OPERATING MODEL · TOKENOMICS · GOVERNANCE

From the validation gate to Azure AI Foundry.

How to turn "we can use many models and primitives" into "we use them validated, with a budget and routing." A complete, didactic walk from Monday with zero infra to Redis and Foundry.

AUTHOR
Paula Silva

ROLE
Software Global Black Belt

DURATION
60 to 90 minutes

DATE
2026-06-05

AGENDA

Seven parts, one operating model.

PART I	Why this exists, multi-model is the start, not the end
PART II	The validation gate, the five steps every primitive crosses
PART III	Token budget, the hot, warm, and cold context layers
PART IV	Governance and cache, internal catalog plus Redis
PART V	Three example agents, Q&A, security review, scheduled audit
PART VI	Agent surfaces, agent mode, coding agent, and workflows
PART VII	The maturity ladder, from Monday morning to Foundry

PART

I

Why this exists.

Multi-model is the starting point, not the end. The missing piece is the operating model.

THE PROBLEM

Choice without governance becomes a billing bug.

WHAT YOU GET OUT OF THE BOX

Freedom to choose

In GitHub Copilot (IDE, chat, coding agent, code review) and the GitHub Copilot CLI you pick from many models. Via Agent HQ, third-party models from Anthropic, OpenAI, Google, Cognition, and xAI join inside the paid plan. Each agent pins its model in the frontmatter, or Auto decides.

WHAT GOES WRONG

Cost with no brakes

Agents reach for the most expensive model on trivial tasks. There is no token ceiling, so the long tail blows the budget. Nothing is measured before production, so quality is a bet. Freedom without governance is the billing bug.

THE CENTRAL PRINCIPLE

Every primitive is a versioned artifact, with a budget and a gate.

Before any primitive reaches production, it crosses a gate. The public awesome-copilot repo is inspiration, not production. What ships is versioned, measured, and validated.

Predictable

cost, a known ceiling per task

Auditable

quality, scored before publishing

Traceable

security, declared model policy

WHAT IS A PRIMITIVE

Five kinds of reusable AI unit, one gate.

01

Agent and prompt

A profile with a fixed model, instructions, and scope. Or a reusable template. The most common primitives.

02

Skill and hook

An encapsulated capability, or an automated trigger that fires on an event. Reused across repos and teams.

03

MCP server

A connector that brings in external systems. The most expensive to call, so it lives in the cold layer.

PART

II

The validation gate.

Five mandatory steps, in order. An artifact is only validated when it crosses all of them.

THE FIVE-STEP GATE

Nothing reaches developers directly.

validation-gate.flow

1 golden tasks representative tasks for this agent

2 measure input, output, cached tokens, p50 and p95

3 ceiling + policy token ceiling per task and model tier

4 judge LLM-as-judge, paired, order swapped

5 publish versioned into the internal catalog

STEP 1, GOLDEN TASKS

Real tasks, with known answers.

01

Representative

Cover the most frequent flows, not exotic edge cases. The work the agent will actually do in production.

02

Stable

The expected output does not change on every run. You can compare against it reliably.

03

Measurable

The response can be scored objectively or by a judge. A real question with a known correct answer.

STEP 2, MEASURE CONSUMPTION

Output is what bites.

5x

Output costs about five times the input. Generating text is more expensive than reading it.

USE P95 **p95**

Not just the average. The average hides the long tail that blows the budget in production.

STEP 3, CEILING AND MODEL POLICY

Two limits per artifact.

TOKEN CEILING

A limit per task

The threshold above which the agent cuts or refuses, based on the observed p95 plus a margin. It works like a stopwatch: past the limit, the agent stops and delivers what it has.

MODEL POLICY

Which tier is allowed

A simple Q&A does not need the most expensive model. A security review might. Centralize access, decentralize choice: the dev picks among the validated models, not the whole world.

STEP 4, QUALITY AND SECURITY

An LLM scores the answers.

01 A separate, capable model scores the agent's responses on the golden tasks.

02 Use paired comparisons and swap the order between rounds to remove position bias.

03 On the GitHub side this pairs with gh aw compile --validate for agentic workflows.

04 On the model side, with Foundry Evaluations, eval of skill and of workflow.

05 Note: Foundry Evaluations does not integrate directly with the model router. The quality, cost, and latency benchmark uses the dedicated evaluation toolkit.

STEP 5, PUBLICATION

Only now is it validated.

The publication records the version, the token ceiling, the model policy, and the evaluation result. From here on, the artifact is what developers see and use.

Version

pinned and traceable

Ceiling

recorded per task

Policy

model tier declared

PART

III

Token budget.

All context that enters an agent has a cost. The right layer for each thing is the difference between predictable spend and a billing bug.

HOT, WARM, COLD

Three layers, by frequency and cost to bring in.

01

Hot, every call

copilot-instructions.md, AGENTS.md, custom instructions. Small and stable, because it multiplies on every request. The always-on tax.

02

Warm, on demand

GitHub Copilot Memory, semantic cache on Redis, recent skill results. Cheap to bring in, enters when relevant.

03

Cold, explicit retrieval

RAG and repo embeddings, docs, out-of-scope code, external MCP. Expensive, so bring in little and well.

THE GOLDEN RULE

Push the maximum into warm and cold, and keep the hot tiny.

Every token in the hot layer is paid on every call. Every token in the cold layer is paid only when retrieved.

PART

IV

Catalog and cache.

The control side and the cost side. Only use from the internal catalog, and cache what repeats.

INTERNAL CATALOG GOVERNANCE

A control plane that enforces the policy.

GITHUB SIDE

agents/ plus marketplace

An agents/ directory at org or enterprise level, plus an internal plugin marketplace. The Agent HQ control plane, with an allowlist of agents and models, enforces it: the dev only sees and uses what was validated.

FOUNDRY SIDE

Policy plus fallback set

The model router honors the same Foundry model deployment policy, governed by Azure Policy at deploy time. The configured model subset also acts as a fallback set, blocking prompts from being processed by unapproved models.

TWO CACHES, NOT ONE

Prompt caching and semantic caching are complementary.

PROMPT CACHING, THE VENDOR'S

Identical prefix reuse

Reuse of an identical prefix (KV cache). Anthropic by explicit breakpoint, OpenAI automatic, and Azure AI Foundry too: prefix-based compute reuse for supported models, requiring identical first tokens, with a cache lifetime up to 24 hours.

SEMANTIC CACHING, WHERE REDIS ENTERS

Similarity by embedding

Redis LangCache is a fully managed semantic cache that cuts LLM cost and improves response times. RedisVL SemanticCache is the self-hosted option. Redis also serves as the warm-layer store, not just a cache.

HOW THE SEMANTIC CACHE WORKS

Check before calling the model.

semantic-cache.log

1 check cache before the model call ask the cache first

2 hit, return the cached answer skip the model entirely

3 miss, call the model store the prompt-answer pair

4 tune TTL and similarity threshold the cache handles embeddings and similarity

WHERE THE CACHE SHINES

Agents use more tokens, so reuse pays off.

4x

Agents use about four times more tokens than chat. The repetition surface is large.

UP TO **90%**

Cited reductions on repetitive workloads: internal Q&A, enablement, support, the repeated how-do-I. Not for one-off code generation.

PART

V

Three example agents.

The same gate produces opposite policies, because each agent's measurement leads to different decisions on tier, ceiling, and cache.

AGENT A, AGENT MODE

docs-qa-responder

agents/docs-qa-responder.md

```
model: gpt-4o-mini · tier: economy
max_tokens_per_task: 1200
semantic_cache: enabled
hot: instructions · warm: memory, redis
cache hit 58 to 71 percent
```

High reuse, ideal for semantic cache. A simple question does not justify an expensive model.

AGENT B, CODING AGENT

pr-security-reviewer

agents/pr-security-reviewer.md

```
1 model: claude-sonnet-4-6 · tier: premium
2 max_tokens_per_task: 8000
3 semantic_cache: disabled
4 permissions: contents read, issues write
5 judge favors recall over precision
```

Each PR is unique, so cache is off. Security justifies the most capable model and a high ceiling.

SAME GATE, OPPOSITE POLICIES

The decisions come from measurement, not intuition.

DOCS-QA-RESPONDER

Economy, cache on

Tier economy, ceiling 1,200, semantic cache enabled, high reuse 58 to 71 percent, judge focus on source accuracy.

PR-SECURITY-REVIEWER

Premium, cache off

Tier premium, ceiling 8,000, semantic cache disabled, no reuse, judge focus on recall of findings. Missing a vulnerability is worse than a false alarm.

PART

VI

Agent surfaces.

There is more than one way to run an agent in GitHub Copilot. Each consumes tokens differently, and treating them all the same is the fastest way to blow the budget.

AN ANALOGY TO START

Three ways to work with a capable assistant.

01

You sit beside them

You ask, they do, you review, you ask the next thing. They only work while you are present. This is Agent Mode.

02

You send a task and leave

They work alone for a while and hand you the finished result to review. You do not watch each step. This is the Coding Agent.

03

They act on an event

You leave standing instructions. Every time something happens, they act, with no one watching. This is an Agentic Workflow.

WHAT A TOKEN IS, AND WHY IT COSTS

Two facts change everything.

FACT ONE

Output costs about 5x the input

A token is a piece of text, roughly three quarters of a word. Generating text is more expensive than reading it. An agent that writes long answers costs far more than one that answers tersely, even reading the same thing.

FACT TWO

You pay per call

Each time the agent is triggered, the token bill runs again. A cheap-per-call agent triggered a thousand times can cost more than an expensive one triggered ten times. Keep this for the trigger discussion.

THE THREE SURFACES, SIDE BY SIDE

They charge in different ways.

Agent Mode

In the editor. Many short interactions. Dev in the loop. Charges for the conversation.

Coding Agent

On a GitHub server, alone. Long autonomous session. Charges for autonomy.

Workflows

In an Actions runner, by event. Recurring. Charges for repetition.

The lesson

One policy for all three either over-protects or under-protects. Govern by surface.

Conversation, autonomy, repetition. Three cost profiles, three policies.

SURFACE 1, AGENT MODE

Many short interactions, the hot layer multiplies.

HOW IT CONSUMES

The conversation

The agent lives in the editor, talking to the dev in real time. Each question and answer is a call. Responses are short, but the volume across a workday is high. The hot context repeats on every call.

WHERE IT SAVES

Tiny hot context

If instructions hold 2,000 tokens, ten calls pay 20,000 tokens of repeated instruction before the conversation counts. Multiply by fifty devs. Keep the hot tiny and let Auto pick the model for simple tasks.

SURFACE 2, CODING AGENT

Few long sessions, lots of output.

HOW IT CONSUMES

Long autonomy

It receives a whole task and runs alone on a GitHub server, then opens a pull request. No one is in the loop during the work. It reads many files (cold) and writes a lot (output, which costs 5x). One task can cost more than a full day of Agent Mode.

WHERE IT SAVES

Scope and a session ceiling

A vague task like "improve the code" lets it wander and spend. A specific task like "add email validation to the signup form" has a clear end. The token ceiling is a stopwatch: past the limit, it stops.

SURFACE 3, AGENTIC WORKFLOWS

It acts alone, triggered by an event.

HOW IT CONSUMES

Autonomy times repetition

Inside GitHub Actions, the agent runs every time a trigger fires, forever, until someone turns it off. A workflow on every push in an active repo can run hundreds of times a day. Cost equals cost per run times number of triggers.

HOW IT VALIDATES

Compile, validate, minimal permissions

gh aw compile --validate checks the workflow before production. Permissions must be minimal: an autonomous agent with broad permissions is a security risk, not just a cost one.

THE HIDDEN MULTIPLIER, THE TRIGGER

Same agent, radically different cost.

300 /day

On every push, in an active repo. Highest cost. The human brake disappears in automation.

vs

1 /day

Scheduled once a day for non-urgent work. Lowest cost. Approve the agent AND its frequency.

AGENT C, AGENTIC WORKFLOW

nightly-dep-auditor

agents/nightly-dep-auditor.md

```
1 trigger: schedule · cron: 0 3 * * *
2 model: gpt-4o-mini · tier: economy
3 max_tokens_per_run: 6000 · max_runs_per_day: 1
4 permissions: contents read, issues write
5 the hidden multiplier used in your favor
```

Auditing dependencies is not urgent, so minimum frequency cuts cost without losing value.

PART

VII

The maturity ladder.

Four levels. Each delivers value on its own and prepares the next.

FROM MONDAY MORNING TO FOUNDRY

Four levels of maturity.

L3	QUARTER · ROUTING AND EVALUATION **Foundry**	Model Router selects the optimal LLM per query in real time, Balanced, Cost, Quality modes with failover. Foundry IQ as the retrieval plane. PTUs only after load and hit rate are measured.
L2	MONTH 1 · CACHE AND TELEMETRY **Visibility**	Redis LangCache on Q&A and enablement surfaces. Prompt caching where context is large and fixed. Telemetry of tokens and cost per agent, model, and team.
L1	WEEK 1 TO 2 · GATE AND CATALOG **Discipline**	Golden tasks, measure tokens and cost per agent, ceiling per task and model tier, gh aw compile --validate, publish only the validated, allowlist in the control plane.
L0	MONDAY MORNING · ZERO INFRA **Hygiene**	Lean copilot-instructions.md, AGENTS.md with routing and guardrails, Auto as default, premium off in non-chat, curate GitHub Copilot Memory, credit budgets, export usage data.

THE FIRST 90 DAYS

Three phases, one cadence.

Days 1 to 30, baseline	Config hygiene, lean hot context, Auto as default, premium off in non-chat surfaces, credit budgets and content exclusion. Export the first usage data.	30 days
Days 31 to 60, instrument	Golden tasks per agent, measure tokens and cost, define ceilings and tiers, validate and publish into the internal catalog, configure the allowlist.	30 days
Days 61 to 90, scale	Redis LangCache on Q&A, prompt caching where context is fixed, telemetry per agent and team, prepare the Foundry routing and evaluation plane.	30 days

WHERE EACH DECISION LANDS

Each surface has a dominant lever.

01

Agent Mode, the hot context

Shrinking the always-on context saves on every call, an effect multiplied by volume. This is the dominant lever here.

02

Coding Agent, scope and ceiling

A well-scoped task ends sooner with less accumulated output. The session ceiling caps the long tail.

03

Workflows, the frequency

Reducing the trigger frequency cuts cost in proportion to the disparos avoided. Prefer completion events over each intermediate step.

COMMON MISTAKES

What to avoid.

01 Treating the three surfaces with one policy. Each has a different cost profile. Govern by surface.

02 Validating the agent but forgetting the trigger. A great agent running 300 times a day is a cost problem, not a win.

03 Giving vague tasks to the Coding Agent. "Improve the code" has no end. Specific tasks have a natural stopping point.

04 Inflating the hot context. Every always-on token is paid on every call, by every developer.

05 Broad permissions on an autonomous agent. A security and a cost risk. Minimal permissions for the task, always.

CLOSING

Govern the choice, not the freedom.

Centralize access, decentralize choice. The dev picks among the validated, with a budget and routing.

CONTACT

Paula Silva
Software Global Black Belt

paulasilva@microsoft.com

NEXT STEP

Run the gate on two agents
One Q&A, one review, measure both

Published 2026-06-05

```
// the operating model, in one line
validate → budget → route → cache
per primitive, per surface, per trigger
```