

Pennywise: Learning to Route LLM Calls with an Open Decision Model

Policy-gradient training of Laya as a cost-aware router

Pennywise contributors

Built 2026-10-26 from commit 74657e6-dirty · seed 0 · all numbers generated from evaluation outputs

Abstract. Agents send every request to a frontier model, including the many bounded ones a far cheaper model would answer correctly. We treat model-tier selection as a contextual bandit and use an open-weight, non-autoregressive decision model, Laya [1], as the policy: its distribution over {small, medium, large} is $\pi(a|s)$, available from one forward pass. After a supervised warm start on hindsight-optimal labels, we train one policy per cost weight λ with REINFORCE on $r = \text{correct}(a) - \lambda \cdot c(a)/c_{\text{ref}}$, observing only the sampled arm. We build an outcome table of 948 automatically graded prompts answered by three open-weight tiers (2844 calls; tier accuracies 74.5 / 83.8 / 84.0%, any-tier ceiling 91.6%). On a held-out test set of 189 prompts, the validation-selected auto policy ($\lambda = 0.1$) reaches **83.1%** accuracy (95% CI 77.8–88.4) at **1.9%** of the cost of always calling the largest model (CI 1.7–2.2), versus 84.1% for always-large; the paired accuracy difference is -1.1 points (CI -6.9 to +4.8). We report confidence intervals throughout and compare against strong cheap baselines, including where they win.

Interim draft. Built from real outcomes while RL training is still running; 2 of the λ policies ($\lambda \in \{0, 0.1\}$) had finished, so mode selection, RL rows and λ -sweep figures cover those only. Every RL number will change.

1 Introduction

Coding agents and LLM applications route nearly every call, from classification and summaries to multi-step reasoning, to one expensive model. Teams that measure AI spend in *dollars per merged pull request* pay frontier prices for requests a small open model would answer correctly. On our outcome table the largest tier costs on average $101\times$ more per call than the small tier and $43\times$ more than the medium tier, yet the medium tier’s accuracy (83.8%) is within a point of the large tier’s (84.0%). LLM routing addresses this gap by choosing a model per request [2], [3], [4].

A recent class of *decision models* returns typed answers with probabilities instead of generating text: the closed Jev [5] and the open-weight Laya [1]. Such a model’s output over a small, typed option set is exactly a stochastic policy, so it can be trained directly with policy gradients, without token-level RL.

Contributions (scoped to what we measured):

1. A router that uses Laya’s option distribution as the policy $\pi_\theta(a|s)$ over three model tiers, trained with a supervised warm start followed by REINFORCE on a correctness-minus-cost reward, yielding a family of policies indexed by λ (Section 4).
2. An outcome table of 948 automatically graded prompts (math, science QA, code, and an agent-style slice of bug-fix and CI-triage tasks built by bug injection), each answered by all three tiers (Section 5), with bootstrap confidence intervals and cheap embedding baselines, including per-tier correctness predictors swept over the same λ grid (Section 6).
3. An open implementation: an OpenAI-compatible proxy with sticky per-session routing and tool-call passthrough, so agent CLIs can use the router by changing one base URL (Section 8).

2 Related work

Routing and cascades. RouteLLM trains routers from preference data to choose between a strong and a weak model [2]. FrugalGPT learns cascades over multiple LLMs to reduce cost [3]. AutoMix combines small-model self-verification with a POMDP-based router that decides when to escalate [4]. RouterBench provides a benchmark and a large dataset of inference outcomes for evaluating routers [6]; we have not evaluated on it yet.

Decision models. Jev is a closed decision model returning typed choices, scores and yes/no answers with probabilities [5]. Laya is an open-weight model of the same category: a ModernBERT-large encoder [7] with a decision head that scores each option at its own marker in one pass [1]. REFLEX uses Jev as a fast typed decision layer inside an agent and escalates to a strong LLM when confidence is low [8]. We instead *train* the decision model as a routing policy with RL on observed outcomes, using open weights end to end.

3 Problem formulation

Let s be a request and $\mathcal{A} = \{\text{small}, \text{medium}, \text{large}\}$ the model tiers, ordered by price. Calling tier a on s yields a graded outcome $y(s, a) \in \{0, 1\}$ and a cost $c(s, a)$ in USD, computed from the call’s measured token counts at the tier’s public list price. A router is a policy $\pi(a|s)$. For a cost weight $\lambda \geq 0$ we maximise the expected reward

$$J_\lambda(\pi) = \mathbb{E}_{s \sim \mathcal{D}} \mathbb{E}_{a \sim \pi(\cdot|s)} \left[y(s, a) - \lambda \cdot \frac{c(s, a)}{c_{\text{ref}}} \right], \quad (1)$$

where $c_{\text{ref}} = 0.003138$ USD is the mean cost of always-large on the training split, which makes λ unitless: $\lambda = 1$ trades one correct answer for the cost of one average large-tier call. The *oracle* label $a^*(s)$ is the cheapest tier with $y(s, a) = 1$ (or *large* if none); it maximises Equation 1 for any $\lambda > 0$ in hindsight but is not observable at decision time.

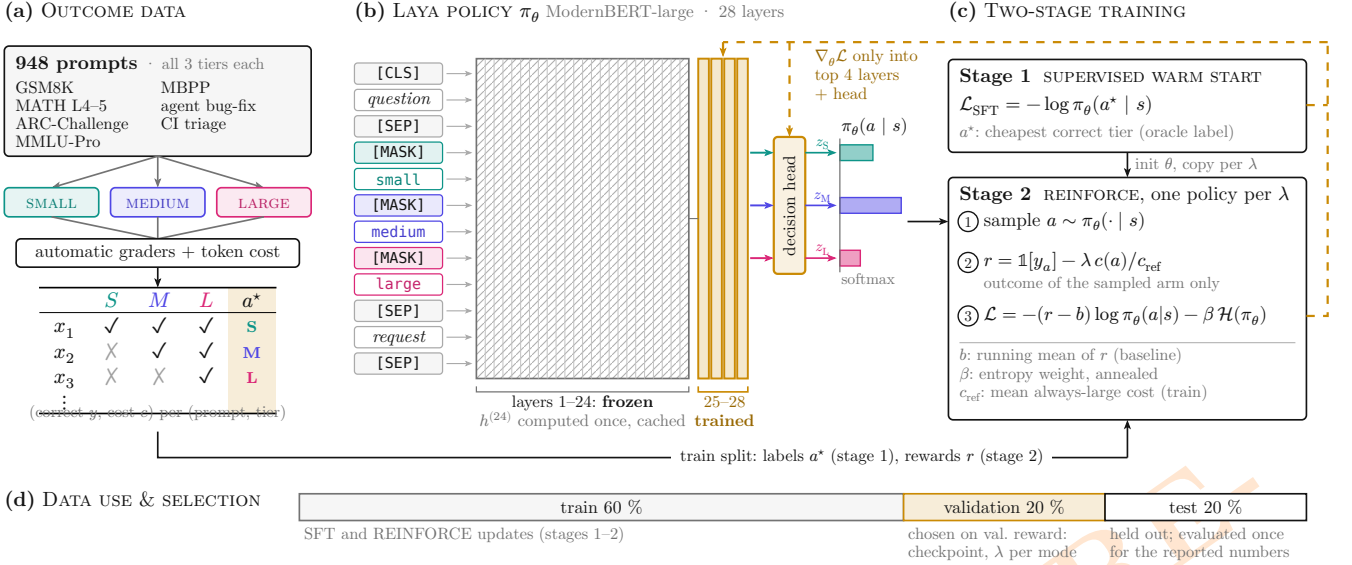


Figure 1. Training pipeline. (a) Every prompt is answered by all three tiers and graded automatically. (b) Laya scores the three tier options at their [MASK] markers in one non-autoregressive pass; the bottom 24 encoder layers are frozen and their outputs cached, the top 4 layers and the decision head are trained. (c) A supervised warm start on oracle labels is followed by REINFORCE per λ , using only the sampled arm’s reward. (d) Checkpoints and proxy modes are selected on validation; the test split is used once.

This is a contextual bandit: the router sees s , picks one arm, and in deployment observes only that arm’s outcome. Our outcome table contains all three outcomes for every prompt, which allows exact offline evaluation; training nevertheless uses only the sampled arm, as an online system would.

4 Method

4.1 Decision-model output as a policy

Laya encodes [CLS] choice question: <instructions> [SEP] [MASK] opt₀ [MASK] opt₁ [MASK] opt₂ [SEP] <request> [SEP] and reads one logit z_a per option at its [MASK] position [1]. With the three tier options, each given a short description, the softmax

$$\pi_\theta(a|s) = \exp(z_a(s)) / \sum_{a'} \exp(z_{a'}(s)) \quad (2)$$

defines the policy. One forward pass yields the full distribution, so $\log \pi_\theta(a|s)$ is available for every action at the cost of a single encoder call (Figure 1 b).

4.2 Supervised warm start

We first fine-tune on the oracle label with cross-entropy over the three options, for up to 10 epochs with early stopping on validation loss. To fit a 4 GB GPU we train only the top 4 encoder layers plus the decision head. Because the lower layers are frozen, their outputs are computed once per prompt and cached; training then runs only the trainable layers, which is identical to the full forward pass up to bf16 rounding.

4.3 REINFORCE with baseline and entropy

Starting from the warm start, for each $\lambda \in \{0, 0.1, 0.25, 0.5, 1, 2\}$ we train a separate policy for 800 steps with batches of 16 training prompts, minimising

$$\mathcal{L}(\theta) = -(r - b) \log \pi_\theta(a|s) - \beta H(\pi_\theta(\cdot | s)), \quad (3)$$

with $a \sim \pi_\theta(\cdot | s)$ sampled from the current policy, r the sampled arm’s reward from Equation 1, b an exponential running mean of the batch reward [9], and β annealed

Input: warm-start θ_0 , train/val prompts, outcome table (y, c) , λ
1: $b \leftarrow 0, \theta \leftarrow \theta_0$
2: **for** $t = 1 \dots T$ **do**
3: sample batch $\{s_i\}$; $a_i \sim \pi_\theta(\cdot | s_i)$ one forward pass
4: $r_i \leftarrow y(s_i, a_i) - \lambda c(s_i, a_i) / c_{\text{ref}}$ sampled arm only
5: $\theta \leftarrow \theta - \eta \nabla_{\theta} \frac{1}{n} \sum_i \mathcal{L}_i$ (Equation 3) top 4 layers + head
6: $b \leftarrow 0.95b + 0.05r$; anneal β
7: every few steps: keep θ if val reward improves
8: **return** best θ

Algorithm 1. REINFORCE routing policy for one cost weight λ .

linearly from 0.05 to 0.005. The checkpoint with the highest validation reward is kept (Algorithm 1).

Proxy modes. Three modes map to trained λ values by fixed rules applied to validation results: **auto** is the cheapest RL policy whose accuracy is not significantly below always-large; **auto-quality** the most accurate; **auto-cheap** the cheapest at least as accurate as always-medium. Here **auto** = 0.1, **auto-quality** = 0, **auto-cheap** = 0.

5 Data and setup

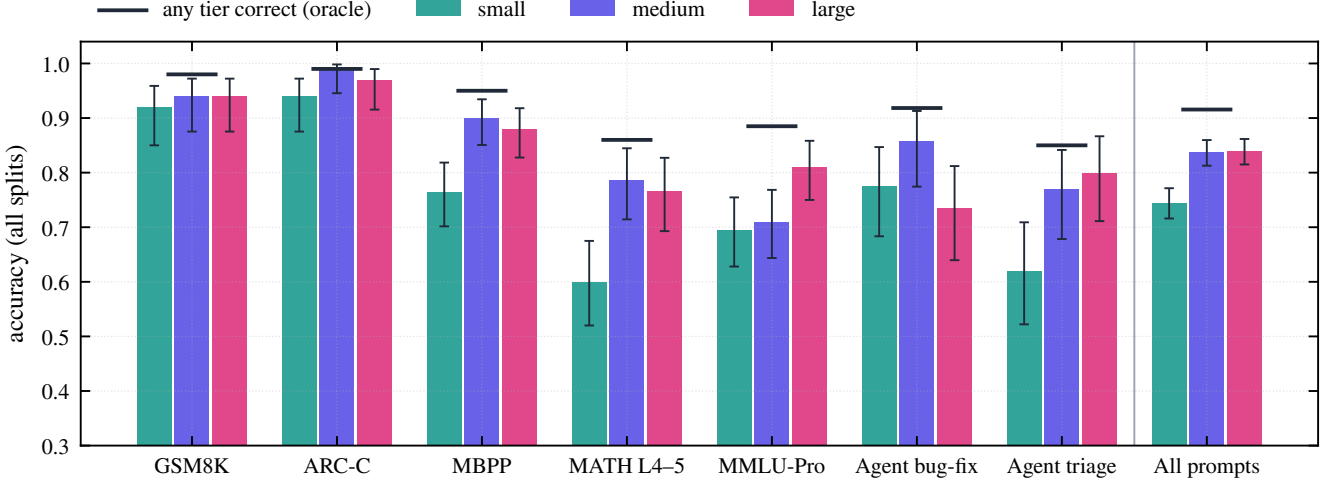
Prompts. 948 prompts with automatically checkable answers, mixing easy and hard sources because a pilot showed current open models saturate the easy ones: GSM8K [10] (final number, exact match), MATH-500 levels 4–5 (the subset [11] of MATH [12]; last `\boxed{\}` answer after LaTeX normalisation), ARC-Challenge [13] and MMLU-Pro [14] (option letter), MBPP [15] (unit tests in a sandboxed subprocess, 5 s timeout), and an agent-style slice built from disjoint, longer MBPP functions: *bug-fix* prompts (one or two injected bugs; graded by the original tests) and *CI-triage* prompts (classify the injected bug type from the code and real test output). Splits are stratified by source, 569 / 190 / 189 (train / val / test).

Model tiers. All three tiers are open-weight models with tool calling, served through OpenAI-compatible endpoints (prices per million tokens, input / output, USD):

Tier	Model	In	Out
small	nvidia/nemotron-3.5-bolt-e4b	0.04	0.05

Table 1. Outcome table by source. Accuracy is over all splits; the oracle label is the cheapest tier that answered correctly (“none”: no tier did). Output-token medians show the reasoning length each tier spends.

Source	n	tr/va/te	accuracy %			oracle label %				median out-tokens		
			S	M	L	S	M	L	none	S	M	L
GSM8K	100	60/20/20	92.0	94.0	94.0	92.0	3.0	3.0	2.0	260	259	342
ARC-C	100	60/20/20	94.0	99.0	97.0	94.0	5.0	0.0	1.0	83	155	280
MBPP	200	120/40/40	76.5	90.0	88.0	76.5	15.0	3.5	5.0	153	396	609
MATH L4-5	150	90/30/30	60.0	78.7	76.7	60.0	23.3	2.7	14.0	1086	1084	1385
MMLU-Pro	200	120/40/40	69.5	71.0	81.0	69.5	11.0	8.0	11.5	310	408	863
Agent bug-fix	98	59/20/19	77.6	85.7	73.5	77.6	13.3	1.0	8.2	107	627	1645
Agent triage	100	60/20/20	62.0	77.0	80.0	62.0	18.0	5.0	15.0	78	167	690
All	948	569/190/189	74.5	83.8	84.0	74.5	13.3	3.8	8.4	208	379	696

**Figure 2.** Accuracy per source and tier (all 948 prompts, Wilson 95% intervals). Black ticks: fraction of prompts answered correctly by at least one tier, the ceiling for any router. The medium tier matches or beats the large tier on most sources; the large tier’s advantage concentrates on MMLU-Pro and agent triage.

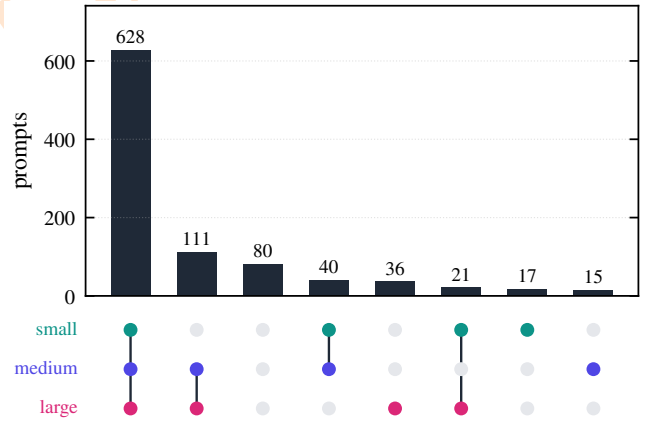
medium	openai/gpt-oss-20b	0.018	0.09
large	nvidia/nemotron-3-ultra-550b-a55b	0.6	2.4

All 2844 calls ran on a free API (actual spend \$0), so every cost is the public list price of the same model on a paid provider; the whole table would have cost \$3.16 (0.54 M input, 2.45 M output tokens). The small tier has no paid listing; its price is a *proxy*, the list price of a model of the same ~4B size class.

The tiers are complementary rather than nested (Figure 3): 66.2% of prompts are solved by all three tiers and 8.4% by none, while 36 prompts are solved *only* by the large tier and 57 are solved by the small tier but missed by the large one. Routing therefore has headroom (any-tier ceiling 91.6% vs. 84.0% for always-large) but cannot be reduced to a difficulty threshold.

Baselines. Always-small/medium/large; uniform random; the oracle (hindsight, not achievable); zero-shot Laya; logistic regression and k -NN predicting the oracle tier from frozen embeddings (base Laya, mean-pooled; and all-MiniLM-L6-v2 [16]), with C and k tuned on validation; and per-tier correctness predictors, one logistic regression per tier estimating $P(\text{correct}|s, a)$, acting by $\arg\max_a P(\text{correct}|s, a) - \lambda \bar{c}_a / c_{\text{ref}}$ over the same λ grid. Jev: not run (no JEV_API_KEY).

Metrics. Test accuracy, mean cost per request, cost relative to always-large, and router latency, each with 95% percentile bootstrap intervals (1000 resamples); comparisons against always-large use paired bootstrap differences.

**Figure 3.** Correctness patterns over the three tiers (dots: tier correct). 8 of 8 possible patterns occur.

6 Results

The validation-selected auto policy ($\lambda = 0.1$) differs from always-large in accuracy by -1.1 points (paired 95% CI -6.9 to +4.8) while costing 1.9% as much (Table 3). Zero-shot Laya reaches 77.2% at 20.1% cost. Figure 6 places the RL family against the per-tier correctness predictors, the strongest cheap baseline; where those match or beat the RL policies, the Laya router’s remaining advantages are architectural (a single model scoring options directly, trainable from bandit feedback) rather than accuracy.

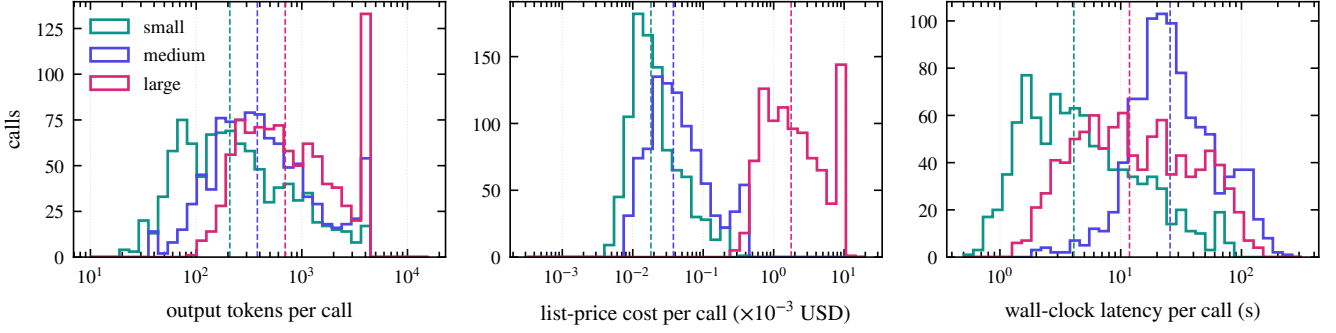


Figure 5. Per-call distributions by tier (dashed: medians). Output length dominates cost; the spike at the right of the large tier’s token histogram is the generation cap. Median wall-clock latency on the free API: small 4.1 s, medium 25.6 s, large 11.8 s (shared, rate-limited endpoints; not a serving benchmark).

Table 3. Test-set results ($n = 189$). Brackets: 95% bootstrap intervals. Per-tier LR rows are shown for a subset of λ . Router latency: median single-request time on the build machine’s GPU. Pareto: not dominated in (accuracy, cost) by another non-oracle policy.

Policy	Accuracy % [95% CI]	Cost % of always-large [95% CI]	Router ms	Pareto
always-small	73.5 [67.2, 79.9]	1.0 [0.8, 1.1]	0.0	✓
always-medium	82.5 [77.2, 87.8]	2.1 [1.8, 2.3]	0.0	
always-large	84.1 [78.8, 88.9]	100.0 [100.0, 100.0]	0.0	
random	79.9 [74.1, 85.2]	41.4 [32.4, 51.5]	0.0	
oracle	93.7 [90.5, 96.8]	26.0 [17.4, 35.1]		
zero-shot Laya	77.2 [71.4, 83.1]	20.1 [12.4, 28.7]	43.9	
LR (Laya-emb)	74.1 [68.3, 79.9]	6.8 [2.6, 12.4]	30.9	
kNN (Laya-emb)	73.5 [67.2, 79.9]	1.1 [0.9, 1.5]	30.9	
per-tier LR (Laya-emb) ($\lambda=0.1$)	82.5 [76.7, 87.8]	8.5 [4.2, 13.4]	30.9	
per-tier LR (Laya-emb) ($\lambda=0.25$)	81.5 [75.7, 86.8]	2.8 [1.7, 4.5]	30.9	
per-tier LR (Laya-emb) ($\lambda=0.5$)	81.5 [75.7, 86.8]	1.8 [1.5, 2.0]	30.9	✓
per-tier LR (Laya-emb) ($\lambda=1$)	81.0 [75.1, 86.2]	1.8 [1.5, 2.0]	30.9	✓
LR (MiniLM-emb)	75.1 [68.8, 81.0]	4.8 [1.5, 9.0]	6.4	
kNN (MiniLM-emb)	73.5 [67.2, 79.9]	1.1 [0.9, 1.3]	6.4	
per-tier LR (MiniLM-emb) ($\lambda=0.1$)	83.1 [77.8, 88.4]	7.6 [3.9, 12.3]	6.4	
per-tier LR (MiniLM-emb) ($\lambda=0.25$)	82.0 [76.7, 87.3]	3.4 [2.0, 5.1]	6.4	
per-tier LR (MiniLM-emb) ($\lambda=0.5$)	81.0 [75.1, 86.2]	1.9 [1.6, 2.1]	6.4	
per-tier LR (MiniLM-emb) ($\lambda=1$)	81.0 [75.1, 86.2]	1.8 [1.6, 2.1]	6.4	
SFT Laya	73.5 [67.2, 79.9]	1.0 [0.8, 1.1]	43.9	✓
RL Laya ($\lambda=0$)	83.1 [77.2, 88.4]	86.2 [78.8, 92.4]	43.9	
RL Laya ($\lambda=0.1$)	83.1 [77.8, 88.4]	1.9 [1.7, 2.2]	43.9	✓

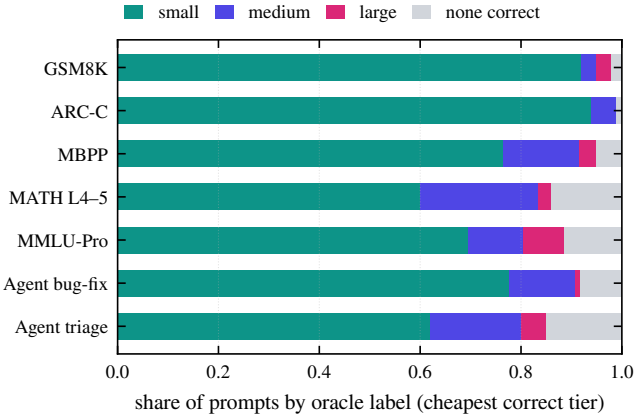


Figure 4. Oracle-label mix per source. Most prompts are solvable by the small tier; the oracle needs the large tier rarely.

Table 2. Training configuration.

Encoder	ModernBERT-large, 28 layers, 421M params (Laya)
Trainable	top 4 layers + decision head
SFT	lr 2e-05, batch 8×2, ≤10 epochs, patience 3
RL	REINFORCE, 800 steps, batch 16, lr 5e-06
Baseline b	running mean, momentum 0.95
Entropy β	0.05 → 0.005 (linear)
λ grid	0, 0.1, 0.25, 0.5, 1, 2
Selection	best val reward (checked every 40 steps)
Bootstrap	1000 resamples, seed 0
Hardware	1× RTX 3050 Laptop (4 GB), bf16

7 Analysis

What the router learns. Figure 10 compares the auto policy’s actions with the oracle label on the test split; it agrees with the oracle on 17.5% of prompts. Off-diagonal mass above the diagonal is money spent on a tier more

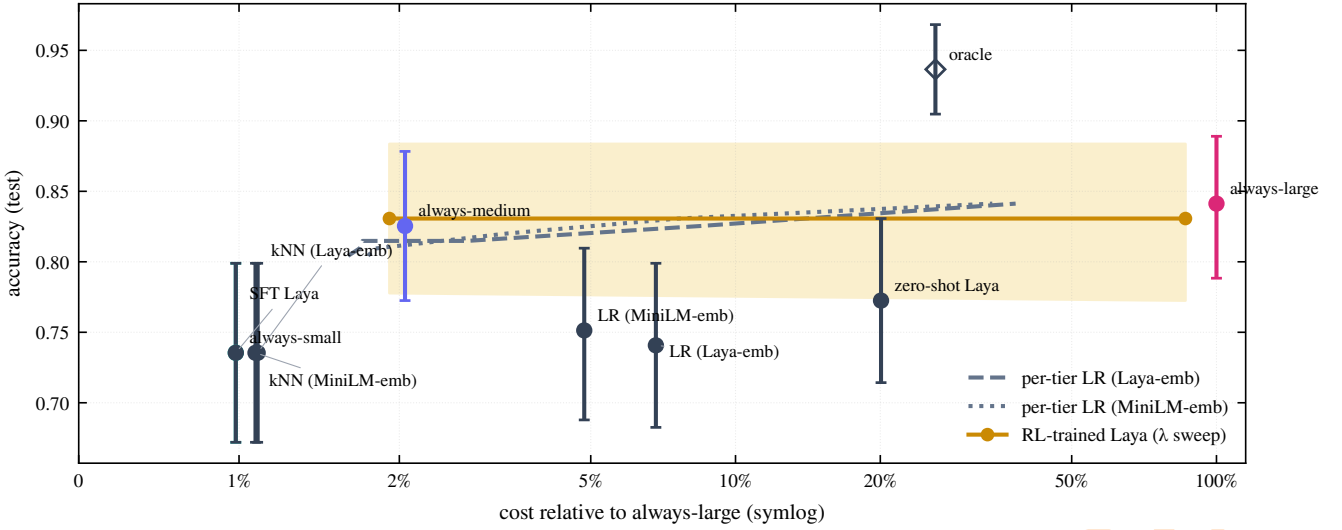


Figure 6. Accuracy vs. cost relative to always-large on the test split (symmetric-log cost axis). Shaded: 95% CI of the RL family; error bars: 95% CI; hollow diamond: hindsight oracle.

Table 4. RL policies vs. always-large on the test split: paired accuracy difference and relative cost with 95% bootstrap intervals, test action share (small/medium/large), mean reward, and the selected checkpoint.

λ	Acc %	Δ vs large (pts)	[95% CI]	Cost %	[95% CI]	S/M/L %	Reward	Best step
0	83.1	-1.1	[-3.7, +1.6]	86.2	[78.8, 92.4]	7/8/85	0.831	800
0.1	83.1	-1.1	[-6.9, +4.8]	1.9	[1.7, 2.2]	6/94/0	0.829	800

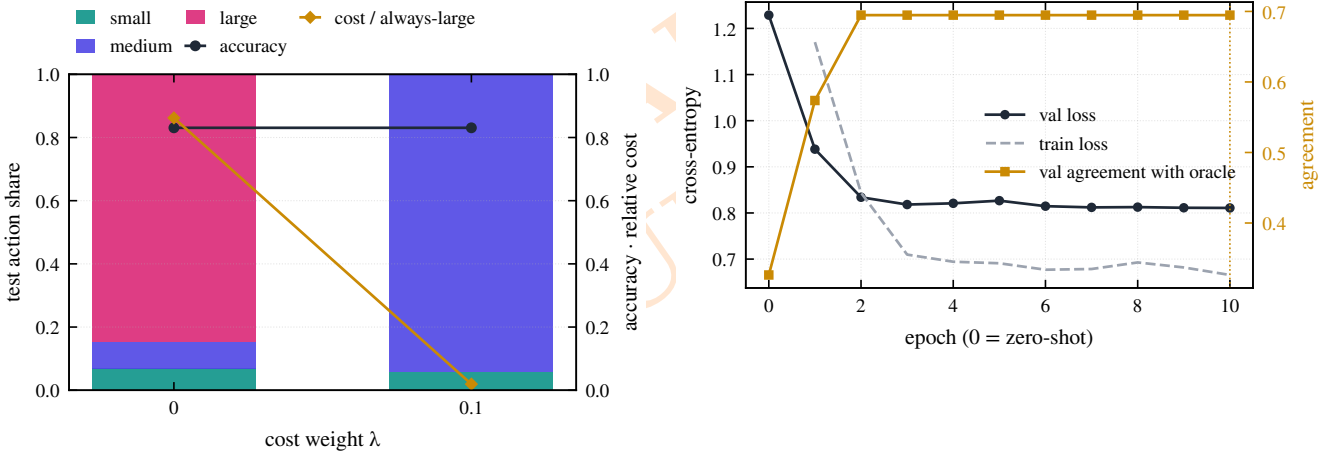


Figure 7. Left: effect of the cost weight, test action share (bars) with accuracy and relative cost (lines) for each trained λ . Right: supervised warm start, loss (left axis) and validation agreement with the oracle label (right axis), which rises from 32.6% zero-shot to 69.5% (epoch 10 dotted).

expensive than necessary; mass below it is accuracy risked on a tier cheaper than the cheapest correct one.

Confidence. Figure 11 relates the policy’s probability of its chosen tier to whether that tier answered correctly (point-biserial $r = -0.09$). The policy’s probabilities are trained for reward, not calibrated for correctness; we report them as-is and do not use them for escalation.

8 System

The router adds one encoder pass, 43.9 ms median on the build machine’s GPU for `auto`. The proxy is OpenAI-compatible: it routes once per conversation (keyed by the system prompt and first user message, or an `x-session-id` header) and keeps that tier for the session so agent tool loops are never re-routed mid-task; tools, tool choice and tool messages pass through unchanged, including when streaming (Figure 12).

9 Limitations and future work

The test set is small ($n = 189$), so intervals are wide; we use a single seed. The setting is offline: although RL uses only the sampled arm, the outcome table was collected with full information, which an online system would not have. Benchmark prompts are not production traffic, and the agent slice consists of synthetic single-function tasks. Costs are list prices of a free-API run, one of them a proxy price. Laya is known to degrade with many options [1], which is irrelevant at three tiers but limits extension to many models. Jev was not benchmarked.

Planned work: partial-feedback online RL with a feedback endpoint; comparison with an exact expected-reward objective (possible offline because all outcomes are known); evaluation on RouterBench [6]; out-of-distribution tests; more tiers; multiple seeds; calibration.

test accuracy (%) · cost relative to always-large

always-small	95 1% cost	85 1% cost	82 1% cost	53 2% cost	72 1% cost	68 0% cost	60 1% cost
always-medium	95 2% cost	100 1% cost	98 2% cost	70 3% cost	65 2% cost	79 2% cost	80 1% cost
always-large	90 100% cost	95 100% cost	90 100% cost	70 100% cost	85 100% cost	79 100% cost	80 100% cost
oracle	95 16% cost	100 1% cost	100 1% cost	83 41% cost	95 33% cost	89 31% cost	90 29% cost
zero-shot Laya	95 2% cost	95 1% cost	80 32% cost	67 21% cost	65 32% cost	74 2% cost	80 4% cost
SFT Laya	95 1% cost	85 1% cost	82 1% cost	53 2% cost	72 1% cost	68 0% cost	60 1% cost
RL auto-quality ($\lambda=0$)	90 100% cost	95 100% cost	88 79% cost	70 93% cost	85 85% cost	84 83% cost	70 77% cost
RL auto ($\lambda=0.1$)	95 2% cost	100 1% cost	95 2% cost	73 3% cost	68 2% cost	84 1% cost	75 1% cost
	GSM8K (n=20)	ARC-C (n=20)	MBPP (n=40)	MATH L4-5 (n=30)	MMLU-Pro (n=40)	Agent bug-fix (n=19)	Agent triage (n=20)

Figure 8. Per-source test accuracy (large digits, %) and cost relative to always-large (small text) for trivial policies, the oracle, Laya without RL, and the three RL modes. Agent-style slices are the two rightmost columns.

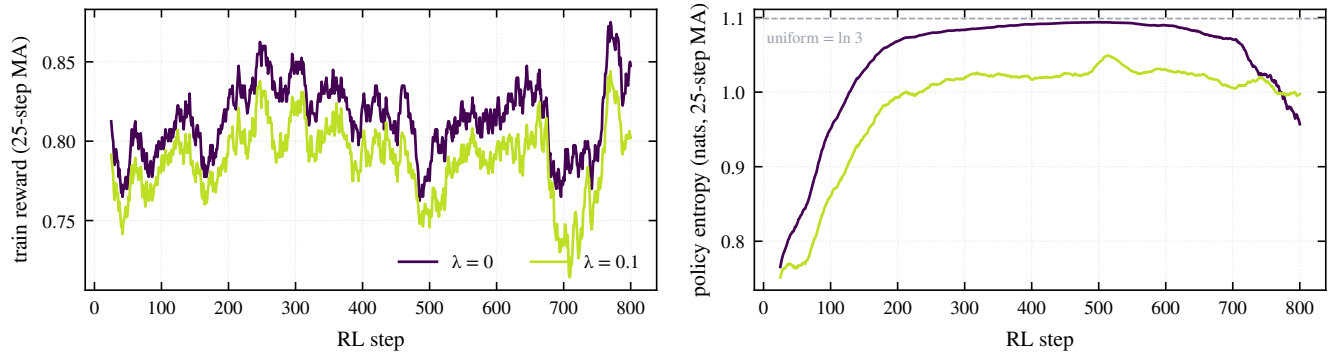


Figure 9. RL training dynamics per λ (25-step moving averages over sampled actions): reward (left) and policy entropy (right; dashed: uniform policy).

10 Reproducibility

Code, configs and this report’s sources are at commit 74657e6-dirty. config/models.yaml (tiers, prices) and config/train.yaml (splits, hyperparameters, λ grid, seed) define the run. Pipeline: data/build_prompts.py $\rightarrow$ data/run_models.py

small	8	131	0
medium	2	25	0
large	1	22	0
	small	medium	large

router action (auto, $\lambda = 0.1$)

Figure 10. Router action vs. oracle label on the test split (counts) for auto.

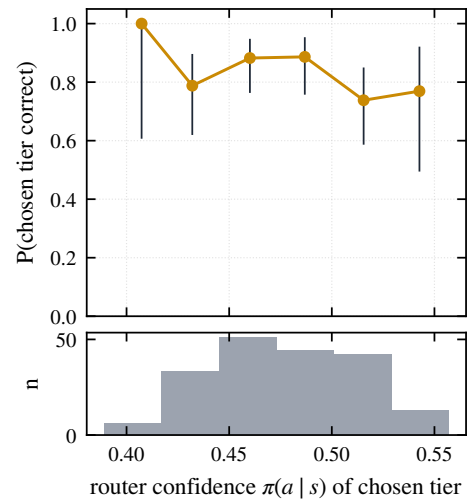
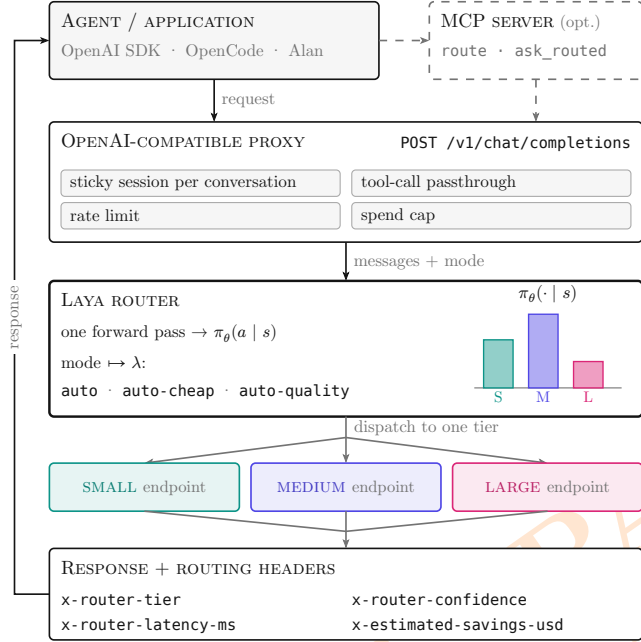


Figure 11. Reliability of the auto policy’s confidence (bins with ≥ 3 prompts; Wilson 95% intervals) and its histogram.

$\rightarrow$ eval.features $\rightarrow$ eval.baselines $\rightarrow$ router.sft $\rightarrow$ router.rl $\rightarrow$ eval.evaluate $\rightarrow$ eval.export_viz $\rightarrow$ paper/

Table 5. Per-source accuracy % / cost % of always-large on the test split.

Source	always-small	always-medium	always-large	RL Laya ($\lambda=0.1$)
Agent bug-fix (n=19)	68.4 / 0.3	78.9 / 1.7	78.9 / 100.0	84.2 / 1.1
Agent CI triage (n=20)	60.0 / 0.6	80.0 / 0.9	80.0 / 100.0	75.0 / 0.9
ARC-Challenge (n=20)	85.0 / 0.7	100.0 / 1.1	95.0 / 100.0	100.0 / 1.1
GSM8K (n=20)	95.0 / 1.1	95.0 / 2.2	90.0 / 100.0	95.0 / 2.2
MATH L4-5 (n=30)	53.3 / 1.7	70.0 / 3.0	70.0 / 100.0	73.3 / 2.8
MBPP (n=40)	82.5 / 0.7	97.5 / 2.1	90.0 / 100.0	95.0 / 2.1
MMLU-Pro (n=40)	72.5 / 1.0	65.0 / 1.9	85.0 / 100.0	67.5 / 1.9

**Figure 12.** Serving path. Agents point an OpenAI SDK at the proxy; Laya routes once per conversation and the chosen tier answers; routing metadata is returned in response headers. An MCP server exposes the same router as tools.

build.py. Every number, table and figure in this report is regenerated from the evaluation outputs by paper/gen_tables.py and paper/gen_extra.py.

References

- [1] Convai Innovations, “Laya: model card.” [Online]. Available: <https://huggingface.co/convaiinnovations/laya>
- [2] I. Ong *et al.*, “RouteLLM: Learning to Route LLMs with Preference Data.” [Online]. Available: <https://arxiv.org/abs/2406.18665>
- [3] L. Chen, M. Zaharia, and J. Zou, “FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance.” [Online]. Available: <https://arxiv.org/abs/2305.05176>
- [4] P. Aggarwal *et al.*, “AutoMix: Automatically Mixing Language Models.” [Online]. Available: <https://arxiv.org/abs/2310.12963>
- [5] LDS Team, “TypeSafe AI Launches Jev for Structured Software Decisions.” [Online]. Available: <https://letsdatascience.com/news/typesafe-ai-launches-jev-decision-model-889a38c0>
- [6] Q. J. Hu *et al.*, “RouterBench: A Benchmark for Multi-LLM Routing System.” [Online]. Available: <https://arxiv.org/abs/2403.12031>
- [7] B. Warner *et al.*, “Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference.” [Online]. Available: <https://arxiv.org/abs/2412.13663>

- [8] T. Wu and W. Y. B. Lim, “REFLEX with Jev for Efficient Selective Control in LLM Agents.” [Online]. Available: <https://arxiv.org/abs/2609.26532>
- [9] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, no. 3–4, pp. 229–256, 1992, doi: [10.1007/BF00992696](https://doi.org/10.1007/BF00992696).
- [10] K. Cobbe *et al.*, “Training Verifiers to Solve Math Word Problems.” [Online]. Available: <https://arxiv.org/abs/2110.14168>
- [11] H. Lightman *et al.*, “Let’s Verify Step by Step.” [Online]. Available: <https://arxiv.org/abs/2305.20050>
- [12] D. Hendrycks *et al.*, “Measuring Mathematical Problem Solving With the MATH Dataset.” [Online]. Available: <https://arxiv.org/abs/2103.03874>
- [13] P. Clark *et al.*, “Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge.” [Online]. Available: <https://arxiv.org/abs/1803.05457>
- [14] Y. Wang *et al.*, “MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark.” [Online]. Available: <https://arxiv.org/abs/2406.01574>
- [15] J. Austin *et al.*, “Program Synthesis with Large Language Models.” [Online]. Available: <https://arxiv.org/abs/2108.07732>
- [16] sentence-transformers, “all-MiniLM-L6-v2: model card.” [Online]. Available: <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>