Validate-Expensive-Then-Descend: Code-Gated Model Cost Reduction Across a Company-Wide LLM Skill Fleet

ThakiCloud AI Platform 2026-07-12

Abstract

Organizations that automate knowledge work with large language models (LLMs) face a recurring cost dilemma: the most capable models are the safe default for correctness, but they are 5–20x more expensive than smaller siblings, and most automated tasks do not need that capability. Static routing tables and human intuition assign tiers poorly, because "which task needs the big model" is not obvious a priori and drifts as prompts, skills, and models change. We describe **Validate-Expensive-Then-Descend (VETD)**, an operational method deployed on a single-operator automation fleet of sixteen production "skills" (units of company work, from git-work reporting to sales-CRM briefing to nightly research-paper generation). VETD first establishes a per-skill quality reference by running the task across model tiers, scores each tier's output with an expensive-model judge on fixed rubric dimensions, and then **descends to the cheapest tier that a deterministic, code-owned gate certifies as non-regressing**. The gate — not the model, and not the model's self-report — owns the accept/reject decision: a candidate tier is accepted only when its aggregate quality gap stays below a threshold *and* the judge flags zero reasoning-depth gaps and zero fixable prompt gaps. Descent that fails the gate is redirected to two other levers: fixable gaps are closed by editing the skill (recovering quality at the cheaper tier), and irreducible reasoning gaps pin the skill to the expensive tier. We report the deployed policy registry, three real descent decisions (including one flagship opus->sonnet downgrade that preserved quality through a code-owned format guardrail rather than model capability), one honest negative case that the gate correctly refused to downgrade, and a failure mode — quota exhaustion miscounted as quality failure — that briefly corrupted the escalation loop. VETD reframes cost reduction from a modeling problem into a measurement-and-gating problem: quality is held by deterministic code where possible, and the expensive model is spent on *judging descent*, not on *doing the work*.

Keywords: LLM cost optimization, model routing, model cascades, LLM-as-judge, agent skills, format determinism, operational ML.

1. Introduction

A single AI engineer now runs a fleet of autonomous LLM workflows: nightly research writing, daily stand-up digests, sales-CRM briefings, market monitors, blog evolution, code-ship pipelines. Each workflow is packaged as a *skill* — a versioned unit of company work with a prompt contract, deterministic scaffolding (scripts, templates, validation gates), and a scheduled runner. As the fleet grows, so does its recurring model bill, and that bill is dominated by a small number of workflows pinned to the most capable ("frontier") tier.

The naïve fix — run everything on the frontier model — is what the fleet started with, and it does not scale: on this deployment a single sales workflow once consumed roughly 94% of the monthly model spend because both its orchestration and its content authoring ran on the frontier tier. The opposite naïve fix — run everything on the cheapest model — silently degrades the outputs that actually need capability, and the degradation is invisible until a human notices weeks later.

The real question is per-task and empirical: **for this specific skill, with this prompt and this scaffolding, what is the cheapest model tier whose output is not meaningfully worse than the frontier tier's?** Answering it by intuition is unreliable, and answering it once is insufficient because prompts, skills, and available models all change.

We present VETD, the method this fleet uses to answer that question continuously and safely. The core commitments are:

1. **Validate expensive first.** The frontier tier's output is the quality reference. We do not assume the cheap tier is adequate; we measure the gap against the best available output.

2. **Descend under a code-owned gate.** Whether a cheaper tier is accepted is decided by deterministic code applying an explicit rubric, never by the model asserting "this looks fine." The model is confined to *producing content* and *scoring dimensions*; all accept/reject arithmetic is owned by code.
3. **Hold quality with code, not tier, where possible.** Many skills preserve output quality after descent because a deterministic post-processor owns format, counts, and validation. When quality survives descent, it is often the *guardrail* — not the model — doing the work.
4. **Route non-descendable capability, don't fake it.** When the gate refuses descent because of an irreducible reasoning gap, the skill is pinned to the expensive tier. VETD does not pretend a cheap model is adequate when the evidence says otherwise.

Our contribution is not a new routing algorithm but a **deployed operating discipline** and its evidence: the mechanism, the registry it produced, real descent/hold decisions with their measured gaps, and the failure modes we hit in production.

2. Background and Related Work

Model cascades and cost-aware routing. FrugalGPT [1] popularized cascading LLM calls from cheap to expensive with a learned scorer that decides when to stop, trading cost against accuracy. RouteLLM [2] learns a router that dispatches each query to a strong or weak model from preference data. AutoMix [3] self-verifies a small model's answer and escalates to a larger one via a meta-verifier. These systems route *per query at inference time* and typically require training a router or verifier. VETD is complementary and coarser-grained: it routes *per skill*, offline, on a schedule, and the "router" is a deterministic gate over an expensive judge's rubric scores rather than a learned model. This fits an operational setting where each skill is a stable, repeatedly-invoked workflow and the decision only needs to be revisited when the skill or model changes.

LLM-as-judge. Using a strong LLM to score another model's output correlates well with human preference on many tasks [4], and is now a standard evaluation primitive. VETD uses an expensive-model judge, but constrains it: the judge emits per-dimension scores and typed gap labels (fixable / reasoning-depth / unknown) as *data only*; it never returns the downgrade decision. This separation follows the principle that self-reported "pass" is not verification — the accept/reject arithmetic must be owned by deterministic code, which also makes the judge's output auditable and the decision reproducible.

Mixture and orchestration of models. Mixture-of-Agents [5] shows that layering multiple models can beat any single one. Our deployment uses a related but cost-motivated pattern we call the *conductor split*: an inexpensive model orchestrates (routing, deterministic scripts, aggregation) while the expensive model is reserved for the few sub-steps where content quality is the deliverable. This is descent applied at sub-task granularity rather than whole-skill granularity.

Format determinism as a quality guardrail. A recurring finding on this fleet — consistent with the broader "let code own the format" discipline — is that much of what looks like model-capability-dependent quality is actually *format* quality: correct section headers, counts, enum values, valid links, deduplication. When a deterministic post-processor owns those, output quality decouples from model tier, and descent becomes safe. VETD operationalizes this: skills whose format is code-owned are the primary descent candidates.

3. Method

3.1 Company work as a skill fleet

The unit of optimization is a *skill*: a directory containing a prompt contract (`SKILL.md`), supporting scripts and templates, and (for scheduled work) a headless runner and a launch entry. A central **policy registry** (`skill_model_policy.json`) records, per skill, its current model tier, whether it is *pinned* (never auto-changed), its consecutive-failure streak, and a human-readable reason for its tier. The registry is the fleet's routing table and the substrate VETD reads and writes. Table 1 shows the deployed registry.

At the time of writing the registry tracks 16 scheduled skills: 10 on the mid tier (`sonnet`) and 6 on the

frontier tier (`opus`); 10 are pinned and 6 are unpinned (eligible for automatic tier change). A default policy governs any unlisted skill: start on the mid tier, escalate to frontier after two consecutive bad runs.

3.2 Tier sweep (measurement)

For a target skill and a fixed input *fixture* (a representative task instance), the sweep (`gap_matrix`) runs the skill headlessly once per candidate tier — by default `haiku`, `sonnet`, `opus` — and captures each tier's raw output. The fixture is stored so the measurement is repeatable and so descent can be re-checked after a skill edit. This is the "validate expensive" step: the frontier output becomes the reference against which cheaper tiers are compared.

3.3 Rubric scoring (LLM-as-judge, data only)

An expensive-model judge (`gap_judge`, run on `opus`) scores each tier's output against the frontier reference on rubric dimensions (1–5), and emits, for the cheapest candidate, a structured gap object:

- a scalar **`headline_gap`** (aggregate quality distance from the reference),
- a list of **`fixable`** gaps (typed, e.g. *missing_example*, *enum_drift* — quality lost that a prompt/skill edit could recover),
- a list of **`reasoning`** gaps (typed *reasoning_depth* — capability the cheaper tier structurally lacks),
- a list of **`unknown`** gaps.

Crucially the judge returns *these labels and scores only.* It does not decide whether to downgrade.

3.4 The deterministic descent gate

The accept/reject decision is owned by code (`gap_report` / `cost_evolve`). A candidate cheaper tier is accepted for descent **iff all three hold**:

```
headline_gap <= max_gap            (default max_gap = 0.75)
AND |reasoning gaps| == 0
AND |fixable  gaps| == 0
```

The three-way outcome:

- **Downgrade** — all conditions hold: the skill can run on the cheaper tier now. On `--apply`, code updates the registry (`skill_retro de-escalate`).
- **Fix-then-retry** — `headline_gap` acceptable but *fixable* gaps present: the lost quality is recoverable by editing the skill (add a worked example, pin an enum). After the edit, re-sweep; if the gap closes, descend.
- **Hold/pin** — a *reasoning* gap is present, or `headline_gap > max_gap`: the cheaper tier is structurally inadequate; keep (or pin) the expensive tier.

A self-check with synthesized inputs proves the gate wiring without any model call; on the deployed system it passes all cases (e.g. `headline_gap 0.2 <= 0.75, no reasoning gap -> downgrade`; `1 reasoning gap -> hold`; `headline_gap 1.4 > 0.75 -> hold`). Figure 1 shows the decision flow.

Figure 1. The VETD loop. The expensive model is spent only on *judging* (rubric scoring, data-only); the accept/reject decision is owned by deterministic code, which routes each skill to one of three outcomes.

3.5 De-escalation, escalation, and pinning

Descent writes the registry only through `skill_retro`, which enforces safety invariants: **de-escalation (making a skill cheaper) is manual/gated and never automatic** — a skill that ever needed the frontier tier is not silently cheapened; a *pinned* skill is never auto-changed; and de-escalation requires a zero failure streak. Conversely, unpinned skills *escalate* automatically: two consecutive bad runs (non-zero exit, or a hard-failure marker such as an auth error or traceback in the run log) bump the streak and promote the skill to the frontier tier; a clean run resets the streak. VETD is thus asymmetric by design — cheap to try descending, conservative about staying descended, automatic about recovering when a descent proves wrong.

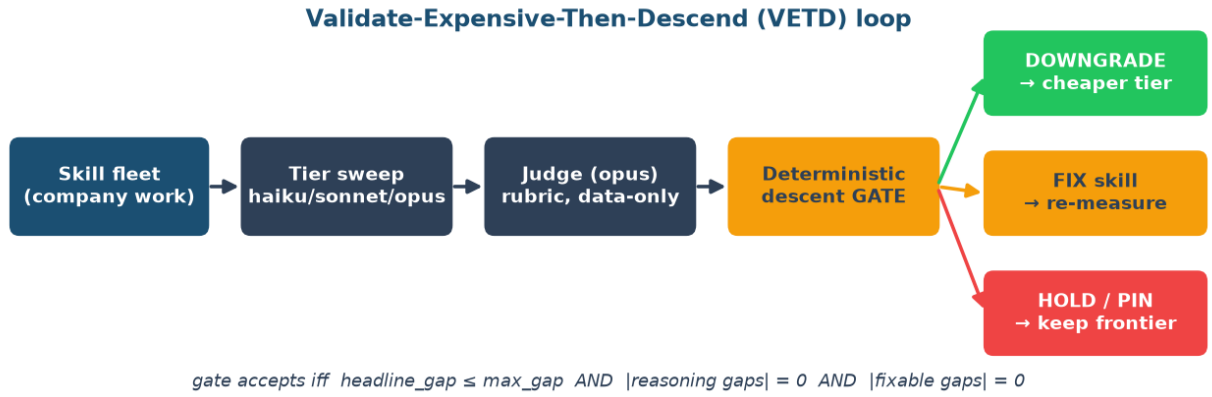

Figure 1: Figure 1

3.6 Guardrail: why quality survives descent

Descent is only safe if the cheaper tier's *content* is adequate; the *format* must be held elsewhere. The fleet's flagship descent (Section 4.2) survives precisely because a post-processor computes and enforces the quality-critical properties (length thresholds, source counts, an AI-writing gate, deduplication, canonical headers) in deterministic code, re-dispatching any worker output that fails. The model composes content; code owns the gate. This is the general precondition VETD looks for: **skills whose measurable quality is format-dominated are the safe descent candidates**, and making format code-owned is itself the enabling refactor.

4. Deployment and Measurements

We report the deployed registry and four real cases. All figures are from this fleet; no numbers are synthetic.

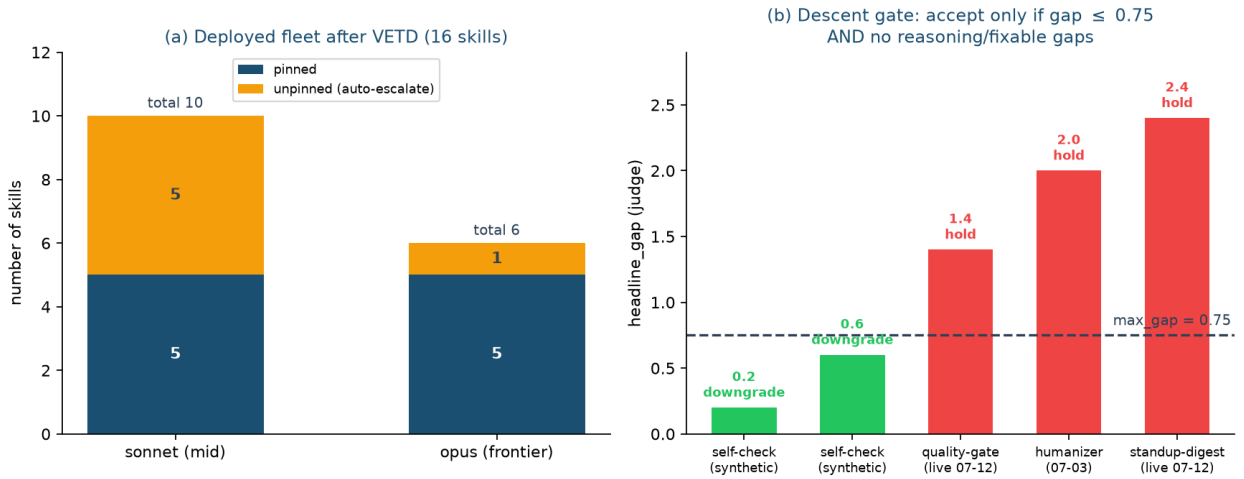

Figure 2: Figure 2

Figure 2. (a) The 16-skill fleet after applying VETD: 10 skills on the mid tier, 6 on the frontier tier; pinned skills are protected from automatic change. (b) Measured `headline_gap` against the `max_gap = 0.75` threshold. Green = accepted for downgrade; red = held. The two live sweeps (`quality-gate`, `standup-digest`, 2026-07-12) and the earlier `humanizer` measurement were all correctly held; only sub-threshold, gap-free cases descend.

4.1 The fleet registry (Table 1)

Skill	Tier	Pinned	Note (abbreviated)
twitter-timeline-to-slack	sonnet	yes	Downgraded opus->sonnet; quality held by format-determinism gate
sales-crm-orchestrator	sonnet	yes	Conductor split: sonnet main, opus content sub-agents
sod-ship	sonnet	yes	Orchestration; de-escalated 2026-07-07
eod-ship	sonnet	yes	Orchestration; de-escalated 2026-07-07
daily-scrum-notion	sonnet	yes	Orchestration; format code-owned
content-actionizer	sonnet	no	Additive + test-gated; sonnet suffices
group-meeting-digest	sonnet	no	Content output; escalate-on-repeat (streak 1)
ai-platform-autoqa	sonnet	no	Orchestration; anomaly-gated
nightly-lab	sonnet	no	Experiment orchestration; exec code-owned
daily-tool-ideas	sonnet	no	Format code-owned
tech-blog-evolve	opus	yes	Blog quality evolution is the deliverable
news-comic-strip	opus	yes	Humor/wit is the deliverable
bespin-news-notion	opus	yes	Editorial curation is the product
bespin-news-blog	opus	yes	Creative angle must not read templated
nightly-paper-factory	opus	yes	Paper quality is the deliverable
daily-ebook	opus	no	Escalated 2026-07-11 after 2 bad sonnet runs

The registry is itself a result: after applying VETD, 10 of 16 skills run on the mid tier. The 6 that remain on the frontier tier are exactly those whose *content* — not format — is the deliverable (humor, editorial judgment, research prose, creative angle). This is the pattern VETD surfaces: **orchestration and format-dominated work descends; irreducible content-quality work stays.**

4.2 Flagship descent: twitter-timeline-to-slack (opus -> sonnet)

This skill posts a daily digest of curated timeline items. It was pinned to the frontier tier and, at five fires per day, was a top quota consumer. VETD's precondition — format-dominated quality — was made true by a refactor: per-item workers now emit *content only*, and a deterministic validator (`tweet_validate`) code-computes the quality gate (minimum body length, minimum source count, a web-search-evidence count, an AI-writing gate) and re-dispatches any failing item; a separate poster owns headers, formatting, and deduplication. With format owned by code, the skill was downgraded opus->sonnet. The reported rationale is

explicit that **quality is preserved "NOT by model tier but by the format-determinism guardrail."** The skill is pinned at the cheaper tier so a transient failure cannot silently re-escalate and undo the saving. Blog authoring inside the same runner was *decoupled* and kept on the frontier tier via a separate model variable — descent at sub-step granularity.

4.3 Conductor split: sales-crm-orchestrator

Before VETD this workflow ran entirely on the frontier tier and accounted for roughly 94% of monthly spend. Rather than a whole-skill downgrade, it was split: the **main conductor** (orchestration, deterministic scripts, news matching) descended to the mid tier, while **customer- and deal-facing content authoring** (briefings, deliverable research, RFP interpretation, proposal architecture) stayed on the frontier tier via dedicated sub-agents. The bulk cost (orchestration) descends; the thin high-value slice (prose) does not. This is VETD applied at sub-task granularity and is the pattern for skills that mix cheap coordination with expensive judgment.

4.4 Honest negative case: humanizer (gate refuses descent)

Not every skill descends. The `humanizer` skill (removes AI-writing tells from Korean text) was measured with `headline_gap = 2.0` — well above the `0.75` threshold — with three *fixable* gaps (missing before/after examples for specific AI tells, sentence-ending variation, register-consistency enum drift) and one *reasoning* gap (`rephrasing_naturalness`). The gate's verdict is **hold**: the reasoning gap alone forbids downgrade, and the headline gap independently exceeds threshold. VETD's correct output here is *not* a downgrade but a **work order**: close the three fixable gaps by editing the skill, then re-measure; the reasoning gap keeps it on the higher tier until the edits prove the cheaper tier adequate. This case matters because a system that only ever downgrades is broken — the gate must be willing to say no.

4.5 Adjacent lever and a real failure mode

VETD sits alongside a second, non-model cost lever the same machinery applies: **MCP-server pruning**. On 2026-06-27 the evaluator flagged five connected tool-servers with *0 calls across 40 sessions*, each with a working fallback, and marked them for reversible removal — each idle server otherwise costs roughly a thousand tokens of schema per turn. Same discipline (measure usage, gate on a reversible criterion, apply), different lever.

The instructive failure came from the escalation half of the loop. In early July, several skills (`sod-ship`, `eod-ship`) auto-escalated to the frontier tier after "bad runs" that were in fact **weekly-quota exhaustion, not quality failures** — the runs finished correctly but were miscounted because the run log contained a limit marker. The fix corrected the failure classifier to treat quota/auth exhaustion as *neutral* (no streak bump), and the affected skills were manually de-escalated on 2026-07-07 with the root cause recorded. The lesson generalizes: **the signal that drives (de-)escalation must distinguish capability failure from availability failure**, or the loop optimizes against noise.

4.6 Live end-to-end run

To exercise the loop end-to-end at write time we invoked the descent measurement live on a fixtured skill. The `quality-gate` skill (a report-validation workflow) was swept and judged on the frontier tier, returning `headline_gap = 1.4` with **two reasoning gaps** (`date_weekday_factual_accuracy`, `numeric_verification_depth`) and three format gaps. The gate's verdict was **hold**, with the code-emitted recommendation to *"programmatize the format gaps first"* — i.e. move the three format-dominated gaps into deterministic code (the Section 3.6 guardrail) and only then re-measure whether the residual reasoning gap still blocks descent. A second live sweep the same night, on `standup-digest`, returned `headline_gap = 2.4` with two reasoning gaps (`factual_grounding`, `analytical_depth`) and the same **hold** verdict and programmatize-first recommendation. Both fresh measurements were refusals: the gate did not rubber-stamp either downgrade, which is the behavior a cost-cutting system most needs and most easily loses. These are fresh, unedited examples of VETD refusing unsafe downgrades and instead emitting concrete work orders. The gate self-check also passed all synthetic cases without any model call, confirming wiring independently of

the judge. (A first live attempt failed on a relative fixture path — a mundane but real operational gotcha: headless sub-runs change working directory, so fixtures must be passed as absolute paths.)

5. Limitations

Fixture coverage bounds the claim. VETD only measures skills that have a stored representative fixture; "company-wide" in practice means "the fixtured subset," which grows as fixtures are added. Skills without a fixture are governed only by the escalation half of the loop (react to failures), not the descent half. Honest scope: this is a method and a partial deployment, not a proof that every workflow has been cost-optimized.

Judge cost and judge bias. The judge runs on the frontier tier, so measurement is not free; VETD amortizes it by running per-skill and only when the skill or model changes, not per production call. LLM-as-judge also inherits known biases; the deterministic gate and typed gap labels reduce, but do not eliminate, judge subjectivity in the `headline_gap` scalar.

Single fixture != distribution. A skill judged descendable on one fixture may regress on inputs the fixture does not represent. Multi-fixture sweeps and periodic re-measurement mitigate this but are not yet exhaustive.

Non-stationarity. Prompts, skills, and model versions drift; a descent valid today can become invalid after an upstream model change. The registry records reasons and the escalation loop catches regressions, but continuous re-validation is required, not a one-time pass.

Negative results are cheap to hide. A system incentivized to cut cost will bias toward downgrading. The `humanizer` hold shows the gate resisting that pressure, but the safeguard is procedural (the code-owned gate), not adversarial; a stronger design would add an independent refutation step before committing a downgrade.

6. Conclusion

VETD treats model-cost reduction as a measurement-and-gating problem rather than a modeling problem. By validating against the expensive tier first, confining the expensive model to *judging* rather than *doing*, and letting deterministic code own both the accept/reject gate and (where possible) the output format, a single operator descended the majority of a 16-skill fleet to a mid tier while keeping the few genuinely capability-bound workflows on the frontier tier — and caught the one time the loop mistook a quota outage for a quality failure. The method's honest boundary is fixture coverage and non-stationarity: it is a continuous discipline, not a one-shot optimization. Its transferable core is a single reframing — **spend your best model on deciding where you no longer need it.**

References

[1] L. Chen, M. Zaharia, J. Zou. "FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance." arXiv:2305.05176, 2023.

[2] I. Ong, A. Almahairi, V. Wu, et al. "RouteLLM: Learning to Route LLMs with Preference Data." arXiv:2406.18665, 2024.

[3] A. Madaan, P. Aggarwal, A. Anand, et al. "AutoMix: Automatically Mixing Language Models." arXiv:2310.12963, 2023.

[4] L. Zheng, W.-L. Chiang, Y. Sheng, et al. "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena." arXiv:2306.05685, 2023.

[5] J. Wang, J. Wang, B. Athiwaratkun, et al. "Mixture-of-Agents Enhances Large Language Model Capabilities." arXiv:2406.04692, 2024.