Claude Fable 5 costs $10 input and $50 output. Opus 4.5 through 4.8 cost $5 input and $25 output. Sonnet 5 costs $3 and $15, with introductory pricing of $2 and $10 until August 2026. Sonnet 4.6 costs $3 and $15, and Haiku 4.5 costs $1 and $5. Two cache rules complete the card:

a cached read costs 10% of the input price, and a 5 minute cache write costs the input price plus 25%. For Sonnet 4.6 that means $0.30 per million cached tokens read and $3.75 per million written.

On the consumption side, the appendix converter treats one AI Credit as roughly one cent of list price. Plan capacity follows: Copilot Pro at $10 per month includes 1,000 AI Credits, Pro+ at $39 includes 3,900, Business includes 1,900 per user and Enterprise 3,900 per user. Annual plans still measured in PRUs convert with multipliers between 3.6x and 6x, so always check the exact table on docs.github.com before committing a budget.

> DEFINITION
>
> If a copy of the pricing appendix still lists Opus at $15 input and $75 output, it predates the 2026 repricing: the current Opus 4.5 to 4.8 list is $5 and $25. Prices move; the cache rules of 10% for reads and plus 25% for 5 minute writes have stayed stable. Verify at docs.github.com and anthropic.com/pricing before publishing any estimate.

Appendix C, FinOps report templates

Appendix C provides two report cadences. The weekly operational report, for tech leads every Monday, tracks total spend against last week, cost per thousand tokens against a target below $0.005, cache hit rate against a target above 60%, the top 10 consumers with average cost per session, detected anomalies with their investigation notes, and the budget alerts that fired. The monthly executive report, for the CFO and the VP of Engineering, adds cost per developer, budget utilization, distribution by cost center and by model, the optimization wins of the month, areas of concern such as user outliers, and the ROI of the playbook itself.

The templates come with the tooling to feed them: three Power BI dashboards (Executive Overview, Tech Lead Operational and FinOps Detail), six KQL queries for Application Insights covering cost per team, runaway sessions above $5, cache hit rate per feature, daily anomalies beyond three standard deviations, model distribution per user and output to input ratio red flags, and a ready to import

Grafana dashboard JSON with total cost, consumption trend, a cache hit gauge and a top users table.

Appendix D, where the complete code lives

Appendix D centralizes the implementation referenced across the playbook. The production model router classifies each request into five tiers, bundled, cheap, workhorse, premium and long context, trying a free static keyword pass first and falling back to a dynamic classifier on Haiku 4.5 that costs a fraction of a cent per call; it carries a fallback chain and emits every routing decision to telemetry. The hierarchical cache stacks three layers: exact match in Redis, semantic match with a 0.93 similarity threshold, and the provider cache itself.

Two more assets close the appendix: an OpenTelemetry setup with token, cost and call counters dimensioned by model, user, feature and team, feeding the KQL queries and dashboards of Appendix C, and a lean MCP server template whose three tool definitions total about 250 tokens. A custom agent template shows the guardrails in file form: pinned model, explicit tool list, context_files, max_iterations of 25, a 500,000 token session budget and pre and post tool hooks. All four appendices ship with this playbook as markdown files, from A_appendix_optimized_prompts.md to D_appendix_complete_code.md, next to the chapters; copy them into your repo and adapt them to your stack.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps