TECHNICAL REPORT · AVZ-TR-2026-01 · REVISION V4

Learning to Route: A Shadow-First Contextual Bandit for Per-Request LLM Model Selection

Avriz Engineering · avriz.io · August 2026 · control-plane/rl_router.py

ABSTRACT

Serving every request with a frontier language model wastes an order of magnitude of cost on work a small model handles; serving every request with a small model fails the work that matters. Production LLM platforms therefore route: they pick a model *per request*. We describe the routing system deployed on Avriz, a managed development-box platform, in three layers: (i) a hand-built **LLM judge** that classifies each turn's difficulty onto a five-rung model ladder, with a mid-turn escalation valve; (ii) a **shadow-mode contextual bandit** that learns per-rung reward models online from the platform's existing billing ledger, without storing chat content and without affecting user traffic; and (iii) a flag-gated **ε-greedy serving path** that lets the learned policy earn real traffic incrementally, bounded by a rung ceiling, a deterministic constraint pre-filter and a plan-tier arm set, and measured by an on-policy A/B comparison. The design premise throughout is that *observation precedes influence*: the learner trains off-policy from decisions the judge already makes, activates only past an explicit data bar, and every step toward autonomy is one configuration flag from rollback. We give the full reward construction, the per-arm estimator and its update rule, the activation and serving math, an economics-fingerprint mechanism that resets a rung's learned weights when the model or price behind it changes, and the honest limitations of direct-method off-policy evaluation at our scale.

1 Introduction

Avriz gives each subscriber an isolated cloud machine with a coding agent on it. Every conversational turn the agent takes is ultimately a call to some large language model, and the platform's managed tier lets it choose *which*. The price spread across the available ladder is large — wholesale output-token cost spans 12× from the cheapest rung to the most expensive (Fig. 2) — while a majority of real turns ("rename this", "what does this function do?") are handled perfectly by the cheapest rung. Routing is thus a direct lever on both margin and user experience: under-routing a hard request produces a bad answer; over-routing an easy one burns money.

The first version of a router is always a heuristic, and heuristics rot: thresholds hand-tuned in June are silently wrong by September as models, prices and usage change. The interesting question is not "can we write a good routing rule" but "can the platform *keep* the rule good, from its own traffic, without ever risking that traffic". This report documents our answer, which is deliberately conservative: a learning system that spends its whole early life in shadow.

Three properties drove the design. **Zero-risk learning**: the learner must not change a single user-visible decision until a human flips a flag — and even then only for a bounded slice of traffic under a rung ceiling. **Zero new data**: training must use only what the platform already records for billing (token counts, realized cost, status codes) plus derived, content-free features; no chat text is stored by, or recoverable from, the routing system. **Legibility**: every stage — features, prior, reward, estimator, activation — is inspectable in an admin report, and the model itself is a set of eleven-dimensional linear weights a human can read.

2 Background and related work

Selecting one action per context and observing a reward only for the chosen action is the *contextual bandit* problem, the one-step special case of reinforcement learning [8]. The linear-payoff formulation and its optimistic variant LinUCB were established for news recommendation by Li et al. [3]; posterior-sampling alternatives descend from Thompson [1] (see [7] for a modern tutorial), and confidence-based exploration from UCB [2]. Evaluating a *new* policy from logs generated by an *old* one is the off-policy problem: the direct method (fit a reward model, evaluate the candidate against it), inverse-propensity weighting, and their doubly-robust combination are analyzed by Dudík et al. [4]; Bottou et al. [5] treat the same machinery as counterfactual reasoning about a production system, which is exactly our setting. Practical system design for production bandits — logging, delayed rewards, gating — is discussed in Agarwal et al. [6] and evaluated at scale in [9]. Cost-aware routing across an LLM cascade was introduced as FrugalGPT [10]; learned routing between model pairs from preference data appears in RouteLLM [11]. Our contribution is not algorithmic novelty — the estimator is deliberately the simplest thing that can learn — but an end-to-end production recipe: reward built from a billing ledger, off-policy training from a heuristic logging policy, an explicit activation bar, and a ceiling-bounded ε-greedy graduation path, all inside a system where routing failures self-heal.

3 The live router: judge, ladder, escalation

The unit of decision is a **turn**: one human message and the burst of agent LLM calls it triggers. The platform's model ladder is ordered easy→hard:

$$\mathcal{A} = \langle \texttt{gemini}, \texttt{fugu}, \texttt{sonnet}, \texttt{opus}, \texttt{f-ultra} \rangle, \quad |\mathcal{A}| = 5 \tag{1}$$

The ordering is by *capability tier*, not by price: the ladder is the difficulty tiers of the judge in order, and price is not monotone along it (rung 1, `fugu`, is dearer per output token than rung 2, `sonnet` — Fig. 2). Every index in this report — the prior's rounding target in eq. (5), the escalation step in eq. (3), the serving ceiling in eq. (12), and both axes of the

confusion matrix — is an index into *this* order.

On each fresh turn, a **judge** — itself the cheapest capable model, prompted to answer with a single word — maps the message to a difficulty tier, and the tier to a rung:

$$g : \text{text} \to \{\text{trivial}, \ldots, \text{expert}\} \to a_0 \in \mathcal{A} \tag{2}$$

The judge's failure mode is legible from its own construction: it sees only the message *text*. A terse "fix the login bug" grades `trivial` and pins a fifty-step debugging session to the weakest model. Two mechanisms bound the damage. First, follow-up calls within a turn reuse the turn's decision (a 10-minute memo), so the judge fires once per human message, not per call. Second, an **escalation valve**: when a turn's call count crosses fixed checkpoints, the rung is bumped to the next *available* rung above it (an unfunded or unreachable rung is skipped, so a bump may move more than one index) and stops at the top of the ladder:

$$c \in \{12, 48\} : \; a \leftarrow \min\{a' \in \mathcal{A}_{\text{avail}} : a' > a\} \tag{3}$$

taking a itself when that set is empty. Escalations are the router confessing its own mistake in-flight, which makes them a precious training signal (§5). Users who pin a model bypass all of this; bring-your-own-key boxes never enter the managed path at all.

4 Problem formulation

Routing is a contextual bandit. At turn t the platform observes a context x_t (derived from the message and light tenant state), chooses an arm $a_t \in \mathcal{A}$, and receives a scalar reward R_t observable *only for the chosen arm* — the counterfactual "what would `opus` have done with this turn" is never seen. The goal is a policy π maximizing $\mathbb{E}[R]$.

4.1 Context: an eleven-dimensional, content-free feature map

The feature extractor is pure and cheap — no model call, no stored text. From the last user message m and tenant context it computes counts and flags, then scales them to roughly unit range with capped ratios (order is frozen; new features append):

$$\begin{aligned}\phi(m, \text{ctx}) = [\; & 1,\ \text{n\_words}/60,\ \text{n\_chars}/400,\\ & \text{code\_fence},\ \text{traceback},\ \text{n\_paths}/3,\\ & \text{hard\_verbs}/3,\ \text{easy\_verbs}/2,\ \text{question},\\ & \text{prior\_rung}/4,\ \text{esc\_rate}\;] \in \mathbb{R}^{11}\end{aligned} \tag{4}$$

The signals are chosen to see exactly what the text-only judge cannot: pasted code fences, traceback fingerprints, referenced file paths, hard verbs ("refactor", "migrate", "deadlock") versus easy verbs ("rename", "typo"), the rung the conversation was already on, and the tenant's recent escalation rate — a running confession of how often this tenant's turns get under-routed.

4.2 The prior: a transparent difficulty scorer

Before any learning, a hand-set linear scorer maps features to a difficulty in $[0, 4]$, aligned to ladder indices — long asks trend harder, code and tracebacks mean real work, easy verbs pull down, a conversation may step down at most one rung between turns:

$$s(x) = \text{clip}(u \cdot x,\ 0,\ 4), \qquad a_{\text{prior}} = \mathcal{A}[\,\text{round}(s(x))\,] \tag{5}$$

This scorer is the candidate policy's **prior and cold-start**: until the bandit clears its data bar (§5.3), the shadow decision *is* the prior, so early shadow behavior is deterministic and auditable.

4.3 Reward from the billing ledger

No new telemetry is collected. A turn's reward is assembled after the fact from `llm_calls` rows the platform already writes for metering — attributed to the decision by a 600-second turn window, bounded by the tenant's next decision, and scored only after a 120-second settling delay:

$$R = 1 \cdot \mathbf{1}[\text{success}] - 0.5\,\text{cost}_{¢} - 0.4 \cdot \mathbf{1}[\text{esc}] - 1.0 \cdot \mathbf{1}[\text{fail}] \tag{6}$$

Success means at least one usable served completion. Cost enters in *cents* so its magnitude is commensurate with the indicator terms on typical turns. Soft, self-healing routing events (provider exhaustion, empty completions, auth rolls) are excluded from failure — they are the platform routing *around* trouble, not the decision being wrong. A mid-turn escalation, by contrast, is charged to the decision: it is a non-counterfactual, in-production measurement of the judge's own mistake, and it is the single most valuable bit in the dataset.

5 The learner: per-arm linear models, trained online off-policy

5.1 Estimator

Each rung a owns a linear model of estimated reward:

$$r_a(x) = w_a \cdot x, \qquad w_a \in \mathbb{R}^{11} \tag{7}$$

This is the regression / *direct method* contextual bandit — LinUCB [3] without the per-arm confidence matrix. The simplification is deliberate: at our data scale a full covariance estimate is noise, while eleven readable weights per arm are something an operator can audit in the admin panel.

5.2 Update rule

Weights train online as rewards land in the backfill pass, one SGD step of ridge regression per scored turn, on the arm that was *actually served* — the judge's pick in shadow mode, and, once serving is on (§6), the bandit's own pick on the turns it served:

$$w_a \leftarrow w_a - \eta\,[(w_a \cdot x - R)\,x + \lambda w_a], \;\; \eta = 0.05,\; \lambda = 10^{-3} \tag{8}$$

i.e. per-sample gradient descent on $\frac{1}{2}(w \cdot x - R)^2 + \frac{1}{2}\lambda\|w\|_2^2$. Training is **off-policy by construction** while the system is in shadow: arm a's model learns only from turns that genuinely landed on a, whichever policy put them there. There is no importance weighting — the logging policy is a deterministic heuristic with no propensities to invert — which is precisely why the system graduates to *measured* on-policy comparison (§6) rather than trusting counterfactual estimates.

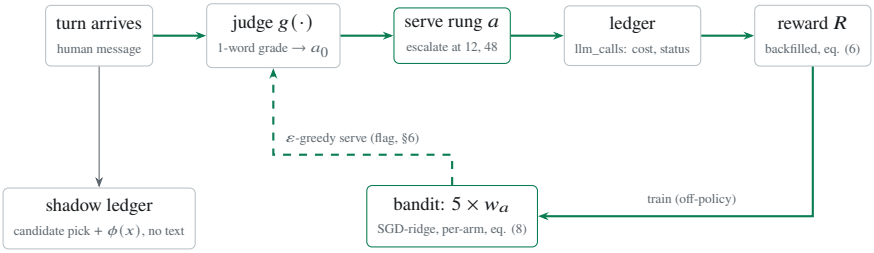

Figure 1: The learning loop. Solid arrows run in production today: every graded turn flows judge → serve → ledger → reward → bandit training. The dashed arrow is the flag-gated serving path (§6): a configured fraction of turns bypasses the judge for the trained policy. The shadow ledger stores the candidate's pick and a content-free feature vector — never chat text.

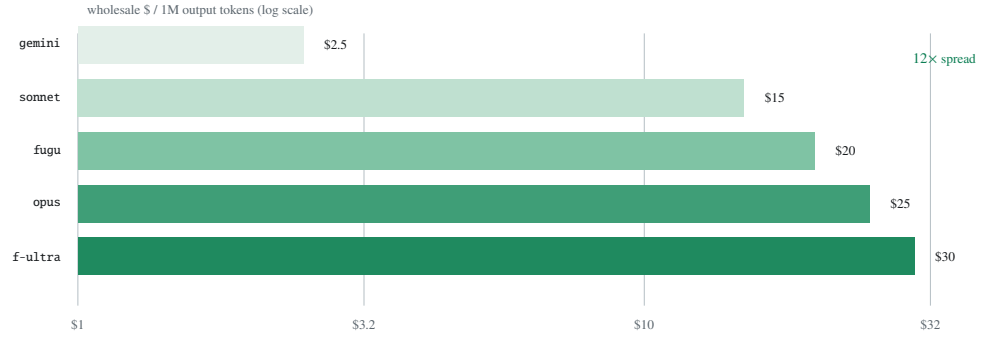

Figure 2: Why routing pays. Wholesale output-token cost per ladder rung (log scale; bars sorted by *price*, which is not the ladder order of eq. (1) — `fugu` is rung 1 but the third-dearest rung). A mis-route to the top rung costs 12× the bottom one; the judge's job is to spend that multiplier only when the turn earns it. Input-token rates show the same ordering.

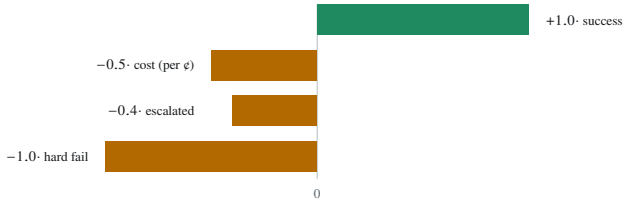

Figure 3: Reward decomposition, eq. (6). One positive outcome term and three penalties, drawn at their weights. Cost is continuous (its bar shows the weight per cent of realized spend); the others are indicators. Weights are caller-overridable in the report so the balance can be re-tuned when data argues for it.

5.3 Activation: the data bar

The learned policy is inert until it has coverage. With n_a the training count on arm a:

$$\text{active} \iff \textstyle\sum_a n_a \geq 300 \;\wedge\; |\{a : n_a \geq 30\}| \geq 2 \qquad (9)$$

and its decision considers only well-sampled arms, with an optional optimism bonus held at zero in shadow:

$$\pi^*(x) = \underset{a\,:\,n_a \geq 30}{\arg\max} \left[r_a(x) + c/\sqrt{n_a + 1} \right], \quad c = 0 \qquad (10)$$

Below the bar, the candidate is exactly the prior (5); above it, the bandit overrides. The admin report states which regime is live — *warming up (heuristic prior)* vs *bandit driving* — with per-arm counts, so there is never ambiguity about what produced a shadow pick.

5.4 Forgetting: resetting an arm when its model changes

An arm's weights encode "what reward this rung earns on requests like x" under a *specific* underlying model at a *specific* price. Two events break that premise, and they break it differently. A **price change** (same model, cheaper) is largely self-correcting: the cost term in eq. (6) is recomputed at backfill time from the actual ledger rows, so post-change turns already carry the new economics into R — only the fitted w_a lags, and because the update (8) is *constant*-step-size SGD (not a decaying rate that would freeze), it never stops tracking; its effective memory is $\sim 1/\eta \approx 20$ samples *on that arm*. The uncomfortable case is a **model swap behind a fixed rung id** (Sonnet 5 → 6, or a repointed pool): the arm keeps its weights *and* its large n, so its history is not merely stale but actively wrong, and the accumulated count suppresses the very re-exploration that would fix it.

We make that a first-class, operator-free event with an **economics fingerprint** per rung:

$$f_a = \text{sha256}(\text{model-id} \,\|\, \text{base-url} \,\|\, \text{price}_{\text{in}} \,\|\, \text{price}_{\text{out}}) \qquad (11)$$

The fingerprint is stored beside each arm's weights and re-checked on every backfill. When it flips — any of model, endpoint, or price changed — that arm is **reset** ($w_a \leftarrow 0$, $n_a \leftarrow 0$) and re-stamped, so it drops back through the difficulty prior (5) and must re-earn its way past the activation bar (9) on *current* data. The reset is surgical — only the changed rung forgets; well-sampled neighbors keep their history. An operator "reset arm" control does the same on demand, for a knowing swap you don't want to wait on. This is deliberately hard forgetting rather than a global decay factor: a decay rate

slow enough to preserve a stable rung's hard-won signal is too slow to erase a swapped model's, whereas the fingerprint erases exactly the arm that changed, exactly when it changes.

6 Serving: earning traffic ε-greedily under a ceiling

Shadow evaluation is necessary but not sufficient: direct-method estimates share the reward model's bias [4]. The graduation path therefore serves the learned policy on a small, configured slice of live traffic and *measures* the comparison. With `pct` $\in [0, 100]$ an operator flag (default 0) and a_{ceil} a rung ceiling:

$$a_{\text{served}} = \begin{cases} \Pi_C[\min(\pi^*(x),\, a_{\text{ceil}})] & \text{w.p. } \varepsilon, \text{ if active} \\ \Pi_C[g(\text{text})] & \text{otherwise} \end{cases} \tag{12}$$

Three properties make this safe to turn on. It is a **hard no-op below the data bar** — deploying the code changes nothing until an admin moves the flag. The **ceiling bounds the blast radius**: an over-eager policy cannot jump traffic to the top rung while trust is being built. And the platform's existing self-healing sits underneath: if a served rung is unavailable or unfunded, the standard fallback ladder catches it, exactly as for judge picks.

6.1 The admissible set C, and what else moves the served rung

Neither policy chooses from the bare ladder. A deterministic pre-filter — evaluated *before* the judge grades or the bandit picks — narrows $\mathcal{A}$ to an admissible set C, and Π_C in eq. (12) snaps a pick that fell outside C onto the nearest admissible rung (preferring the rung *above* on ties, since availability beats a latency promise). C is the intersection of three things.

Per-request constraint rails. A caller may demand a latency bound or a floor rung via request headers; the latency filter drops a rung only on *probe evidence* that it is slower than the bound (an arm with no measured latency yet is kept — unknown is not slow). An unsatisfiable constraint set is discarded entirely rather than routing to nothing: availability beats the promise.

Plan tier. A tenant's plan defines which rungs its turns may use at all; past the free tier's premium budget the arm set narrows to the fast rungs rather than switching the agent off. The judge, the bandit and the shadow log all carry on against the smaller set.

Rung availability. An unfunded or unreachable rung is snapped up to the next available one, and any turn carrying tool definitions is routed off the bottom rung outright — the Gemini wire format cannot round-trip a multi-turn tool conversation in the OpenAI shape our agents speak.

These rails are constraints, not learning: they are deliberately outside the reward loop. But they shape the data it sees, and the paper's own numbers inherit that (§8). When the bandit's pick survives to be served, the serving path also records the policy's **confidence**, a logistic squash of the gap between the best and second-best predicted value among eligible arms, so the admin report can slice measured outcomes by how sure the policy was.

Turns the bandit serves are tagged (`live_why='bandit'`), which closes the loop twice over: their rewards are *measured* rather than modeled, and those measured rewards train the policy on its own decisions — the system's first on-policy data. The admin report then renders an A/B block comparing bandit-served vs judge-served reward, success and cost head-to-head; widening ε is justified by that measured table, not by counterfactual estimates.

7 Evaluation methodology

The admin report backfills rewards on demand and renders, in order of how much we trust each number.

Agreement and confusion. Judge-vs-candidate agreement rate and the full 5×5 confusion matrix (Fig. 7). Disagreement mass concentrated one rung *above* the judge on turns that later escalated is the signature of the failure mode we built this to catch.

Escalation catch-rate. Among turns the judge under-routed badly enough to trip eq. (3), the fraction where the candidate had started higher. This is a *non-counterfactual* win: the escalation actually happened, in production, under the judge.

Reward on the agreement subset. Realized mean reward where both policies chose the same rung — a sanity check that the reward signal itself is stable before any delta between policies is read.

Counterfactual cost estimate — labeled an estimate. Where picks diverge, the candidate's rung was never served; its cost is modeled from that rung's realized per-turn mean. This is the direct method of [4] and inherits its bias; the report says so in the UI, and the number is treated as a screen, not a verdict. The verdict comes from the measured on-policy A/B of §6.

8 Privacy and safety properties

The routing system stores, per decision: a timestamp, the tenant id, the two model picks and why the served one was chosen, the raw feature counts behind eq. (4) — they are scaled into the unit-range vector at training time, not at write time — a turn key, and the outcome scalars of eq. (6). It stores no message text, and the features are deliberately non-invertible summaries (counts, capped ratios, binary flags).

The turn key deserves naming rather than burying: it is a truncated SHA-256 of the user message, used to tell one turn from the next and to bound reward attribution. It is neither plaintext nor reversible, but it is *content-derived*, so it admits confirmation of a guess — someone holding both the database and a candidate message can test whether that exact message was sent. We state that rather than claim more than the mechanism delivers; a per-tenant keyed digest would close it and costs nothing but a migration.

Users who pin a model in settings short-circuit the resolver before either the judge or the bandit is consulted; BYO-key boxes never enter the managed path; utility calls stay pinned to the cheapest funded rung. The learner is fleet-shared — every tenant's turns pool into one policy — which is both the cold-start solution (a single tenant would never clear eq. (9)) and

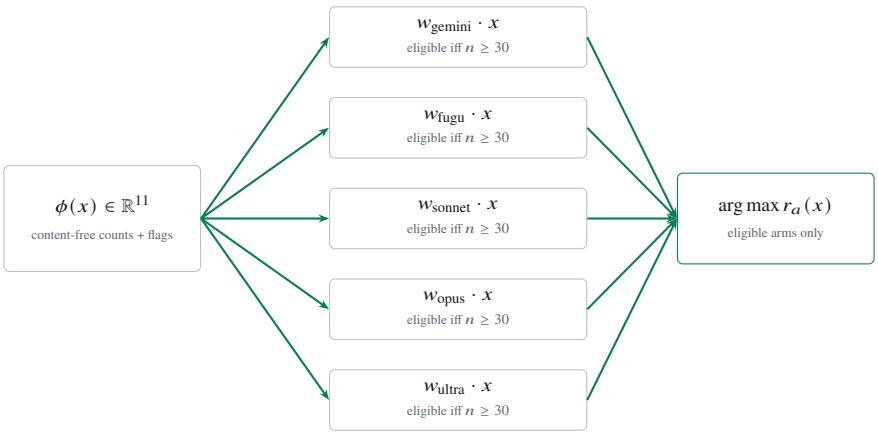

Figure 4: The policy network, such as it is. One shared 11-dimensional feature vector feeds five independent linear heads — one per rung, in ladder order — and the policy is an argmax over the heads that have cleared 30 training turns. 55 parameters total: small enough to read, large enough to beat a text-only judge on signals the judge cannot see.

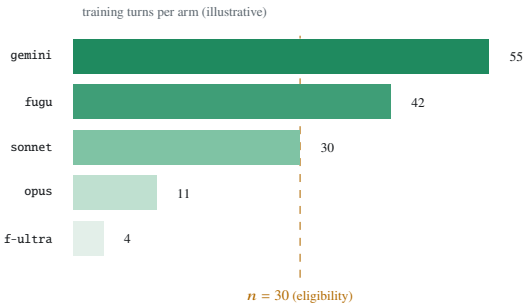

Figure 5: The activation bar in practice (illustrative counts). Traffic is skewed cheap, so bottom rungs cross the per-arm threshold first. In this state eq. (9) is *not* yet met ($\sum n_a = 142 < 300$, still warming up); once the total clears 300 with ≥ 2 eligible arms, the bandit starts driving the shadow pick — and only the eligible arms compete in the argmax, so a barely-sampled top rung can't be chosen on noise.

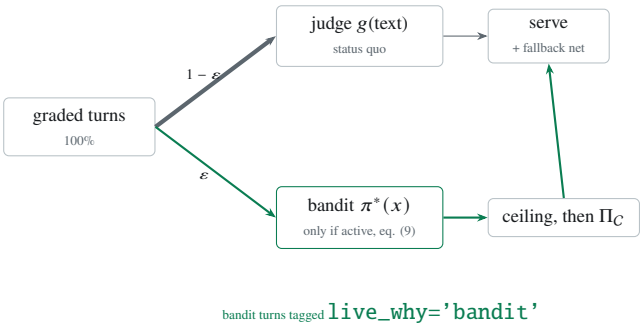

Figure 6: ε-greedy serving, eq. (12). The bandit path only exists while the policy is active, its pick is clamped first by the ceiling flag and then onto the admissible set C of §6.1, and every rung served — from either path — lands on the same self-healing fallback ladder. Tagged bandit turns feed the measured on-policy A/B.

a compounding effect: a lesson extracted from one tenant's mis-routed turn immediately improves routing for all.

The honest caveat is representation skew, and it has two sources. Early on, the policy's opinions are dominated by the most active tenants. Structurally, the plan-tier arm sets of §6.1 mean cheap-tier traffic can only ever train the fast rungs, so the upper arms are fed exclusively by the tenants entitled to them — the coverage in eq. (9) is not merely slow to arrive on those rungs, it is drawn from a narrower population. The serving guardrails (§6) carry both risks until the data broadens. All of it — shadow recording, training, serving fraction, ceiling — is governed by runtime configuration, so every stage is one flag from off.

9 Limitations and future work

Direct-method bias. Without logging propensities, counterfactual value estimates lean entirely on the reward model; we mitigate by treating them as screens and promoting only on measured on-policy comparisons, but a stochastic logging policy (even $\varepsilon = 0.02$ of exploration) would unlock doubly-robust estimation [4].

Reward shaping. Eq. (6)'s weights encode a judgment (a cent of spend is worth half a success). The report accepts weight overrides so sensitivity can be checked, but a principled calibration against retention or task-completion telemetry is future work.

Feature ceiling. Eleven linear features cannot represent interactions (a traceback in a *long* message means something different). The frozen-order vector design admits appending; the estimator admits replacement by any regressor without touching the recording, reward, or serving layers.

Non-stationarity. Discrete shifts — a model swapped behind a rung, a reprice — are handled by the fingerprint reset of §5.4, and constant-step-size SGD tracks gradual drift within a fixed model. What remains unaddressed is *slow* drift that never trips the fingerprint (a model's behavior changing under a stable id and price): the constant learning rate tracks it, but with a lag inversely proportional to that arm's traffic, so a rarely-served top rung adapts slowest. A recency-weighted or windowed fit is the natural upgrade, warranted only once such drift is measurable.

Beyond routing. The ledger → reward → per-arm learner → flag-gated serving pipeline is decision-agnostic. Candidate next users on the platform: which memory facts to inject into a turn, when to sleep an idle box, and voice selection.

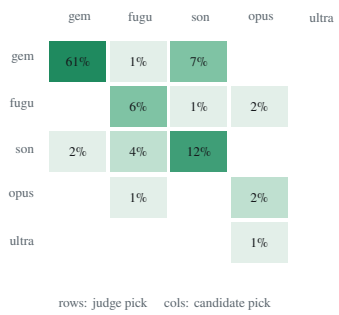

Figure 7: The confusion matrix the report renders (illustrative shape). Mass on the diagonal is agreement. The cells to watch are just above the diagonal: turns where the candidate starts one rung higher than the judge. When those correlate with the judge's own mid-turn escalations, the candidate is catching under-routing before it happens — the exact failure mode of §3.

10 Conclusion

A production LLM platform can grow a learned router the way it would grow any risky capability: behind observation first, a data bar second, and a bounded, measured serving fraction third. The machinery required is small — one ledger table, 55 linear parameters, three config flags — and every piece of it is legible to the operator who must trust it. The judge stays as the prior and the safety net; the bandit earns each increment of traffic with measured wins. Nothing about the recipe is specific to routing, which is rather the point.

References

1 W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933.

2 P. Auer, N. Cesa-Bianchi, P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

3 L. Li, W. Chu, J. Langford, R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. *WWW*, 2010.

4 M. Dudík, J. Langford, L. Li. Doubly robust policy evaluation and learning. *ICML*, 2011.

5 L. Bottou, J. Peters, J. Quiñonero-Candela, et al. Counterfactual reasoning and learning systems: the example of computational advertising. *JMLR*, 14:3207–3260, 2013.

6 A. Agarwal, S. Bird, M. Cozowicz, et al. Making contextual decisions with low technical debt. *arXiv:1606.03966*, 2016.

7 D. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen. A tutorial on Thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018.

8 R. S. Sutton, A. G. Barto. *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

9 A. Bietti, A. Agarwal, J. Langford. A contextual bandit bake-off. *JMLR*, 22(133):1–49, 2021.

10 L. Chen, M. Zaharia, J. Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv:2305.05176*, 2023.

11 I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, et al. RouteLLM: Learning to route LLMs with preference data. *arXiv:2406.18665*, 2024.

Errata — Revision V4

V4 came out of an audit of this report against the deployed code. Two of its items correct claims that were *wrong*, including one that V3 introduced; the rest close the gap that opened as the system shipped past the paper. The design is unchanged.

1. **Model-ladder ordering — retraction of V3 item 1.** V3 swapped `fugu` and `sonnet` in eq. (1) "to follow the price hierarchy of Fig. 2." That was a mistake: the ladder is ordered by capability tier, and price is not monotone along it — `fugu` is rung 1 at $20/M output, above rung 2 `sonnet` at $15/M. The deployed ladder is ⟨gemini, fugu, sonnet, opus, f-ultra⟩ and always was. Eq. (1), the per-arm heads of Fig. 4, the activation rows of Fig. 5 and both axes of Fig. 7 are restored to it; Fig. 2's caption now says explicitly that its bars are price-sorted, not ladder-sorted.
2. **Source of truth.** `_resolve_model` lives in `control-plane/llm_gateway.py`, not `main.py`; the closing note was pointing at a file the router had already left.
3. **Per-decision storage (§8).** The enumeration omitted the turn key — a truncated SHA-256 of the user message — and described the stored features as the scaled vector of eq. (4) when what is written is the raw counts, scaled later at training time. Both are corrected, and the guess-confirmation property of a content-derived hash is now stated instead of implied away.
4. **The serving path grew (§6.1, new).** Eq. (12) described a ceiling and nothing else. Deployed serving also applies per-request constraint rails (a latency bound and a floor rung, filtered on probe evidence), a plan-tier arm set, availability snapping, and a hard reroute of tool-bearing turns off the bottom rung; the bandit's pick now also carries a confidence score. Eq. (12) gains the admissible-set projection Π_C, and §8's skew caveat gains the structural half that the tier arm sets create.
5. **Escalation and notation.** Eq. (3) said the valve bumps one index; it bumps to the next *available* rung, which may skip an unfunded one. The ceiling in eq. (12) is renamed a_{ceil}, since $a_{\max}$ was doing duty for both it and the top of the ladder.
6. **Training attribution (§5.2).** "The arm the live judge served" was true only in shadow. Training keys on the arm actually served, whichever policy chose it — which is precisely what makes the on-policy claim in §6 hold once $\varepsilon > 0$.

One claim in §1 — that a majority of real turns are handled by the cheapest rung — is a statement about production traffic rather than about code, and was not re-verified in this audit.

Errata — Revision V3 (retained; item 1 withdrawn by V4)

1. **Model-ladder ordering.** Corrected the ladder in eq. (1) and the confusion matrix to follow the price hierarchy of Fig. 2, swapping the positions of `fugu` and `sonnet`. *[Withdrawn — see V4 item 1.]*
2. **Loss function.** Aligned the §5.2 objective with the gradient step of eq. (8): the ridge term is $\frac{1}{2}\lambda\|w\|_2^2$.
3. **Activation caption.** Corrected the logic — with $\sum n_a = 142 < 300$ the activation bar of eq. (9) is *not* met, so the system is still warming up.
4. **Typographical artifacts.** Fixed a rendering artifact in eq. (3) and a mislabeled fraction in the serving figure (now $\varepsilon = \text{pct}/100$).

Companion reading: the narrative version of this system lives in the Avriz engineering notebook at `https://avriz.io/eng` (§07 "Model routing" and §08 "The learning flywheel"). Source of truth: `control-plane/rl_router.py` (features, reward, bandit, constraint rails) and `_resolve_model` in `control-plane/llm_gateway.py` (judge, ladder, escalation, ε-greedy serving).