PRODUCT DOCUMENT · WORK ACROSS MODELS

Work across models: the model is a placed resource

Which model does which work, at what cost, and who reviews it. One settings table decides; the engine places every run and refuses by name what does not fit. With our own routing and the figures from our own record.

PUBLIC EDITION

10,556	5	2×	0
agent runs in our record, 1 Oct – 16 Nov 2026	backends and 8 models in use	the estimate is the hard cap on every run	model names in code

APPLIES TO

Solo, Team and Enterprise

every number a setting with a default

PUBLISHED

17 November 2026

Saphan Studio · saphan.ai/product

CONTENTS

01 Summary

02 Words used in this document

03 One step from our record, end to end

04 Units of work and who signs

05 Size, estimate, cap and budget

06 Choosing the model

07 Routing: how we route our own work

08 What each backend does, with real orders

09 When rounds fail, and who reviews

10 Placement and capacity

11 What a person steers

12 What this document does not claim

13 The Saphan ecosystem and this document set

HOW TO READ THIS DOCUMENT

Section 1 is the summary and section 2 the words it uses. Section 3 follows one real step from our record; the sections after it explain each part: who signs, how size and money are counted, how a model is chosen, our own routing and what each backend does, what happens when rounds fail, how seats are filled, and the levers a person holds. The boxes marked “How we run it at Saphan” and every figure come from our own record.

SECTION 01

Summary

Fixing a typo in an error message and building a new payment flow should not run on the same model. Yet a team that runs coding agents usually picks one model and uses it for everything.

Two things then go wrong. Simple work burns the budget of the strongest model. Hard work lands on a model too weak for it, fails, and someone runs it again by hand on a stronger one. Nobody sees the cost until the limit is hit.

Saphan Studio treats the model as a **resource that is placed**, the way a scheduler places jobs on machines:

- every piece of work has a **size**, a **class of work** and a **cost estimate**;
- every **seat**, one vendor account on one machine, has a known model, capacity and cost;
- the engine decides where the work runs, records why, and refuses by name what does not fit.

One settings table decides which model does which work. No model name lives in code. A person sets the table; the engine applies it to every run. The figures in this document come from our own record: 10,556 implementation and review runs between 1 October and 16 November 2026, on five backends and eight models.

SECTION 02

Words used in this document

WORD	MEANING	EXAMPLE
Seat	one vendor account on one machine	a Claude Code account on one build machine; an agent-loop seat that calls NVIDIA’s hosted API on another
Backend	the agent program a seat runs	Claude Code, Codex, Cursor, Qwen Code, Saphan’s own agent loop
Model	the model that backend uses on that seat	Claude Opus 5.5, Claude Sonnet 5.5, GPT-6 Sol, Grok 4.7, Nemotron 3.5, GLM 5.3
Train	a body of work with one goal, one budget and one signed order	the train that builds our new scheduler: 30 steps
Step	one deliverable inside a train	“placement retries when its obstacle may have changed”
Round	one attempt at a step: code, then verification and review	round 3 rejected, round 5 accepted
Estimate and cap	the declared cost of a round, and its hard limit at twice the estimate	estimate $5, cap $10
Owed	a review or act recorded as a debt, visible until it is paid	“security review by Codex: owed”
Unmeasured	a cost or reading that was not taken; never shown as zero	a backend that reports no cost per run

SECTION 03

One step from our record, end to end

A real step from the train that builds our new scheduler: **placement retries when its obstacle may have changed**. Every number below is from the run record, 15–16 November 2026.

ROUND	IMPLEMENTATION: CLAUDE CODE WITH SONNET 5.5	REVIEW: CODEX WITH GPT-6 SOL
1	size M · estimate $50, cap $100 · actual $4.14	reject, four blocking findings
2	size M · estimate $35, cap $70 · actual $2.21	reject
3	size M · estimate $35, cap $70 · actual $1.61	reject
4	**re-cut to size S** · estimate $6, cap $12 · actual $0.65	no verdict: the test could not compile on its host
5	size S · estimate $5, cap $10 · actual $0.25	**accept**
6	size S · estimate $5, cap $10 · actual $0.25	the accepted change absorbed with the trunk

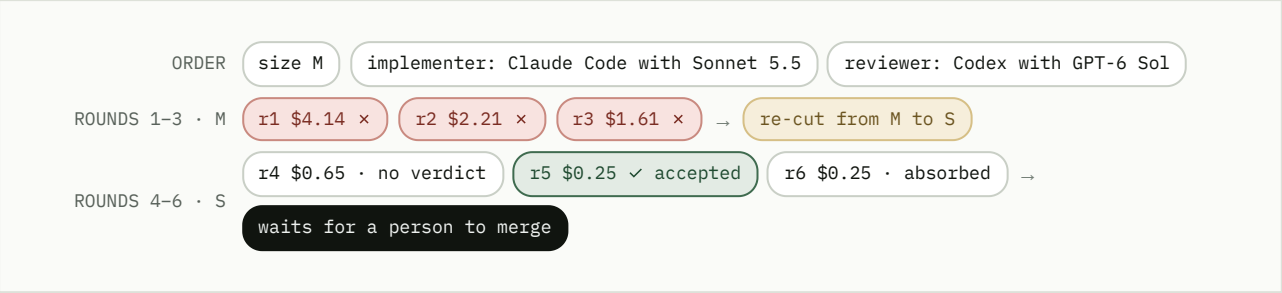


FIGURE 1 · ONE STEP FROM OUR RECORD: THREE REJECTED ROUNDS AT M, RE-CUT TO S, ACCEPTED. IMPLEMENTATION COST PER ROUND AS MEASURED

What the record shows: after three rejected rounds the step was **cut, not abandoned**, from M to S. The cost per round fell from $4.14 to $0.25. A second vendor reviewed every round. The accepted round cost a fraction of its estimate. On 16 November the step was accepted and waiting for a person to merge it.

SECTION 04

Units of work and who signs

`train order signed by a person` → `step order signed by the planner` → `implementer rounds` → `review` → `a person merges`

The roles are those of the orchestration rules: the **planner** plans a train and writes its step orders, the **implementer** codes one round in its own working copy, the **reviewer** judges it, the **tester** checks the whole train, and a **person** approves and merges.

A person signs once per train. The train order carries the budget the planner estimated and a validity window. From then on the planner signs that train's step, round and review orders with its own enrolled key, and the merge stays a human act.

> **How we run it at Saphan.** The train that builds our new scheduler has 30 steps under one signed train order. The planner signs the step orders; a person merges.

SECTION 05

Size, estimate, cap and budget

Work comes in five size classes: XS, S, M, L and XL. Two of them have a written definition:

SIZE	DEFINITION	FROM OUR RECORD
XS	one file, its failing test, no design decision	used on orders today; becoming a class of its own in the engine
M	one package with its tests, about 300 lines or fewer, over an existing seam	the step in section 3 started at M
S, L, XL	the engine's other size classes; an organisation writes its own definitions	the step in section 3 was re-cut to S

MONEY RULE	FROM OUR RECORD
The planner declares an **estimate** on every order, before anything runs	every order in section 3 carries one
The **cap** is twice the estimate, derived by the engine; a run is stopped at its cap	estimate $50, cap $100
A run over its estimate but under its cap still finishes, and is reported when it closes	an Opus 5.5 order estimated at $80 cost $95.48
The **train budget** is itself an estimate: the planner adds up the estimates of the train's steps, and a person signs it with the train order	a 30-step train quoted at $1,310 has spent $231 of measured cost after 16 days
Every signed step order's estimate counts against the train budget; a step beyond it is refused `budget-exceeded` until a person raises the budget	—

Cost calibration: a separate vector of model calibration

Each model is calibrated on more than one vector. **Cost is a vector of its own**, kept apart from what a model can do. For every run the record holds the estimate and the actual cost, per model and per class of work, and the estimates are corrected from it.

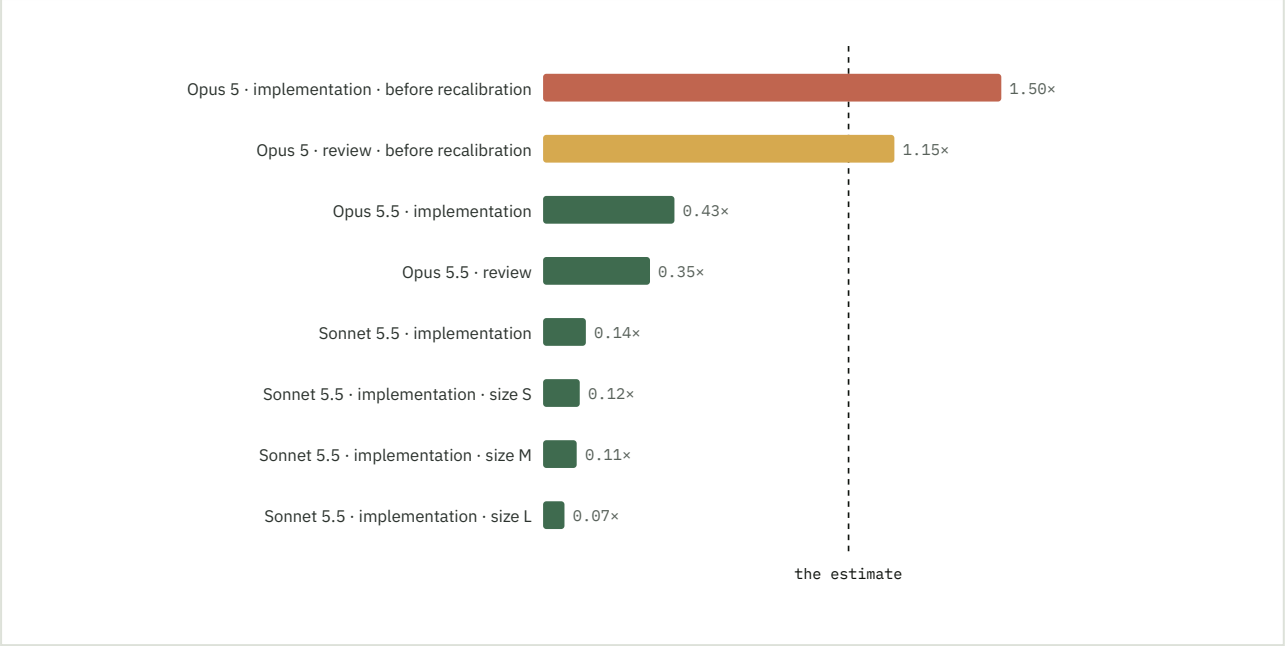


FIGURE 2 · MEDIAN ACTUAL COST AS A MULTIPLE OF THE ESTIMATE, CLAUDE CODE RUNS, 1 OCT – 16 NOV 2026 (OPUS 5 ROWS: BEFORE THE ESTIMATES WERE RECALIBRATED IN OCTOBER). OTHER BACKENDS REPORT NO COST PER RUN: UNMEASURED, NOT ZERO

> **How we run it at Saphan.** In the first week of October, 77 percent of Opus 5 implementation runs cost more than their estimate. The estimates were recalibrated. With Opus 5.5 and Sonnet 5.5 since then, the median actual cost is 7 to 43 percent of the estimate. Only Claude Code reports a cost per run; the other backends are unmeasured, and the record says so instead of showing zero.

SECTION 06

Choosing the model

The model for a run is chosen by one rule, the same for every vendor. The first match wins:

01 a model named on the dispatch itself;

02 the model named on the order;

03 the model of the seat class (lane) the order asks for;

04 the most specific row of the settings table for this role, backend and class of work;

05 the default for this role and backend;

06 otherwise the run is **refused by name**. The engine never guesses a model.

What a model may take

Capability belongs to the model, never to the vendor or the backend. Model classes cut across vendors: NVIDIA’s Nemotron and Anthropic’s Claude Haiku sit in the same class, below Claude Sonnet and Claude Opus. Each model has its own largest size of work, and its own classes of work it may and may not take. Work above a model’s size is cut, not failed, as in section 3.

	CLAUDE CODE OPUS, SONNET	CODEX GPT SOL	CURSOR GROK 4.7	AGENT LOOP NVIDIA NEMOTRON	QWEN CODE GLM, QWEN
Plans for a train	**Opus, max effort**	—	—	—	—
Development, S and M	**develops**	—	**develops**	**first choice**	**M and smaller**
Development above M	**first choice**	—	**develops**	cut to M	cut to M
Network work	**first choice**	never	—	—	—
Security and UI development	**first choice**	—	**develops**	never	never
Named review	**fresh session**	—	—	**second model only**	—
Security and UI review	—	**Codex only**	—	**if Codex is busy; Codex owed**	never

first choice also used never, or cut first

FIGURE 3 · WHO DOES WHICH WORK AT SAPHAN, BY BACKEND, AS OUR ROUTING STANDS SINCE 13 NOVEMBER 2026. EVERY CELL IS A SETTING

> **How we run it at Saphan.** NVIDIA Nemotron is first choice for S and M work, even though Claude Code carries most of the runs, because we chose so in our own configuration, like every other cell here. A Nemotron run costs us the least, so we send it small work first and let it try. When it does not manage, only a cheap round is lost: the round rule moves the step to a stronger model, and Nemotron is never the only reviewer. Another organisation sets this cell its own way.

SECTION 07

Routing: how we route our own work

Routing is fully customisable by rules. The organisation decides which model does which work and who reviews it. The table below is our own routing at Saphan, in force since 13 November 2026.

WORK	AT SAPHAN	NEVER
Plans for a train	Claude Code with Opus, at maximum effort	—
Development	Claude Code; Cursor with Grok 4.7 at any size and class	—
Small implementation work, S up to M	Saphan's agent loop with NVIDIA Nemotron first; Qwen Code with GLM and Qwen models at M and smaller	—
Network work	Claude Code	Codex
Security and UI development	Claude Code, Cursor	Qwen Code, the agent loop
Named review	Claude Code in a fresh session, after the planner's own verification	the author's own session
Security and UI review	Codex only; if Codex is busy, the agent loop reviews and the Codex review is recorded as owed. Work never waits	Qwen Code

WORK	AT SAPHAN	NEVER
NVIDIA Nemotron as a reviewer	a supporting second model only	the only review

> **How we run it at Saphan.** On 13 November we moved development to one vendor, narrowed another vendor to security and UI reviews, and turned NVIDIA's model from first reviewer into a supporting second reviewer. The change was one decision, recorded and signed in our rules, and in force across all our seats the same hour. No code changed.

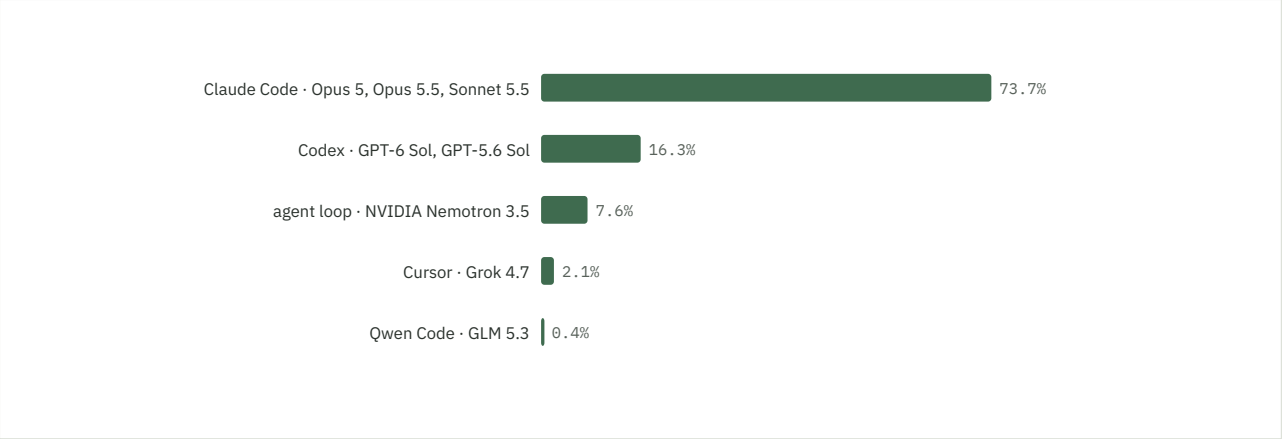


FIGURE 4 · SHARE OF RUNS BY BACKEND, 10,556 IMPLEMENTATION AND REVIEW RUNS IN OUR RECORD, 1 OCT – 16 NOV 2026

Claude Code carries three quarters of our runs; Codex is the main second vendor, mostly for reviews; Nemotron carries about 8 percent; Grok 4.7 and GLM are present and smaller. Dollar shares between vendors cannot be compared, because only Claude Code reports a cost per run.

SECTION 08

What each backend does, with real orders

The same routing, as the work each backend gets. Each order is a real one from our record, 2–16 November 2026.

BACKEND	ITS WORK AT SAPHAN	A REAL ORDER
Claude Code with Opus 5.5	plans at maximum effort; named reviews in a fresh session; large and core work	"the record store", estimated $80, cost $95.48
Claude Code with Sonnet 5.5	development	"a wrapped login URL never reads code", estimated $4, cost $0.67
Codex with GPT-6 Sol	security and UI reviews only, since 13 November	"desktop works standalone"
Cursor with Grok 4.7	development at any size and class	"the server image copies what it builds"
Saphan's agent loop with NVIDIA Nemotron 3.5	implementation at S and M first; a supporting second reviewer	"a slug that cannot commit is refused early", size S

BACKEND	ITS WORK AT SAPHAN	A REAL ORDER
Qwen Code with GLM and Qwen models	work at M and smaller	no run between 2 and 16 November; the last was a GLM 5.3 review on 27 October

SECTION 09

When rounds fail, and who reviews

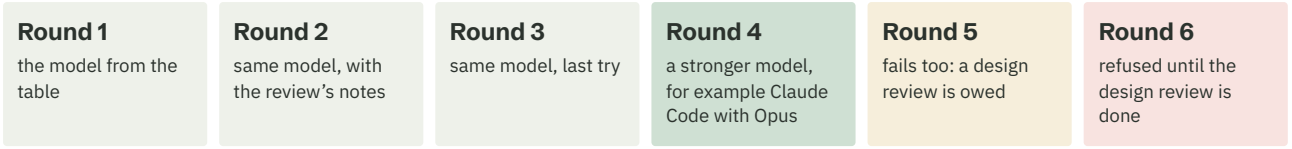


FIGURE 5 · WHAT HAPPENS AS ROUNDS FAIL. THE THRESHOLDS ARE SETTINGS

AFTER	THE ENGINE
3 failed rounds	moves the step to a stronger model, for example to Claude Code with Opus
5 failed rounds	records a design review as owed
a 6th round	refuses it until the design review has happened

> **How we run it at Saphan.** In the train that builds our new scheduler, 12 of the 30 steps needed four or more rounds. That is the reason this rule exists: a planner should not have to notice and move a step by hand.

Review pairing: the second model

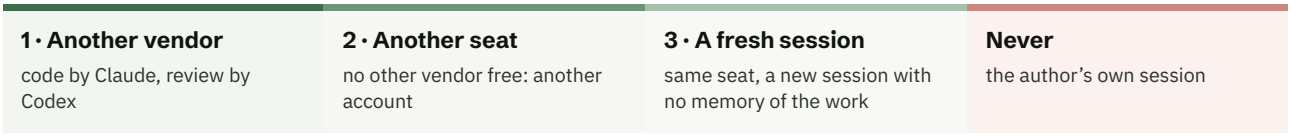


FIGURE 6 · THE INDEPENDENCE LADDER: THE FIRST FREE RUNG IS USED, AND A SKIPPED NAMED REVIEW IS RECORDED AS OWED

- **Independence ladder.** A review runs on another vendor; if none is free, on another seat; if none, on the same seat in a fresh session. Never in the author's own session. Cross-vendor review is a setting.
- **An owed review stays visible.** When the named reviewer is busy, the review goes to the next reviewer and the named review is recorded as **owed**.
- **A missing review is never silent.** When no review was written, the engine writes a placeholder that says so and carries the owed review. Every step holds the same set of documents.

> **How we run it at Saphan.** Codex reviews our security and UI work. When Codex is busy, the agent loop reviews, the work does not wait, and the record shows "security review by Codex: owed" until Codex has done it.

SECTION 10

Placement and capacity

A run is composed from a few inputs: the order, the role, the estimate and a machine, or *auto*. Everything else is derived from the record: backend, binary and transport from the machine's record; base, branch, working directory, identity and the cap. The engine renews its own machine leases and clears its own stuck work, so a person sees only the acts that need a person.

RULE	WHAT IT MEANS
Signed orders wait in a stock; the scheduler places them continuously as seats become free	work queued at night starts as soon as a seat fits it
A seat is chosen only if it is allowed for the class of work and its model may take the work's size	an agent-loop seat is never offered a security step
Seats are tried in order of remaining headroom, never round-robin; a refusal costs nothing, and is classed: try the next seat, try the same seat later, or stop	a full seat is tried again later; a defect that will not clear is not retried
One seat is one vendor account with a session window and a week window; whichever is spent binds. A reading that was not taken is unmeasured, never zero	0% of the session used but 100% of the week: the seat cannot run

SECTION 11

What a person steers

Every change to a lever is a recorded decision. Only a person merges.

LEVER	EFFECT
Sign a train order	starts the train, approves the budget the planner estimated, delegates its steps to the planner
Edit a step	its vendor and model (for that step only); its estimate, with the cap at twice; the machines it may use and a preferred seat, neither of which pins it; its review rule
The settings table	models per role, backend and class of work; size limits; caps; escalation thresholds; scheduler bounds
Standing rulings	organisation-wide routing, recorded with the person's own words and the date, like our change of 13 November

SECTION 12

What this document does not claim

- The routing and figures are ours, from our record of 1 October to 16 November 2026. Costs are measured where the agent reports one per run, today Claude Code; the rest is unmeasured, never zero.
- It describes what the scheduler decides, not how it is built. Saphan Studio's governance approach is U.S. and EU patent pending.

SECTION 13

The Saphan ecosystem and this document set

Saphan Studio is one product in a small ecosystem. The tools around it are open source under Apache 2.0 and each works on its own; together they cover a change from the first read of the code to the evidence that it is right.

`Discover reads the code` → `Skill Studio checks the skills` →
`Saphan Studio runs the agents under a person's approval` → `Mock Studio stands in for real systems` →
`Test Studio and Proven judge the result`

TOOL	ITS PART
Saphan Studio saphan.ai	The control plane: signed orders, confined agents, independent review, human approval, one signed record.
Discover github.com/saphan-ai/discover-studio	Reads a codebase before anyone changes it, with a source for every claim; proposes the skills that fit it.
Skill Studio github.com/saphan-ai/skill-studio	Checks and signs the skills agents are given, so anyone can verify them again.
Mock Studio github.com/saphan-ai/mock-studio	Stand-in systems agents and tests work against instead of the real ones.
Test Studio github.com/saphan-ai/test-studio	Shows what a test run really proved, in the framework the team already uses.
Proven github.com/saphan-ai/proven	An AI code reviewer that reports a problem only with evidence.

The document set

DOCUMENT	WHAT IT ANSWERS
Saphan Studio for developers and DevOps	start here: what Saphan Studio is and how the pieces fit
Skills and craft	what every agent carries into the work
Testing and review	how a change earns its merge
Orchestration rules	a deterministic engine, model work inside it
Security	what an agent can reach, and what you can prove afterwards
Agent framework flows	how a customer's LangGraph flow runs signed, confined and recorded
Work across models (this document)	which model does which work, at what cost, reviewed by whom

For partners: a founding customer onboarding guide and an Enterprise continuity plan, at saphan.ai/for-partners.

All documents: saphan.ai/product · Code quality against published baselines: saphan.ai/research/proven-effort · Operator documentation: docs.saphan.ai · Contact: hello@saphan.ai