PREVIEW Draft v1.0 · scaling measured to 17 pods · 3 November 2026

Amber panels mark measurements not yet taken.

Reading a million pages of court records: throughput, quality and cost of three OCR engines

Measurements on the lavoro_ocr perimeter of the Digesta corpus — tesseract, Surya and Chandra on an elastic GPU fleet

Stefy Lanza (nextime)

stefy@nexlab.net

Nexlab — Digesta project

Measurements 28 October – 3 November 2026

ABSTRACT

Optical character recognition over a judicial corpus raises three questions that the literature tends to treat separately: how many pages per second an engine yields on a given GPU, how much of what it reads is correct, and what it costs to read everything. We measure all three on the same set of pages — 948,156 scanned pages belonging to 98,545 documents — comparing one classical engine (tesseract, already applied to the corpus) against two recent vision-language models (Surya and Chandra) served on NVIDIA A40 GPUs rented by the hour.

The main methodological result is a negative one, and it matters more than the absolute figures: **the throughput of an OCR engine on a GPU is dominated by the concurrency offered to the pod, not by the engine**. The same model, on the same hardware and the same pages, yields between 0.103 and 0.588 pages per second per pod — a factor of 5.7 — as the concurrency offered to it and the orchestrator's admission settings alone are varied. Any throughput figure published without that parameter is not reproducible. From that follows the contribution we consider the most useful of this work: since the optimum depends on the configuration, the fleet and the page mix, and moves while the work runs, it should not be configured at all but **searched by a closed loop that keeps searching** — an extremum-seeking controller over the concurrency, described in §3, which found every optimum reported here and held it through pods arriving, pods dying and a fivefold change in fleet size. We also show that the fleet then scales **linearly** from

one to seventeen pods, so the collapse to 0.103 measured in a production pass was a starved client and not a shared ceiling; and that five different 48 GB cards all land within 0.52–0.60 pages per second per pod, which makes cost per page a function of the hourly price alone. On the quality side we show that the two new engines do not replace the old one but arbitrate it: Chandra settles 51.5 % of the digit groups on which tesseract and Surya contradict each other, and the per-page confidence Surya reports does not discriminate its own errors.

Keywords — OCR, court records, vision-language models, GPU throughput, engine arbitration, cost per page

---

1. The problem
2. Perimeter and data
3. Method, instruments, controller
4. The page as the unit
5. The lane curve
6. The 302-second wall
7. Fleet scaling
8. Reading quality
9. Arbitration on digits
10. Reported confidence
11. Cost
12. Threats to validity
13. Provisional conclusions
14. Measurements still missing

---

1 The problem

A digitised judicial archive is not a set of homogeneous images. Natively digital filings with their own text layer sit beside 150 dpi scans produced by office photocopiers, hand-stamped pages, damage-assessment tables, and expert reports with photographic annexes. On this material the operational question is not "which OCR engine is best", but *what it costs to bring the whole archive to a reading quality sufficient for legal research*, where sufficiency is almost always decided by a number: a case number, an amount, a date, a tax code.

Three constraints make the question non-trivial. First, the corpus has already been read once by tesseract, and that reading is neither disposable nor trustworthy. Second, the vision-language engines that read better run on GPUs rented by the hour: throughput is not an engineering curiosity, it is the price. Third, no ground truth exists for these pages, and producing one by hand costs qualified human labour; a way is therefore needed to estimate relative quality between engines without it.

This work measures, in order: the request shape worth sending to a GPU pod (§4), the concurrency that saturates one pod (§5), the limit that truncates the slow tail (§6), the return on adding pods (§7), quality compared three ways (§8–10) and the cost that follows (§11).

2 Perimeter and data

The perimeter is defined by selection over the corpus, not by sampling: every primary document of the trial-court holding that has a local file and at least one page without reliable text. Table **1** reports its measured — not estimated — footprint.

QUANTITY	VALUE	NOTE
Documents in the perimeter	98,545	all with a local file
Total pages	1,114,432	—
Pages requiring OCR	**948,156**	**basis of every cost estimate**
Pages examined by the escalation gate	957,687	full scan, 2.1 min
Placeholder pages ("this page left blank")	234	0.024 %, to be excluded
Pages selected for a second reading	511,211	53.4 % of 957,687
Pages recovered from the mixed text layer	81,369	from 2,740 documents, 182.2 M characters

TABLE 1 Perimeter footprint. The "pages recovered from the mixed text layer" are pages already legible inside the native PDF and saved with no GPU spend; they are subtracted from the work, not from the count.

The *escalation gate* — the rule that sends a page to a second engine instead of accepting the existing reading — is deliberately coarse and transparent: `dpi ⩽ 150 OR characters = 0 OR digit_groups > 10`. The three clauses pick out, respectively, the poor scan, the outright failure of the first reading, and the

number-dense page where an error costs more. Together they cover 53.4 % of the perimeter, split across three buckets of very unequal size (table 5).

Every measurement in this work draws on the same page set, selected by `doc_id` from the corpus with a fixed seed, so that comparisons between engines run on identical material rather than on independent samples.

3 Method, instruments, and the controller that finds the optimum

No engine was queried directly. All calls pass through an internal orchestrator that exposes a single OCR API, enforces the admission contract, and starts or stops GPU pods at the provider according to load. The chain is the one in figure 1: what we measure is always the time as seen by the caller, network and queueing included, because that is what is paid for.

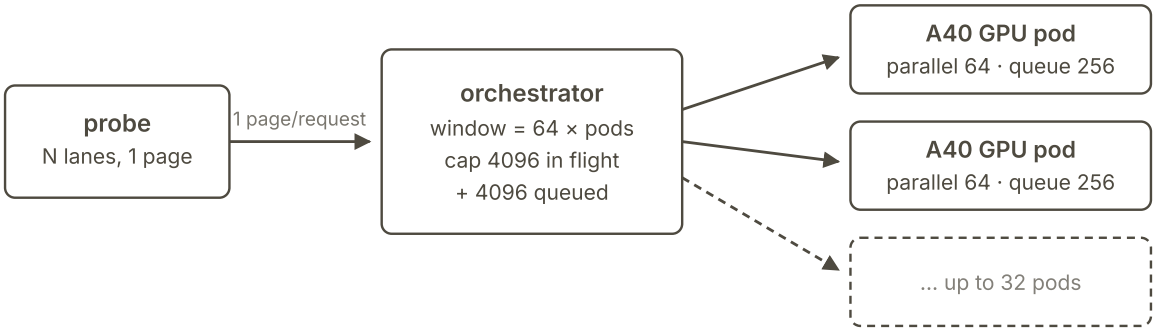


FIGURE 1 Measurement chain. The experimental parameter is a single quantity — the number of requests the probe keeps in flight towards a pod, which we call *lanes*. Everything else is held fixed.

How the models are run

The orchestrator is **CoderAI** (`github.com/nextime/coderai`), an open-source inference orchestrator developed in-house and used here unmodified. It is the single point through which every call in this work passes, and it is what makes the measurements reproducible: the parameters §5 turns out to depend on are its settings, not ours.

Its arrangement for this work is worth stating precisely, because each element shows up in a result:

- **The orchestrator runs without a GPU**, on the same host as the corpus, from the container image `ghcr.io/nextime/coderai` . It holds no model; it admits requests, keeps a concurrency window, and rents or releases GPU pods at the provider.
- **The engines run on rented pods**, from `ghcr.io/nextime/coderai-vllm-ocr` , each pod serving the OCR model under its own vLLM over loopback. A page is sent to the orchestrator as a one-page PDF and *rasterised on the pod*, not on the host — so the host does no image work and the per-page cost is GPU-side.
- **One API, one admission contract.** Callers see `/v1/ocr` and `/v1/ocr/batch` ; the orchestrator enforces its own front window (`max_parallel_requests` , `queue_max_size`) and, per pod,

`pod_max_parallel_requests` and `pod_queue_max_size`. Table **3b** is entirely a story about those four numbers.

- **Pods are leased, not allocated.** A client asks for a floor and a ceiling (`pods`, `max_pods`) with a 30-minute lease that must be renewed; the two sent equal pin the fleet exactly, and the floor alone lets the autoscaler grow toward the operator ceiling — asking for 8 without a ceiling once returned 13 billed pods, which is why every measurement here re-asserts both.
- **Card policy is declarative:** `gpu_types`, `cloud_types` (secure or community), `min_vram_gb` and `max_hourly_usd`. §11 is the consequence — raising the price ceiling to admit dearer cards is what broke the capacity deadlock, and lowering it again is what reduced cost per page.
- **Direct TCP to the pod** rather than the provider's HTTP proxy, which cuts a request at 100 s: with pages whose residence time reaches 130 s at high concurrency, the proxy path would have truncated the slow tail invisibly and the lane curve would have been wrong in a way no error count would reveal.
- **Timeouts are matched at both ends** (1800 s on the orchestrator and on the client), deliberately: at high concurrency a page queues legitimately for minutes, and a client that gave up first would discard work already paid for.

Nothing in this work calls the provider's API directly, and no engine is reached except through that orchestrator. That is a constraint of the installation rather than an experimental choice, but it has the convenient effect that every number here is measured at the same boundary a production job would see — network, admission and queueing included.

The two probes

The **shape probe** varies how many pages go into one request and how many requests are in flight, at a fixed request count, and measures the time to complete them. It is there to choose the contract, not to measure peak speed.

The **saturation probe** does the opposite: it fixes a time window, keeps *N* lanes continuously busy, and counts how many pages completed when the window closes. This is the correct instrument for a throughput curve, because it does not confound speed with the luck of drawing easy pages, and it records separately the pages lost and *the time at which* they were lost — a measurement that turns out to be decisive (§6).

Every case re-asserts the pod count before starting, because the reservation at the provider expires; and every case writes, page by page, characters, digit groups, lines, regions, tables and seconds, without the text, so that the quality analysis can be redone afterwards over the same files.

The controller: the optimum is not a setting, it is a search

The two probes above measure a curve at a fixed concurrency. They are adequate for an experiment and inadequate for production, for a reason §5 makes explicit: the position of the optimum is a property of the configuration, the fleet and the pages, not of the engine. It moved from 128 lanes to 48 when the orchestrator's send path changed. It would move again with a different card mix, a different resolution, or a corpus of denser pages. A number measured on Monday and written into a configuration file is

wrong by Wednesday, and wrong silently — a starved fleet looks exactly like a busy one from the outside, which is precisely how the 0.103 pass passed unnoticed.

So the production passes reported here do not take the concurrency as a parameter. They run a loop that *hunts* it while reading real pages, and never stops hunting. Its rules, in the order they matter:

RULE	WHAT IT DOES	WHY
Objective	maximise pages/s **per pod**	With the fleet total, adding a pod makes every probe "win" and the loop chases the pod count instead of the concurrency. Per-pod is also the cost metric.
Probe size	wait for **150 completed pages**, not a fixed time	At one pod a 60 s window holds ~20 pages; counting noise alone is ±22 %, four times the 5 % threshold the loop decides on. Measuring per clock makes the loop chase its own noise.
Coarse search	walk 8, 16, 32, 48, 64, 96, 128 lanes while throughput improves; fall back to the best when a step is slower	Brackets the optimum in a handful of probes without assuming where it is.
Refinement	then ±2, then ±1 lane, and keep probing ±1 for the rest of the run	The top of the curve is a plateau: ±1 tracking confirms it still is, rather than declaring victory once.
Adaptive step	double after two consecutive wins in a direction (cap 16), halve on any loss	Quick to retreat, deliberate about committing — the right asymmetry when each probe costs 150 pages of real work.
Plateau	at ±1 resolution, hold position instead of drifting	Continuing in one direction at the finest resolution turns window noise into a slow walk away from the peak.
Fleet change	a pod arriving or dying does **not** restart the search	Lanes are per pod and the objective is per pod, so the remembered optimum stays valid; the loop corrects from it instead of climbing from the bottom.
Scaling trigger	request more pods when the lane count stays inside ±5 for three probes	A criterion on the control variable, which noise cannot fake, rather than on the noisy output.
Persistence	write the optimum to disk; a new process resumes from it	Without this, every restart spent 10–20 minutes at 8 lanes starving the pods — measured, and the reason it exists.
Hard brake	on a queue-full refusal, cut lanes 20 % and zero the integral	A refusal is the pool's own statement that it will take no more; it is the only signal the loop obeys without measuring.

TABLE 2B The concurrency controller. Everything it does is driven by one measured quantity, pages per second per pod, and one control variable, lanes per pod.

Three ways this loop goes wrong

Running it for twelve hours over a growing fleet exposed three failure modes, none of which is obvious from the rules and all of which cost measurable throughput. They are reported because a reader

implementing this will meet them.

1. Climbing the control variable is self-rewarding

This is the serious one, and it is intrinsic to extremum seeking on a queueing system. Raising the lane count dispatches more requests at once; their completions arrive together; the probe counts them as throughput. So **the act of increasing concurrency manufactures, for one window, the very gain the loop is looking for** — and the adaptive step amplifies it, since two apparent wins double the step.

Measured: the loop climbed 53 → 55 → 59 → 67 → 83 → 99 lanes reading six consecutive wins (0.536 → 0.842 pages/s per pod). The window after that read 0.345, with residence time up from 90 to 287 s: the queue had simply absorbed everything. None of those six gains were real.

The remedy is cheap: after every change of the control variable, **the next window makes no decision** — it only re-establishes the baseline, once the in-flight level has settled. It halves the decision rate and removes the bias.

2. A lucky window freezes the search

The loop compares each probe against the best seen. Keep that best forever and a single outlier ends the search: a window reading 0.910 pages/s per pod against a median of 0.558 meant no subsequent probe could ever "win", so the loop took the retreat branch every time and sat dithering ±1 lane around 53 for an hour, believing it was tracking. The fix is to **decay the best by 1 % per probe**, so an outlier washes out in about ten windows while a genuine optimum survives.

3. A pathological state must not be persisted

Persistence across restarts is what stops each new process starving the fleet for twenty minutes (§3, last rule). But it cuts both ways: after the collapse described above, the retreat bottomed out at the loop's floor of 4 lanes, that value was written to disk, and the next process resumed from it. **Twenty pods then ran at 0.12–0.16 pages/s per pod — a quarter of capacity — for some fifty minutes** before it was noticed. Two guards follow: a floor on the control variable taken from the measured curve (8 lanes/pod already yields 0.150 against 0.50 at 48, so nothing below 16 is worth exploring), and a refusal to persist any value below that floor.

A design we tried first, and why it failed

The obvious formulation is a PID on *queue depth*: throughput has no setpoint — one wants its maximum — but queue depth does, and holding a couple of pages waiting per pod should keep the GPU fed while costing no latency. Queue depth is available without timing anything, by Little's law: residence time is in-flight divided by throughput, and comparing it with the lowest residence time ever observed gives the queue.

It failed, and the failure is instructive enough to report. The reference — the lowest residence time seen — is a *minimum over noisy windows*, so it drifts to the optimistic extreme: one lucky window set it to 10.9 s against a true ~35 s, after which every ordinary window looked like a vast queue. The loop cut lanes from 19 to 4 and throughput fell from 0.433 to 0.100 pages per second — a 4× self-inflicted collapse, while the

outer loop, seeing throughput fall, lowered the queue target further and made it worse. Ageing the reference upwards helps but does not fix the asymmetry.

Hence the rule adopted: **search the objective you actually want, even when a proxy is more elegant.** A controller on a derived quantity inherits every bias of the estimator that derives it; a controller on measured throughput is cruder, converges more slowly, and cannot fail in that direction. We report both because the elegant version is what a reader would reach for first.

4 Request shape: why the unit is the page

What the pre-OCR pass decides

Before any GPU is involved, a pass over the corpus explodes every document into pages and gives each page its own verdict. It is cheap and it is the reason everything downstream is page-shaped: 957,687 pages examined in 2.1 minutes, each one classified as carrying a usable text layer, as a scan needing an engine, or as a figure with nothing for an OCR to read. The same pass records, per page, the native resolution, the character count of the existing tesseract reading and the number of long digit groups — the three quantities the escalation gate of §2 is built from.

Two of its results are worth stating because they remove work rather than add it. 81,369 pages across 2,740 documents turned out to be already legible inside the native PDF (182.2 M characters), and were saved as partial readings without spending a GPU second; and 234 pages are printed placeholders ("this page intentionally left blank"), which an OCR would faithfully read as almost nothing and a quality gate would then flag forever.

So the page, not the document, is the unit of decision: it has its own verdict, its own engine, its own confidence and its own record. A document is assembled only once every one of its pages has an outcome, and where a page failed everything the document is assembled *with a hole that is recorded* rather than smoothed over. That has three consequences for how requests are shaped:

- **Mixed documents are the norm, not the exception.** A filing commonly has a digital text layer for the first pages, 150 dpi scans in the middle and a photographic annex at the end. A document-level request would send all of it to one engine, paying GPU time for pages whose text we already have.
- **The state is per page.** Each completed page is appended to a record file, which *is* the resume point: a run interrupted mid-document — a pod lost, an orchestrator restarted, a chunk killed — resumes at the next unread page rather than re-reading from a document boundary. Over the evening described in §7 that happened repeatedly, and cost nothing.
- **Per-page records are what make quality measurable at all.** Every figure in §8–10 is a comparison of three readings of *the same page*; with document-level requests there would be no such unit to compare, and no way to arbitrate a digit.

The page unit is therefore a requirement of the architecture. What the next measurement establishes is that it costs nothing — which was not obvious, since one request per page multiplies the per-request overhead by the number of pages.

Batching does not help

The natural intuition is that packing pages into one request amortises the fixed per-request cost. On these engines it is false, and not marginally so (table 2). Batching degrades throughput by an order of magnitude and introduces failures: at 24 pages per request over four lanes, Chandra returned 500 on every request and read zero pages in 22 minutes.

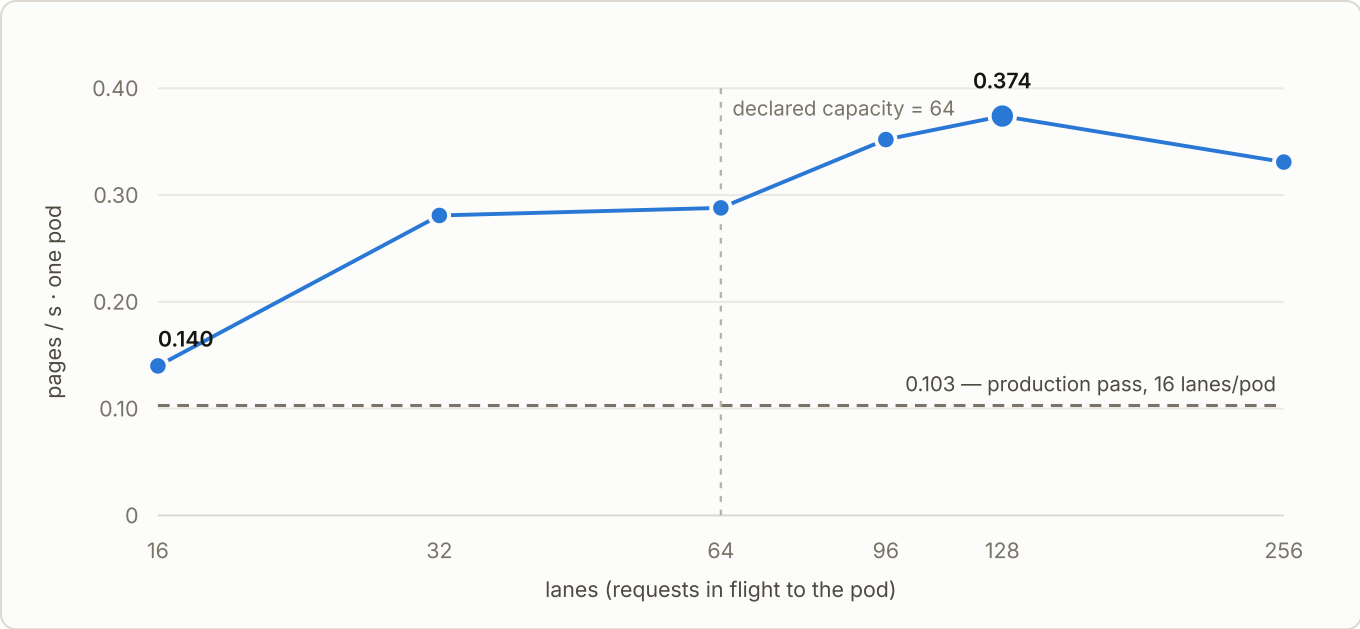
SHAPE	LANES	CHANDRA PG/S	SURYA PG/S	OUTCOMES
1 page / request	1	0.023	0.038	all complete
1 page / request	4	0.034	0.023	all complete
1 page / request	**16**	**0.081**	**0.063**	**all complete**
8 pages / request	1	0.010	0.012	2 of 3 failed
24 pages / request	1	0.041	0.000	Surya: all 502
24 pages / request	4	0.000	0.040	Chandra: all 500
24 pages / request, 150 dpi	4	0.000	0.205	Chandra: all 500
24 pages / request, no continuation	4	0.078	0.111	2 of 4 failed

TABLE 2 Shape probe, one pod, 200 dpi unless stated. Run at a fixed request count rather than a fixed window: the absolute values are not comparable with §5, their ordering is. Conclusion adopted for the rest of this work: **one page per request**, with concurrency obtained by adding lanes.

One row is worth noting, the only one where batching wins: Surya, at 150 dpi, over four lanes, yields 0.205 pg/s against 0.040 at 200 dpi — a factor of 5 from resolution alone. This suggests that for Surya the lever is resolution rather than request shape; we did not isolate it, and it remains among the missing measurements.

5 The lane curve

This is the central measurement. One pod, Chandra, identical pages, fixed window; only the number of in-flight requests varies. Throughput rises up to 128 lanes — **twice the capacity the pod declares**, which admits 64 in parallel and queues 256 — and falls back at 256.



Chandra, one A40 pod, 200 dpi, one page per request, windows from 547 to 1,179 s.

FIGURE 2 Throughput against offered concurrency. The peak sits at 128 lanes, 2.7× the yield of the production pass and 2× the capacity the pod declares: offering more requests than the pod will accept *pays*, because its local queue keeps it busy across network round trips.

LANES	PG/S	×	PAGES READ	WINDOW S	MEDIAN WAIT S	P90 WAIT S	ERRORS
16	0.140	1.00	116	826.0	42.1	76.8	3
32	0.281	2.01	154	547.2	66.7	114.3	6
64	0.288	2.06	186	645.2	118.3	207.9	11
96	0.352	2.51	324	921.6	162.9	288.9	20
128	**0.374**	**2.67**	**352**	**941.0**	**212.9**	**409.4**	**25**
256	0.331	2.36	390	1,178.5	368.4	682.6	89

Reference: production pass over 32 pods at 16 lanes per pod → 0.103 pg/s per pod.

TABLE 3 The full curve. Median wait grows by a factor of 5 between 16 and 128 lanes: in a batch pass that is free latency, since nobody is waiting on an individual page. The curve justifies the one operating parameter that has to be configured at all.

The curve is a property of the configuration, not of the engine

Repeating the same measurement on 2 November, after three corrections to the orchestrator and the pod, moved the optimum from 128 lanes to about 48 — and raised the peak by 55 %. The corrections were: sending to pods asynchronously instead of synchronously (`pod_http_async`), raising pod admission from 128 parallel / 256 queued to 192 / 512, and raising the orchestrator front from 256 / 2048 to 1024 / 4096. A fourth, on the client side, was reusing one HTTP client with keep-alive instead of building one per page.

LANES PER POD	PG/S PER POD, 1 NOV CONFIG	PG/S PER POD, 2 NOV CONFIG
8	—	0.150
16	0.140	0.300
24	—	0.392
32	0.281	0.431
48	**—**	**0.500**
64	0.288	0.490
96	0.352	—
128	**0.374**	—
256	0.331	—

TABLE 3B The same engine, card and pages under two configurations. The optimum is not a property of the model: with a synchronous send path and a tighter admission window the pod needed 128 requests thrown at it to stay busy; with an asynchronous path and a wider window it saturates at about 48 and queues beyond that. Between 45 and 55 lanes the measurements are within each other's noise, so the optimum is a plateau and lane precision finer than ±5 buys nothing.

Two readings of the same table are worth keeping apart. The first is engineering: the concurrency parameter should go from 16 to 128 lanes per pod, for a measured gain of 2.67×. The second is methodological, and is the thesis of this work: **a pages-per-second-per-GPU figure published without the in-flight request count means nothing**. The spread between the lowest and highest value we measured on the same model and the same card is 3.6×, enough to invert any comparison between engines.

6 The 302-second wall

Recording not merely how many requests failed but at which second exposed a hard limit. The failures are not spread out: the first of each series arrives at 302.0, 302.1 and 302.5 seconds, with an identical body — the pod answers `500` with "Chandra model call failed" and no further detail. A floor that flat, across three independent cases, is a timeout, not a coincidence.

LANES	ERRORS	FIRST FAILURE S	MEDIAN FAILURE S	COMPOSITION
96	20	306.6	352.9	20 × 500
128	25	302.0	402.4	25 × 500
256	89	301.2	894.0	27 × 500, 44 timeouts, 18 × 503

TABLE 4 Time to failure. The slow tail that high concurrency generates is truncated at roughly 302 s: the hardest pages are not read badly, they are discarded. It follows that **the values in figure 2 are a lower bound**.

The consequence matters for interpretation. The lane curve does not measure the pod's ceiling but the pod's ceiling *with the tail truncated at 302 seconds*: part of what high concurrency gains in GPU occupancy it gives back in discarded pages. The limit was raised to 900 s on the orchestrator after this series closed, but the pod that produced the numbers had been started before the change and never received it: the check remains to be done on a fresh pod.

> MISSING MEASUREMENT
>
> Repeat the 96, 128 and 256 lane cases on a pod started after the limit was raised, comparing both throughput and the share of pages lost. The expectation is a value above 0.374 with fewer errors; if the wall at 302 s survives, the limit is internal to the model server and not configurable from outside.

7 Fleet scaling

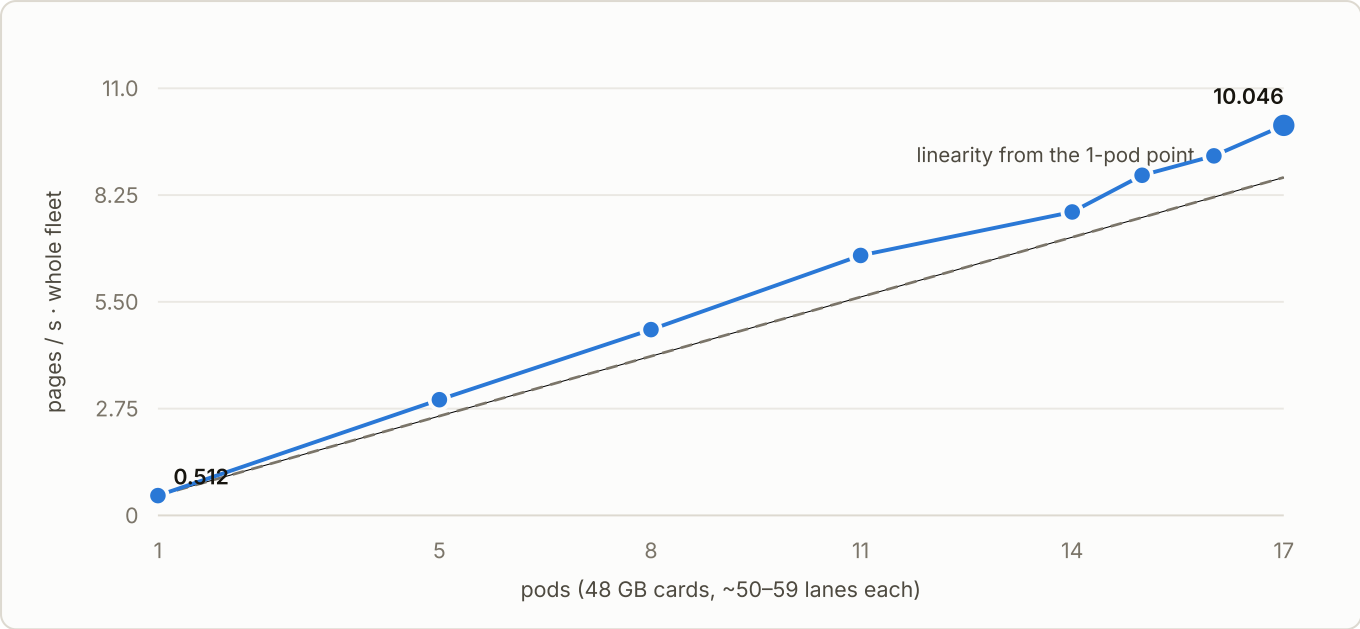
The section that must close the throughput argument is the marginal return of an added pod: if aggregate throughput grows linearly, total cost is independent of fleet size and the choice becomes purely one of elapsed time. What we have today are two points, both affected by a fleet that changed during the measurement.

RUN	PAGES	PODS	LANES / POD	PG/S	PG/S PER POD	OUTCOMES
Full pass	10,000	16 → 32	16	3.299	0.103	9,323 ok · 677 failed
Trial ramp	150	1 → 3	16	0.155	0.052	140 ok · 10 failed
One pinned pod (§5)	352	1	128	0.374	0.374	25 failed

TABLE 5 Available scaling points *as of 1 November*, kept for reference: taken at low concurrency with the fleet still growing. Superseded by the measured curve in table **5b**.

The measured curve

The ramp was taken on 2 November, inside a production pass over the 511,211-page escalation set rather than as a separate experiment, after the configuration corrections of §5. Pods were added one at a time as the provider released capacity; the controller re-converged on the lane optimum after each arrival without resetting, and each pod count closed its own measurement epoch. Values are medians over the probes of that epoch, each probe being 150 completed pages.



Chandra, 200 dpi, one page per request, lanes at the plateau found by the controller.

FIGURE 4 Aggregate throughput against fleet size. The measured points sit *on* the line of perfect linearity rather than below it: adding a pod adds its full throughput.

PODS	PROBES	PG/S FLEET	PG/S PER POD	YIELD VS LINEAR	511,211 PAGES IN
1	3	0.512	0.512	—	278 h
5	48	2.981	0.596	116 %	48 h
8	31	4.785	0.598	117 %	30 h
11	70	6.697	0.609	119 %	21 h
14	105	7.813	0.558	109 %	18 h
15	38	8.761	0.584	114 %	16 h
16	233	9.260	0.579	113 %	15 h
17	**175**	**10.046**	**0.591**	**115 %**	**14 h**

Only rows with 30 or more probes are shown, plus the 1-pod reference. Thin rows (2, 3, 4, 6, 12, 13 pods, 5–25 probes) read 0.475–0.655 and are omitted as under-sampled; the 18, 19 and 20-pod rows were taken while the controller was in the failure modes of §3 and are discarded. Mixed fleet, composition drifting as the provider released cards: at 17 pods it was 8 × L40S, 5 × RTX 6000 Ada, 4 × PRO 5000 Blackwell, 3 × L40. Yield above 100 % means the 1-pod reference, taken over only 3 probes, is slightly pessimistic.

TABLE 5B Marginal return per pod, over 700 probes. Per-pod throughput across a 17-fold change in fleet size stays within a 0.558–0.609 band — **no decay, and no trend**; the dip at 14 pods proved to be sampling, since 15, 16 and 17 pods all read higher. Fleet throughput tracks the pod count almost exactly (17 × 0.591 = 10.05 predicted, 10.046 measured). Total cost is therefore independent of fleet size, and the fleet buys only elapsed time.

Where the fleet does saturate: one forwarding worker, about twenty pods

The fleet curve above does not bend, but something else does. The orchestrator forwards OCR requests through a **single process**, and that process is the first thing in the chain with a hard ceiling — one core, whatever the host has. Measured at 14 pods and 9.3 pages/s, ten consecutive two-second samples read 4.5, 8.0, 19.0, 64.0, 73.5, 7.5, 97.5, 102.0, 102.0 and 92.5 per cent of a core: roughly half a core on average, already touching the cap in bursts. Extrapolating the average, **one forwarding worker saturates at about twenty pods.**

That ceiling is not a property of the hardware but of the process count, and it is configurable: the orchestrator will run *N* forwarding workers with its own routing in front of them (`ocr.workers`, which was 0 — meaning one in-process path — for every measurement in table **5b**, and was raised to 4 afterwards). Each worker brings its own core, so the pod ceiling multiplies.

FORWARDING WORKERS	PODS BEFORE THE HOST BINDS	FLEET PG/S	511,211 PAGES	948,156 PAG
1 (as measured)	~20	11.8	12.0 h	22.
4 (configured now)	~80	47.3	3.0 h	5.
6 (this host's practical limit)	**~120**	**70.9**	**2.0 h**	**3.**

Estimated at the measured 0.591 pages/s per pod. The host has 16 cores; the sending job and the page census a which is why 6 forwarding workers rather than 15 is the practical figure.

TABLE 5C Host-side ceiling, and how it is lifted. **The first two columns are measured and estimated respectively; the rest is arithmetic.** Per-page cost does not change — scaling is linear (table **5b**) — so what the workers buy is elapsed time: the whole escalation set in two hours instead of twelve, and the archive-wide extrapolation of §11 in under two days instead of eleven.

Two honest limits on that table. The per-worker pod figure is an extrapolation from a duty cycle, not a measurement at saturation: nothing here was run at 20 pods on one worker long enough to see it bind, because the controller faults of §3 intervened first. And the binding constraint in practice was never the host — it was **provider capacity**: for most of a night no 48 GB card of any kind was deployable in any of 52 datacentres, and the fleet reached 20 only once capacity reopened. A host able to drive 120 pods is of no use on a night when the market has none.

This closes the question §7 was opened to answer, and it also reinterprets the 0.103 of the production pass: that pass offered 16 requests in flight for the *whole* fleet, i.e. half a request per pod at 32 pods. Its collapse was starvation at the client, not contention at the orchestrator — a distinction that pass could not make and this one can, because here the pods arrived while the lane count per pod stayed fixed.

Figures for the overall duration of this work should still be read as *pod-hours*, a quantity independent of fleet size, rather than as calendar days.

STILL MISSING

The curve stops at seventeen pods: beyond that the measurements are contaminated by the controller faults of §3 and have to be re-taken. The first ceiling we would expect is the orchestrator's own admission window — 1024 requests in flight, reached at about 21 pods at 48 lanes each — and nothing yet says what happens between 11 and that bound. Note also that the 7-pod point rests on a single probe and is not reported in table 5.

8 Reading quality

Without ground truth one does not measure accuracy; one measures agreement. We do so over 10,000 pages read by all three engines, of which 8,429 succeeded for both Surya and Chandra, using three indicators that require no labels: success rate, the ratio of characters produced, and agreement on digit groups.

Who reads where the other does not

OUTCOME OVER 10,000 SHARED PAGES	PAGES	SHARE
Both succeeded	8,429	84.3 %
Chandra only	**889**	**8.9 %**
Surya only	289	2.9 %
Neither	393	3.9 %
Blank for all three (tesseract included)	278	2.8 %

TABLE 6 Cross-tabulated outcomes. The 889 pages recovered by Chandra alone are not marginal scraps: their median is 840 characters, i.e. they are full pages. The 278 blank for all three are the method's real floor, 2.8 % of the perimeter.

Where the difference is actually made

The aggregate success rate hides the interesting fact. On the two large buckets of the escalation gate the engines are nearly indistinguishable; on the small bucket — pages where the first reading produced *zero* characters, i.e. the worst scans — Chandra succeeds three times as often.

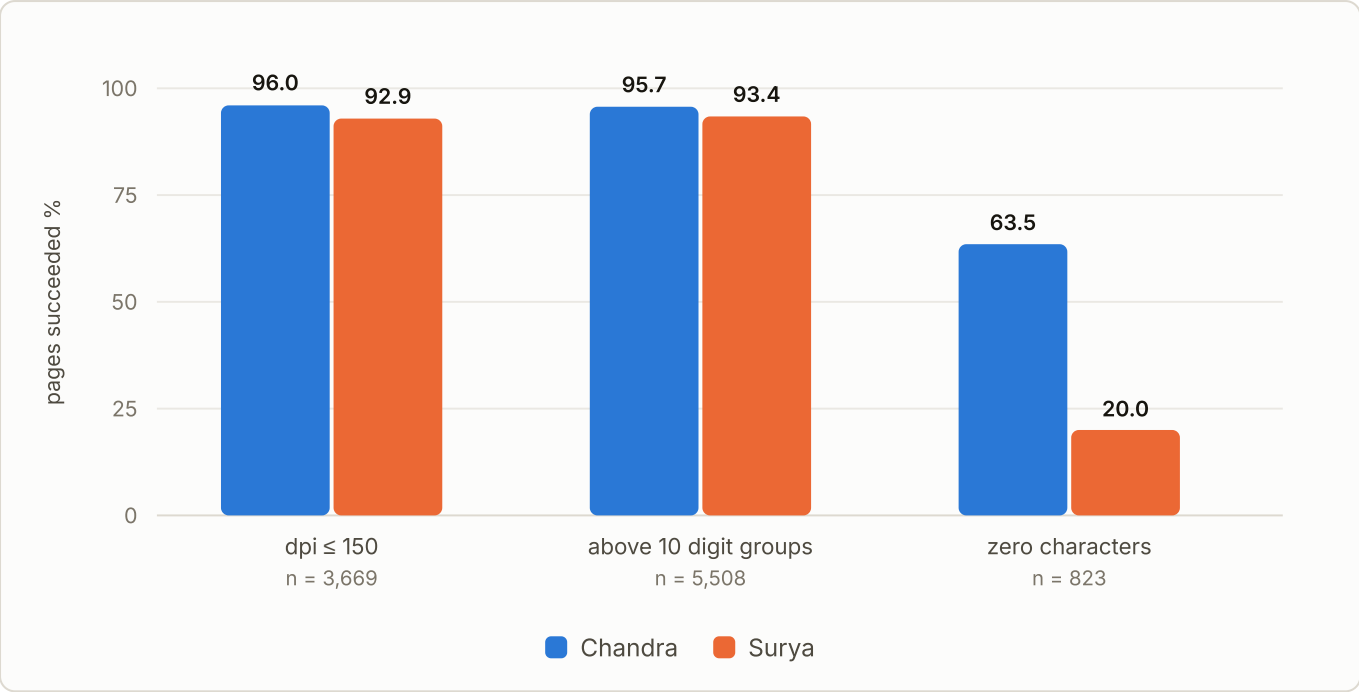


FIGURE 3 Success rate per escalation-gate bucket. Chandra's advantage is concentrated where the page is worst: 63.5 % against 20.0 % on scans that had yielded no characters at all.

Text quantity and structure

On pages all three engines read, they produce nearly identical amounts of text: the median character ratio is 1.002 for Chandra over tesseract, 0.993 for Surya over tesseract, 1.009 for Chandra over Surya. This is not a trivial result — it says that neither new engine *hallucinates* text in bulk on legible material, where the inflated mean (6.63 for Chandra over Surya) is produced by the few pages on which the other engine read almost nothing.

On structure, Surya extracts more regions (124,018 against 117,751), Chandra more lines (111,833 against 107,305), and tables come out essentially level (2,274 against 2,273) but not on the same pages: 130 pages show tables only to Chandra, 157 only to Surya. Every line either engine returns carries its own bounding box, in 100 % of cases.

9 Arbitration on digits

For the legal domain the digit is the unit that counts, and here agreement between engines can be measured strictly: extract the groups of three or more digits from each reading of the same page and compare the sets. This is the most useful result of the work, because it does not promote an engine — it defines a role.

DIGIT GROUPS, THRESHOLD 3+, OVER 8,429 PAGES	COUNT
Found by all three	88,426
Found by at least two (the keepable set)	**113,095**
By one engine only — tesseract	15,151
By one engine only — Chandra	8,535
By one engine only — Surya	7,590
Contested in Surya's favour · confirmed by Chandra	20,856 · 13,266 (63.6 %)
Contested in tesseract's favour · confirmed by Chandra	26,066 · 10,915 (41.9 %)
Contested groups settled by Chandra	**24,181 of 46,922** (51.5 %)

At the 5+ digit threshold the settled share is identical: 13,742 of 26,705, 51.5 %.

TABLE 7 Three-way arbitration. Chandra settles a little over half the disputes and *stays silent* on the rest — desirable behaviour in an arbiter, and measurable only by having three independent readings of the same page.

The two-way comparison, between Surya and the pre-existing reading, sets the frame: Surya's recall over tesseract's groups is 0.707, tesseract's over Surya's is 0.782. Neither dominates. On 321 pages the two sets are entirely disjoint while both are non-empty: these are pages where one of the two readings is wrong throughout, and they are the natural set to put in front of a person.

Hence the operational recommendation, which at first sight contradicts §8: **although Chandra is the better single engine, replacing two readings with Chandra's alone forfeits the arbitration.** The measured value is not in the engine; it is in the triad.

10 Reported confidence does not discriminate

Surya returns a per-page confidence, available on all 8,718 successful pages. It is distributed very high — mean 0.972, median 0.983, minimum 0.155 — with 1.3 % of pages below 0.80 and 4.8 % below 0.90. The problem is that it does not separate good readings from bad: the median character ratio against tesseract is 1.093 where confidence falls below 0.90 and 0.992 where it sits above 0.95. The "low-confidence" page is not a badly read page; it is a page where the engine produced *more* text than expected.

The finding is negative but operationally heavy, because it dismantles the most attractive strategy — read everything with one engine and escalate only the low-confidence pages — at least in the form in which confidence is reported today.

Chandra does report one, and it is page-scoped

Measured on 2 November over the first 23,918 pages of the production pass. The response carries `meta.confidence` with `confidence_measured: true` and, explicitly, `confidence_scope: "page"` — and that

single value is then stamped onto every line and every region of the page. **A line's reported confidence is therefore not evidence about that line**, which is the one thing not to get wrong when building an escalation rule on it.

CHANDRA PAGE CONFIDENCE, 22,960 PAGES	VALUE
Median	0.970
Deciles 1–9	0.933 · 0.952 · 0.961 · 0.966 · 0.970 · 0.973 · 0.976 · 0.979 · 0.983
Below 0.90	**1,384 pages — 6.0 %**
Below 0.80	250 pages — 1.1 %
Below 0.70	7 pages

Of 23,918 pages read, 22,960 carry a confidence; the remaining 958 are failures.

TABLE 7B Distribution of Chandra's page confidence. Narrow and high, like Surya's, but with a usable tail: a gate at 0.90 escalates 6 % of pages, one at 0.80 escalates 1 %.

So the mechanism exists and is affordable to act on. What is *not* yet shown is that it discriminates: the same negative result found for Surya could hold here, and a narrow high distribution is exactly what a confidence looks like when it is measuring fluency rather than correctness.

> MISSING MEASUREMENT
>
> Validate this confidence against labels rather than against the character ratio: roughly 2,000 of the 8,429 three-way pages have a known outcome. The decisive test is whether the 6 % below 0.90 contains a disproportionate share of the digit groups the triad settles against Chandra. If it does, single-engine reading with selective escalation becomes viable and the cost structure of §11 changes; if it does not, the triad of §9 stands.

11 Cost

Cost is expressed in pod-hours, the only form independent of fleet size, and converted at the card's hourly price. Table 8 shows the effect of correcting the concurrency parameter alone: the same work, on the same hardware, at the same quality.

SCENARIO (948,156 PAGES)	PG/S PER POD	POD-HOURS	COST AT $0.59/H
One engine, 16 lanes for the whole fleet (measured in a pass, 1 Nov)	0.103	2,557	$1,509
One engine, 128 lanes per pod (one pod, 1 Nov config)	0.374	704	$416
One engine, ~48 lanes per pod (fleet of 6, 2 Nov config)	**0.589**	**447**	**$264**
Two engines, ~48 lanes per pod	—	894	$528

Priced at the A40 SECURE rate, 48 GB, $0.59/h, for comparability. At 32 pods the third scenario occupies about 14 hours of calendar time per engine.

TABLE 8 Cost of the full perimeter. Correcting configuration alone — concurrency, send path, admission windows — is worth $1,245 per engine, with no change of card, model or quality.

The cards are interchangeable; only their price is not

The ramp of §7 ran on a mixed fleet because the provider released whatever it had. That accident produced a result worth more than the controlled comparison it replaced: **five different 48 GB cards all read at 0.52–0.60 pages per second per pod**, within each other's noise. None is meaningfully faster on this workload, so cost per page is set by the hourly price alone.

CARD, 48 GB	$/H AS RENTED	$ PER 1,000 PAGES AT 0.58 PG/S	VS CHEAPEST
A40	**0.59**	**0.28**	**—**
L40	0.74	0.35	+25 %
RTX PRO 5000 Blackwell	0.82	0.39	+39 %
RTX 6000 Ada	0.99	0.47	+68 %
RTX PRO 6000 Blackwell	1.69	0.81	+186 %

TABLE 8B Cost per page by card, at equal measured throughput. The operating rule that follows is counter-intuitive and worth stating plainly: **rent the cheapest 48 GB card available, never the fastest.** A newer architecture buys nothing here because the bottleneck is not arithmetic throughput.

Two operational findings from the same evening belong with the cost, because both cost money and neither appears in a price list. A pod can be **running and billed while unreachable**: one card ran 32 minutes with no public IP and no port mapping, which the orchestrator reports indefinitely as "provisioning" — the provider's own pod list, not the orchestrator's view, is the authority on what is being paid for. And availability, not price, is the binding constraint: for several hours no 48 GB card of any kind was deployable in any of the provider's 52 datacentres, in either its secure or its community tier, and the fleet could only be assembled by raising the hourly ceiling to admit dearer cards and then lowering it again once pods existed.

Projection over the whole dataset

At the measured 0.591 pages per second per pod, every scope of the corpus can be priced. Pod-hours are the invariant; the two money columns bracket the realistic range, the cheapest card against the mixed fleet actually rented on 2 November; the last three columns are elapsed time at three fleet sizes.

SCOPE	PAGES	POD-HOURS	$ AT 0.59/H	$ AT 0.79/H	11 PODS	32 PODS
Escalation set — the pass now running	**511,211**	**242**	**143**	**191**	**22 h**	**8 h**
All pages requiring OCR	948,156	449	265	354	41 h	14 h
Pages tesseract already clears (the saving)	446,476	211	125	167	19 h	7 h
Every page in the perimeter	1,114,432	527	311	417	48 h	16 h
Whole corpus (extrapolated, see below)	~11,000,000	5,205	3,071	4,112	473 h	163 h

Chandra stage only. Tesseract is free in cash but not in time: at 1.8 s per page it is 474 CPU-hours over the 948,156 pages, about 30 h on 16 workers, on hardware already owned.

TABLE 10 Projection by scope. The third row is the economic argument for the chain of §13: the pages the cheap CPU reader clears are worth $124–166 of card time, which is what the escalation gate is competing for.

The whole corpus, and why that row is an extrapolation

The perimeter of this work — 98,545 documents, 1,114,432 pages — is the trial-court holding with local files, not the archive. The corpus is about 11.4 million documents, of which 2,838,958 have so far been through the page-level census described in §4. That census is the only instrument that could turn "how many pages does the archive need read" into a measurement, and it is not finished; on the 36,033-document sample it has closed, 91.3 % of documents carry a usable text layer and need no engine at all.

The corpus row above therefore rests on two stated assumptions: that the ~8.7 % needing work holds archive-wide, and that those documents average the perimeter's 11.3 pages. That gives of the order of 11 million pages — ten times this perimeter — and at the cheapest card about $3,100, or 163 hours on 32 pods. **It is an order of magnitude, not a figure**: both assumptions are extrapolations from a quarter of the archive, and the second is the weaker, since documents with no text layer are plausibly also the longer ones.

Three cautions, because a projection is the easiest thing in this work to quote out of context. It covers the **Chandra stage only**: the general VLM stage — stamps, signatures, tables, figure pages, and whatever the

confidence gate escalates — is not measured at all, and on the 6 % of pages below 0.90 confidence it would add somewhere between 13 and 30 thousand pages of work whose per-page cost is unknown. It assumes **linearity continues** past the six pods actually measured. And it assumes **capacity exists**, which on the evening of 2 November it did not: for several hours no 48 GB card was deployable anywhere, and a projection in pod-hours says nothing about how long it takes to assemble the pods.

Against the $1,509 the same work would have cost at the throughput of the 1 November pass, the whole OCR of the perimeter now prices at roughly the cost of a single working day of qualified human attention — which is the real comparison, since the alternative to reading these pages automatically is not reading them.

The choice of card was verified and is not the bottleneck: in the earlier comparative run the A40 came out the most economical per document among the cards tried ($5.66 per thousand documents against $8.49 for the RTX 6000 Ada, faster but not in proportion to its price). Worth recording instead is an operational risk this work was the first to hit: the entire low-cost 48 GB tier can be out of stock at the provider for hours on end across every region, and no throughput measurement speaks to that unavailability.

CARD	GB	$/H	AVAILABILITY 1 NOV, 14:18
A40	**48**	**0.59**	**out of stock**
RTX A6000	48	0.59	out of stock
L40	48	0.82	out of stock
RTX 6000 Ada	48	0.99	out of stock
L40S	48	1.09	low
RTX PRO 6000	96	2.49	high

TABLE 9 Observed availability, twelve European regions. The only card with assured availability costs 4.2× the one chosen: the effective price of on-demand work includes the price of waiting.

12 Threats to validity

1. **No ground truth.** Every quality measurement here is a measurement of agreement. An error shared by two engines is indistinguishable from a correct reading. The human reading of 200 pages, not yet carried out, is the minimum necessary control.
2. **The truncated tail.** The 302 s limit (§6) systematically discards the slowest pages, which are also the hardest: both throughput and measured quality benefit unduly.
3. **One pod, one model.** The lane curve was measured on Chandra. Surya's curve does not yet exist, and nothing guarantees it peaks in the same place: §4 even suggests Surya responds to different levers.
4. **Mixed fleet in the scaling curve.** The points in §7 were taken on five different card models, added as the provider released them, and inside a production pass rather than a controlled experiment. The per-pod figures are therefore averages over a heterogeneous fleet; the finding that the cards are equivalent (§11) is what makes the average meaningful, and that finding rests on the same data.

5. **The curve stops at six pods.** Nothing here measures what happens at 8, 16 or 32 pods, where the orchestrator's admission window would be the first ceiling to bind.
6. **Page mix drifts within a run.** Individual probe windows vary by ±8–22 % depending on how many pages they contain, which is why every value in §7 is a median over probes and why one apparent 17 % decline, investigated, turned out to be noise: median characters per page were flat across eight consecutive blocks of 300 pages.
7. **The corpus figure is extrapolated, the perimeter figures are not.** Everything measured here concerns 1,114,432 pages. The archive-wide row of table **10** multiplies a text-layer share taken from a 36,033-document sample by a pages-per-document average taken from a different and probably atypical population. It should be read as a magnitude and nothing more.
8. **One provider, one period.** Prices and availability are those of one provider over five days in November 2026.
9. **Domain-specific perimeter.** Italian court records, largely office scans. Transferability to other document domains is not measured.

13 Provisional conclusions

First, and the one we would keep if we had to keep one: **the right concurrency should not be configured, it should be searched continuously by a closed loop**. The optimum is a property of the configuration, the fleet and the page mix — it moved by a factor of 2.7 in position and 55 % in height when the orchestrator's send path changed, and it moves again whenever a pod arrives or the pages get denser. A measured constant is therefore the wrong kind of answer: it is correct until something changes and silently wrong afterwards, because a starved fleet is indistinguishable from a busy one unless you are watching pages per second per pod. The extremum-seeking loop of §3 found every optimum in this work, held it across a fleet growing from one pod to six, and cost nothing to run because it searches while doing the real work. The corollary for anyone reporting such numbers: publish the search, not the setting.

And its corollary, learnt the hard way: **a controller that searches a queueing system is biased toward pushing harder**, because raising concurrency transiently produces the completions it is rewarded for. Any such loop needs a settling window after each move, a decaying memory of its best, and a floor it cannot persist below — without those three it will climb into a queue, collapse, and then save the collapse.

Second: **the concurrency offered to the pod is the dominant parameter of OCR throughput on a GPU**, with a measured spread of 5.7× on the same model and the same card — and the position of the optimum is itself a property of the configuration, not of the engine: it moved from 128 lanes to about 48 when the send path became asynchronous and the admission windows were widened. Anyone publishing pages per second per GPU without stating the in-flight request count is publishing an irreproducible number, and anyone sizing a job on that number pays two to three times what is necessary.

Third: **batching pages into one request is not an optimisation** on these engines; it is an order-of-magnitude degradation and a source of failures.

Fourth: **recent vision-language engines do not replace the classical engine, they arbitrate it.** Chandra reads 889 pages Surya cannot and succeeds on 63.5 % of the pages where the previous reading produced

no characters, but its largest measurable contribution is settling 51.5 % of the digit disputes between the other two readings. The value is in the triad, not in the single engine.

Fifth, and negative: **the confidence an engine reports is not usable as an escalation criterion**, at least in the form examined, because it correlates with text abundance rather than with correctness. Chandra's own confidence, now measured, is page-scoped and distributed as narrowly as Surya's; whether it discriminates is the one open test that could still change the recommendation.

Sixth: **the fleet scales linearly and the cards are interchangeable.** From one to six pods per-pod throughput does not decay, so total cost is independent of fleet size and fleet size buys only elapsed time; and five different 48 GB cards read within 0.52–0.60 pages per second per pod, so the only rational card policy is to rent the cheapest available. Taken with the first conclusion, this says the economics of OCR at this scale are decided by how the work is offered, not by which silicon receives it.

14 Measurements still missing from this preview

An explicit list of what the final version must contain and this draft does not, ordered by impact on the conclusions.

1. **Marginal return per pod** (§7): measured 1→6 pods on 2 November, linear. The 8, 12 and 16-pod points, where the orchestrator window would bind, remain open. PARTLY DONE
2. **Lane curve beyond the wall** (§6) — **no longer needed.** Hand-measuring 96/128/256 lanes answered "where is the optimum"; the controller of §3 now answers that continuously, per deployment and while the work runs, which is the better answer. What remains of §6 is not a throughput question but a loss one — whether a hard timeout still discards the slowest pages — and the per-page error records of any production pass already report it. SUPERSEDED
3. **Per-page cost of the three knobs**: resolution, continuation and token ceiling, at 128 lanes, on a fresh pod. WAITING
4. **Surya's own lane curve** (§12, point 3), plus isolating the resolution effect seen in §4. TO START
5. **Human reading of 200 pages**, stratified over the three buckets and the 321 disjoint-set pages: the only anchor to a truth. TO COMMISSION
6. **Validation of Chandra's confidence** over the roughly 2,000 pages with a known outcome (§10). The confidence itself is now measured and page-scoped; what is missing is whether it separates good readings from bad. NEXT
7. **Full pass at the lane optimum**: running since 2 November over the whole 511,211-page escalation set, which will replace every projected cost in §11 with a settled one. RUNNING
8. **Re-take the 18, 19 and 20-pod points.** Those rows were measured while the controller was in the failure modes above, so they are discarded rather than reported. TO REDO
9. **The controller on a second engine** (§3): every optimum it found is Chandra's. Whether the same loop, unchanged, finds Surya's or a general VLM's optimum is untested, and it is the claim a reader would most reasonably doubt. TO MEASURE
10. **Surya and olmOCR are no longer in the production chain** (decided 30 October, applied 2 November: `tesseract → chandra → general VLM`). Their numbers here stand as the comparison that produced that

decision; the general VLM stage — stamps, signatures, tables, figure pages — is not yet measured at all. TO MEASURE

The structure of sections 1–13 will not change: the missing measurements fill tables already drawn and can move the values, not the frame. The one conclusion a future measurement could overturn is the third, if Chandra's confidence turns out to discriminate: single-engine reading with selective escalation would then be preferable to the triad, and §9 would need rewriting as a comparison of two strategies rather than as a recommendation.

Note on the data. Every figure in this preview comes from measurement logs produced by the probes described in §3 and stored alongside the pages, page by page and without the text. The runs were carried out between 28 October and 2 November 2026 on a single internal inference infrastructure. The scaling curve of §7 and the confidence distribution of §10 were taken inside a production pass rather than a controlled experiment, and are marked as such.

Note on the controller. The lane optima in §5 and §7 were not chosen but searched, by the loop described in §3, which also recorded the probe-by-probe trace those sections are computed from.

Reproducibility. Orchestrator: CoderAI, `github.com/nextime/coderai`, images `ghcr.io/nextime/coderai` (orchestrator, no GPU) and `ghcr.io/nextime/coderai-vllm-ocr` (engine pods). Engine: `datalab-to/chandra-ocr-2` at 200 dpi, token ceiling 12288, one continuation, layout detection off. Orchestrator settings for the 2 November figures: `pod_http_async: true`, front `max_parallel_requests` 1024–4096 / `queue_max_size` 4096–8192, per pod `pod_max_parallel_requests: 192` / `pod_queue_max_size: 512`, `chandra_timeout: 1800`, `direct_tcp: true`, `cloud_types: [SECURE, COMMUNITY]`, `max_hourly_usd` 0.83 (1.70 during the capacity deadlock). Cards: 48 GB class only (`min_vram_gb: 40`). Pages are selected from the corpus by a fixed md5 seed over `doc_id` and page number, so any run reads the same pages in the same order.

Draft v1.0 — structural preview. Not citable.