A Contextual Bandit for Cheap-vs-Expensive Code Generation

Mauricio Berlanga
Department of Computer Science
Stanford University
mberla@stanford.edu

Extended Abstract

Problem and motivation. Deploying large language models for code generation forces a recurring tradeoff: frontier API models (e.g., Claude Sonnet, GPT Codex) achieve high unit-test pass rates but incur dollar cost and latency, while small locally hosted models are cheap but fail more often on hard tasks. We ask whether a learned *router* can approach strong-model quality while paying for the expensive model only when necessary, and—crucially—whether the executable feedback unique to code generation makes routing easier than in the prompt-only LLM-routing literature.

Approach. We frame routing as a cost-sensitive contextual bandit that maximizes $R(x, a) = s(x, a) - \lambda\, c(a)$, where s is hidden-test quality, c is normalized cost, λ controls the cost–quality tradeoff, and the action a is `CHEAP` (keep the cheap completion) or `EXPENSIVE` (escalate to a strong model). Unlike prior routers that condition only on prompt features, our *verifier-aware* state includes the cheap model's completion and its *visible* unit-test outcome, observed before deciding to escalate. We compare deterministic baselines (always-cheap, always-strong, prompt-length heuristic, cheap-then-escalate, and a hindsight oracle) against learned routers: a prompt-only logistic/MLP imitator, a 149M ModernBERT classifier, a Qwen3.5-0.8B LoRA router, and REINFORCE policy-gradient training. All generation models are frozen; only the router is trained.

Experiments and main findings. On 427 MBPP-sanitized tasks with a leakage-safe visible/hidden test split, a simple deterministic cheap-then-escalate policy reaches **0.958** hidden pass rate at **0.257** normalized API cost while escalating on only 16.9% of tasks—nearly matching the hindsight oracle and dominating every prompt-only learned router. The Qwen3.5-0.8B LoRA router is viable (best cost-quality point 0.961 pass rate at 0.224 cost) but seed-sensitive and does not reliably beat the simple verifier rule. On held-out EvalPlus benchmarks with Claude Sonnet 4.6 as the strong model, cheap-then-escalate matches Sonnet-level plus-test quality at **19–23%** of normalized API spend (HumanEval+: 0.915 at 0.192 cost; MBPP+: 0.825 at 0.229), and an MBPP-trained Qwen3.5-0.8B LoRA router transfers only after cross-benchmark threshold recalibration (combined score 0.833→0.889). Direct REINFORCE either collapses to always-cheap at high λ or fails to improve on supervised routing, a useful negative result in this sparse-reward offline setting.

Accomplishments and insights. We built a reproducible collection/evaluation harness, logged six generation models, and produced cost–quality frontiers with paired-bootstrap 95% confidence intervals on both development and held-out benchmarks. The central insight is a clear hierarchy of routing signals for code: *executable visible verifier feedback is the dominant signal*; static prompt features are insufficient; the Qwen3.5-0.8B LoRA router is viable but requires seed aggregation and calibration; and naive policy gradients are fragile. The project satisfies the course novelty criteria by answering an open question about verifier-aware routing, adapting contextual bandits to a non-trivial application domain, and analyzing failure modes where learned methods underperform a simple rule.

Abstract

Strong code-generation models are accurate but expensive; small local models are cheap but fail on harder tasks. We study *verifier-aware cost-sensitive routing*: a contextual bandit that chooses whether to keep a cheap local completion or escalate to a stronger API model using reward $R(x, a) = \text{quality}(x, a) - \lambda \cdot \text{cost}(a)$. Unlike prompt-only LLM routers, our setting exposes *executable visible unit-test feedback* from a cheap first attempt as routing state. On 427 MBPP-sanitized tasks with leakage-safe hidden scoring, a deterministic cheap-then-escalate policy reaches 0.958 hidden pass rate at 0.257 normalized API cost (16.9% escalation), nearly matching the hindsight oracle and outperforming prompt-only learned routers. A Qwen3.5-0.8B LoRA router is viable (best point: 0.961 pass rate at 0.224 cost) but does not reliably beat the simple verifier rule. On held-out EvalPlus benchmarks with Claude Sonnet 4.6 as the strong model, cheap-then-escalate matches Sonnet-level plus-test quality at 19–23% of normalized API spend. Direct REINFORCE collapses or yields unstable gains in this sparse-reward offline setting. **Key insight:** for code generation, visible verifier feedback is the dominant routing signal; the Qwen3.5-0.8B LoRA router requires careful cross-benchmark calibration and still struggles to beat a simple escalation rule.

1 Introduction

Deploying large language models for code generation involves a recurring tradeoff: frontier API models (e.g., Claude Sonnet, GPT Codex) achieve high unit-test pass rates but incur dollar cost and latency, while smaller locally hosted models are cheaper but fail more often. A natural question is whether a *router* can approach strong-model quality while paying for the expensive model only when necessary.

Prior LLM routing work often conditions on prompt features or model confidence [Chen et al., 2024, Ong et al., 2024]. Code generation offers a different signal: after running a cheap model, we can *execute visible unit tests* and observe whether the completion passes before deciding to escalate. This creates a sequential, verifier-aware contextual bandit where the routing state includes both the task and the cheap model's observable behavior.

We formulate routing as maximizing

$$R(x, a) = s(x, a) - \lambda\, c(a), \tag{1}$$

where s is hidden-test quality, c is normalized cost, and λ controls the cost–quality tradeoff. Actions are `CHEAP` (keep the cheap completion) or `EXPENSIVE` (pay for a strong model). We compare deterministic baselines, prompt-only supervised routers, a compact ModernBERT classifier, a Qwen3.5-0.8B LoRA router, and REINFORCE policy-gradient training.

Our contributions are threefold. **(1)** We show that verifier-aware cheap-then-escalate routing is a strong, deployable baseline that nearly matches the hindsight oracle on MBPP and transfers to EvalPlus with Sonnet 4.6. **(2)** We demonstrate that the Qwen3.5-0.8B LoRA router can learn a cost–quality frontier but is seed-sensitive and calibration-dependent. **(3)** We provide failure-mode analysis showing that prompt-only routing and naive REINFORCE are fragile in sparse-reward settings—insights that hold even when learned methods do not beat simple rules.

2 Related Work

LLM routing and cascades. FrugalGPT [Chen et al., 2024] cascades models using confidence and query complexity; RouteLLM [Ong et al., 2024] trains routers on preference data. These methods typically route *before* generation using prompt-level features. We instead study post-generation routing with executable verifier feedback, which is natural for code but underexplored in the routing literature.

Code evaluation. HumanEval [Chen et al., 2021], MBPP [Austin et al., 2021], and EvalPlus [Liu et al., 2023] provide unit-test-based evaluation with augmented hidden tests that reduce benchmark

leakage. We adopt EvalPlus plus-test scores for held-out generalization and enforce a visible/hidden test split so routing inputs never leak held-out assertions.

Contextual bandits and policy gradients. Our objective is a cost-sensitive contextual bandit [Sutton and Barto, 2018]. We compare supervised oracle imitation (a strong offline baseline when both model outcomes are logged) with REINFORCE [Williams, 1992]. Because our cached dataset contains outcomes for both models, exploration is unnecessary; this makes direct policy gradients a useful *negative* comparison rather than the primary training method.

3 Problem Formulation

Setting. For each coding task x, a cheap local model produces completion $\hat{y}_{\text{local}}$. A verifier executes visible unit tests, yielding score $v(x, \hat{y}_{\text{local}}) \in [0, 1]$ and an error type. The router observes context $x' = (x, \hat{y}_{\text{local}}, v, \text{error})$ and chooses action $a \in \{\texttt{CHEAP}, \texttt{EXPENSIVE}\}$. If $a = \texttt{EXPENSIVE}$, a strong model generates $\hat{y}_{\text{strong}}$ and we score it on hidden tests; if $a = \texttt{CHEAP}$, we score $\hat{y}_{\text{local}}$ on hidden tests. Sequential escalation always pays the cheap generation cost plus the expensive-model cost when escalating.

Leakage-safe evaluation. On MBPP-sanitized, the first assertion shown in the prompt is *visible*; remaining assertions are *hidden*. Routing policies may use visible scores; final claims use hidden scores only. EvalPlus routing rows use cheap *base* test failure as the visible escalation trigger and plus-test scores for evaluation.

Metrics. For each policy and λ, we report hidden/plus-test pass rate, normalized cost, reward (Eq. 1), escalation rate, and regret to the per-task hindsight oracle. We compute paired-bootstrap 95% confidence intervals over tasks (2000 resamples).

4 Methods

Baselines. *Always cheap* and *always strong* bound the frontier. *Prompt-length heuristic* escalates long prompts. *Cheap-then-escalate* keeps the cheap answer if visible tests pass ($v = 1$), otherwise calls the strong model. *Sequential oracle* chooses the reward-maximizing action with full knowledge of both models' hidden outcomes.

Learned routers. *Logistic oracle imitation* uses prompt features (length, signature, test count, complexity flags) with validation-reward threshold calibration. *ModernBERT* (149M parameters) is a compact verifier-aware encoder baseline. *Qwen3.5-0.8B LoRA router* is our headline learned policy: a LoRA fine-tune of `Qwen/Qwen3.5-0.8B` for binary routing, distinct from the cheap codegen model (Qwen2.5-Coder-7B) and the expensive API model (Qwen 3.7 Max on MBPP). SFT uses a text prompt containing the task, cheap completion, visible score, and error to predict a next-token decision; the model is trained to emit `LOCAL/API` tokens, which we map to `CHEAP/EXPENSIVE` actions. Labels are hindsight-oracle decisions under the cost-sensitive reward in Eq. 1. Because our cached dataset logs both models' hidden outcomes for every task, we treat supervised oracle imitation as the primary learned policy: it is the natural offline contextual-bandit baseline when exploration is unnecessary. Threshold calibration then selects the escalation probability cutoff maximizing validation reward, turning the classifier into a cost-aware routing policy.

REINFORCE. We also train a Bernoulli contextual bandit with a detached batch-mean baseline on prompt-only policies and on the Qwen3.5-0.8B LoRA router warm-started from supervised checkpoints. This is our direct policy-gradient comparison to the course's RL material; it tests whether sampled reward optimization improves on supervised routing in a low-data, sparse-reward regime rather than serving as the headline training method.

5 Experimental Setup

Development benchmark. 427 MBPP-sanitized tasks with one greedy generation per model (T=0, n=1). Cheap model: Qwen2.5-Coder-7B-Instruct (local vLLM). Expensive model for headline binary analysis: Qwen 3.7 Max (OpenRouter). Scores are computed in an isolated Python subprocess with 10 s timeout.

Held-out generalization. EvalPlus HumanEval+ (164 tasks) and MBPP+ (378 tasks) with Qwen2.5-Coder-7B as cheap and Claude Sonnet 4.6 as strong. API cost is normalized so mean strong-model spend is 1.0; local API cost is zero in the headline analysis. We selected the strong model deliberately: an initial EvalPlus run used Qwen2.5-Coder-14B, but the cheap-vs-strong quality gap was too small (the 14B model was only marginally better on MBPP+ and slightly *worse* than 7B on HumanEval+), so we switched to Claude Sonnet 4.6 to recover a realistic cheap-vs-premium gap while keeping evaluation affordable ($0.75 total Sonnet spend across both benchmarks).

Cost grid. Pre-registered $\lambda \in \{0, 0.05, 0.1, 0.2, 0.5, 1, 2, 5\}$. Qwen3.5-0.8B LoRA router operating point: $\lambda = 0.125$.

Seeds and variance. Cached code generations use greedy decoding with a single RNG seed (T=0, n=1, seed 0); all routing experiments replay these fixed completions. Learned-router training varies the initialization seed. For the Qwen3.5-0.8B LoRA router we train five seeds ($\{0, 1, 2, 3, 4\}$) at $\lambda \in \{0.05, 0.1, 0.125, 0.2\}$ and report five-seed means at the headline operating point $\lambda = 0.125$. Frontier-refinement values $\lambda \in \{0.15, 0.175, 0.5\}$ use seed 0 only. Warm-started REINFORCE on the Qwen3.5-0.8B LoRA router at $\lambda = 0.125$ and EvalPlus transfer evaluation both reuse the same five supervised adapters. Compact baselines (logistic oracle imitation, ModernBERT) use seed 0. Paired-bootstrap 95% confidence intervals use 2000 task resamples with RNG seed matched to the training seed when applicable.

Implementation. Code, cached generations, and reproducible scripts are in the project repository. Modal was used for detached Qwen3.5-0.8B LoRA training and EvalPlus strong-model collection. All open-model experiments use greedy decoding (T=0) and a 10 s per-test execution timeout.

Correctness safeguard. During an audit of the Qwen router scoring code we found a right-padding bug: batched next-token evaluation read the logits at position -1, but the `Qwen3.5-0.8B` tokenizer pads on the right, so for shorter prompts in a padded batch position -1 was a padding token rather than the final prompt token. We fixed this to gather each sequence's last non-padding position, added a regression test with unequal-length right-padded inputs, and *re-evaluated all 23 persisted supervised adapters*. The correction materially changed the reported behavior: at $\lambda = 0.125$, five-seed escalation tightened from $27.9\% \pm 31.3\%$ (buggy) to $20.6\% \pm 3.7\%$ (corrected), revealing that the previously apparent unstable transition was largely a scoring artifact. All Qwen router numbers in this report use the corrected scoring.

Pipeline notes. Early collection runs suffered from code-extraction bugs and token-limit truncation that depressed expensive-model pass rates; after fixing extraction, system prompts, and `max_tokens`, local 7B pass rate rose to $\sim$78%. To confirm the output cap was not silently biasing quality, we audited completions against `EXPENSIVE_MAX_TOKENS`=2048: only $4/427$ GPT-5.3 Codex tasks (0.9%) and 0 Claude tasks hit the cap, and below-cap pass rates matched overall pass rates, so the cap is not a material skew for the premium models (we also found that some OpenRouter providers report `completion_tokens` far above the client cap due to hidden reasoning tokens, so billed usage is not a reliable truncation indicator). Token and dollar costs were logged per model (Table 1); API subtotal for four expensive models over 427 tasks was $6.88. Table 1 also reports per-model quality: cost rank does not match quality rank (e.g., GPT-5.3 Codex is cheaper than Qwen 3.7 Max but scores slightly lower), which motivates a router that conditions on more than price. For the headline binary study we pair the strongest local model (Qwen2.5-Coder-7B) as the cheap model with the strongest-quality API model (Qwen 3.7 Max) as the expensive model.

6 Results

6.1 Quantitative Evaluation

6.1.1 MBPP development: routing opportunity and oracle frontier

Figure 1 shows the milestone cost–quality Pareto for Qwen2.5-Coder-7B vs. Qwen 3.7 Max. Across 427 tasks, 343 tasks tie on mean test score; the expensive model strictly wins on 82 tasks—the routing opportunity. The hindsight oracle at $\lambda = 0.5$ reaches 0.94 mean score with 12.9% escalation, dominating all prompt-only baselines (Table 2). Cheap-then-escalate (after visible/hidden rescoring) achieves 0.958 hidden pass rate at 0.257 normalized cost (16.9% escalation; bootstrap 95% CI for pass rate [0.940, 0.975]), nearly matching the oracle.

Table 1: Per-model quality, token usage, and OpenRouter pricing for the six-model MBPP collection (427 tasks each, T=0, n=1). All-pass is the fraction of tasks passing every hidden assertion; mean score is the average fraction of assertions passed. **Bold** rows are the two models selected for the headline binary routing study (cheap: Qwen2.5-Coder-7B; expensive: Qwen 3.7 Max).

Model	All-pass	Mean score	In tok	Out tok	Total $
Qwen2.5-Coder-7B[†]	**78.5%**	**0.83**	52,134	18,943	0.00
Qwen3.5-9B	74.5%	0.79	54,674	25,256	0.00
Claude Sonnet 4.6	91.1%	0.94	52,422	24,685	0.53
GPT-5.3 Codex	94.1%	0.96	49,754	107,599	1.59
Qwen 3.7 Max[‡]	**95.6%**	**0.98**	53,820	587,624	4.54
DeepSeek V4 Pro[*]	65.8%	0.68	49,822	229,800	0.22

[†]Selected cheap model. [‡]Selected expensive model (strongest observed quality). [*]Diagnostic only (extraction/hidden-token anomalies); not used in headline routing.

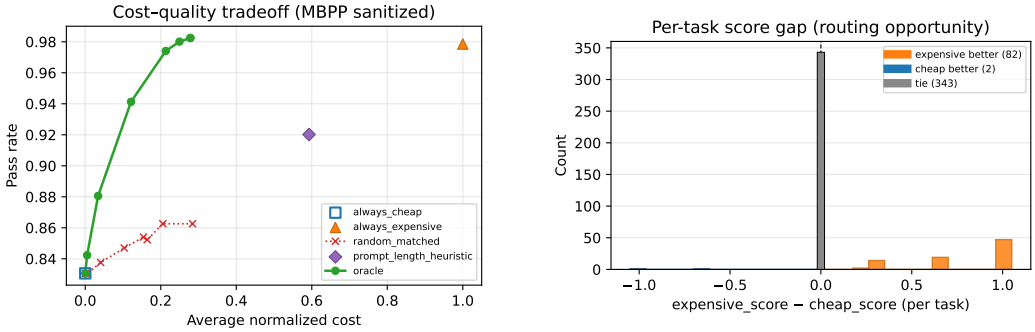

Figure 1: Left: cost–quality Pareto on MBPP-sanitized (7B vs. Qwen 3.7 Max). Right: per-task score gap distribution; escalation helps mainly when expensive strictly outscores cheap.

6.1.2 Learned routers on MBPP

Figure 2 summarizes the Qwen3.5-0.8B LoRA router frontier across five seeds. At $\lambda = 0.125$, the best cost-quality point (seed 3) reaches 0.961 pass rate at 0.224 normalized cost—competitive with cheap-then-escalate (0.958 at 0.257) but with higher seed variance. ModernBERT verifier-aware routing is viable at similar quality but also sensitive to calibration threshold. The Qwen3.5-0.8B LoRA router does not uniformly dominate the deterministic verifier rule; it trades slightly lower cost for comparable quality at some operating points.

Figure 3 shows the full multi-λ Qwen3.5-0.8B LoRA router Pareto with paired-bootstrap variability across seeds. A sharp operating-point transition appears between $\lambda = 0.10$ and $\lambda = 0.20$: calibrated escalation drops from $\sim$82% to $\sim$3.5% of tasks. This is a property of threshold calibration on the cost grid, not a training failure.

6.1.3 EvalPlus generalization with Sonnet 4.6

Figure 4 and Table 3 show held-out EvalPlus results at $\lambda = 0$ with bootstrap 95% CIs. Cheap-then-escalate matches always-Sonnet plus-test quality at 19–23% of normalized API cost.

6.1.4 Cross-benchmark transfer and recalibration

We evaluate MBPP-trained Qwen3.5-0.8B LoRA routers on EvalPlus without recalibration at $\lambda = 0.125$. Seed 0 transfers well (combined score 0.851, cost 0.237), but seed 3 under-escalates (combined score 0.760, 2.6% escalation) because the MBPP validation threshold (0.531) is too conservative on EvalPlus.

Recalibrating the escalation threshold on a 50/50 EvalPlus validation split fixes this failure mode. Across five seeds, mean combined score improves from 0.833 to 0.889 and mean reward from 0.800 to 0.861 (Table 4). For seed 3 specifically, combined score rises from 0.760 to 0.891. Recalibrated

Table 2: Hindsight oracle routing balance across λ (MBPP-sanitized, 7B vs. Qwen 3.7 Max).

λ	Esc. rate	Mean score	Avg. cost	Reward
0	19.2%	0.98	0.28	0.98
0.5	12.9%	0.94	0.12	0.88
1	5.2%	0.88	0.03	0.85
5	0.0%	0.83	0.00	0.83

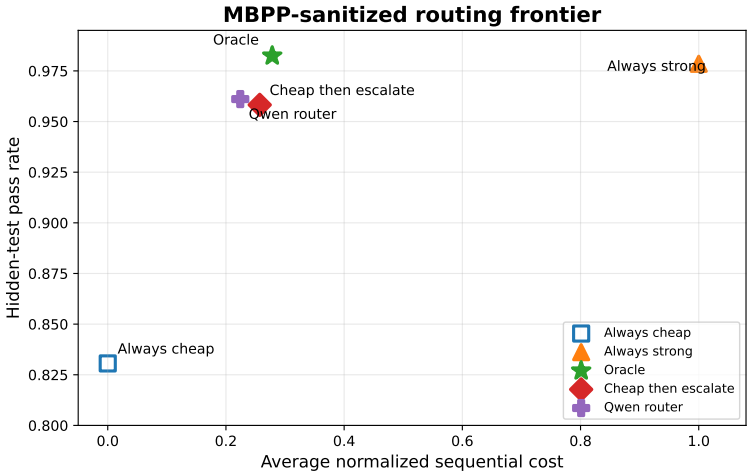

Figure 2: MBPP-sanitized simplified frontier: cheap-then-escalate vs. Qwen3.5-0.8B LoRA router (five seeds) and reference baselines.

Qwen3.5-0.8B LoRA routers still do not beat cheap-then-escalate on reward, confirming that the simple verifier rule remains the practical default.

6.1.5 Local-cost sensitivity

The headline analysis treats local inference as zero API cost. Figure 5 varies local cost as a fraction of mean API cost (0, 5%, 10%, 20%). Because cheap-then-escalate routes on the visible-test outcome alone, its escalation decisions—and hence hidden pass rate—are independent of both λ and the assumed local cost (left panel); only the absolute normalized cost rises linearly. The cost penalty does affect the reward objective: at $\lambda = 0.5$ (right panel), average reward decreases and regret-to-oracle grows modestly as local cost increases (regret 0.051 at the zero-local-cost operating point, bootstrap CI [0.028, 0.079]), but the policy remains close to the oracle throughout. Routing conclusions are therefore robust to assigning local inference a modest nonzero cost.

6.2 Qualitative Analysis

6.2.1 Verifier feedback is the dominant routing signal

The per-task score-gap distribution (Figure 1, right) explains why routing helps at all: on 343/427 tasks both models tie, and the expensive model strictly wins on only 82. Escalation is therefore valuable precisely on the minority of tasks where the cheap model is wrong, and the cheap model's *visible* unit-test outcome identifies most of them: visible-test failure is highly predictive of hidden-score gain from escalation, whereas tasks that already pass visible tests rarely benefit from escalation. By contrast, prompt-only logistic oracle imitation at $\lambda = 0.5$ reaches only 0.864 held-out score with 7.4% escalation (regret 0.150) even after threshold calibration, and a nonlinear prompt MLP and three-model prompt routing behave similarly. **Takeaway:** static prompt features cannot substitute for observing cheap-model execution behavior; the executable verifier is the signal that makes routing work.

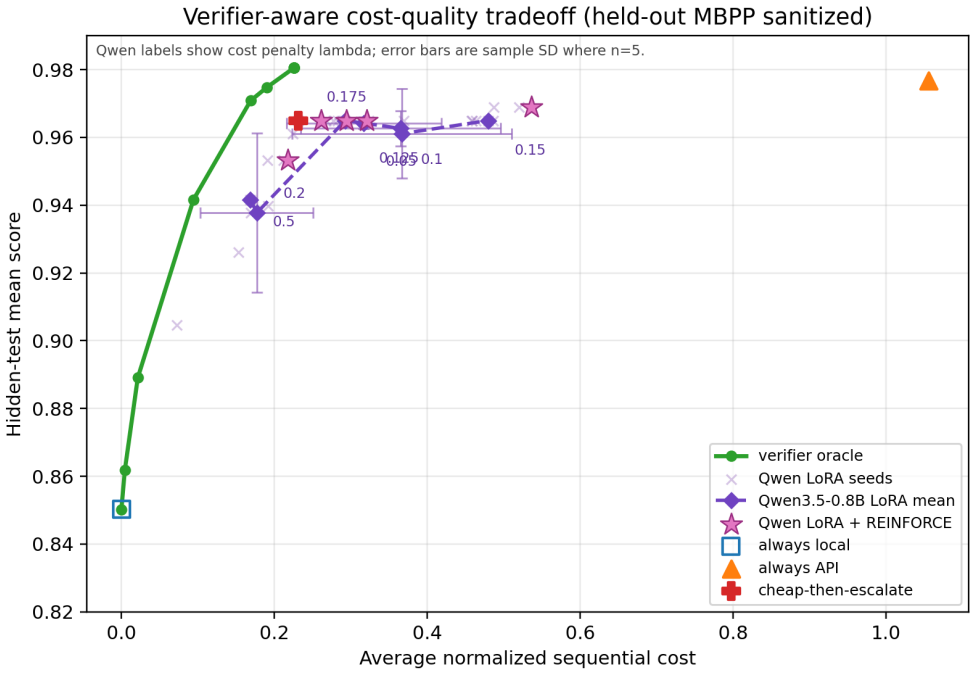

Figure 3: Full Qwen3.5-0.8B LoRA router frontier across λ and five seeds (MBPP held-out test).

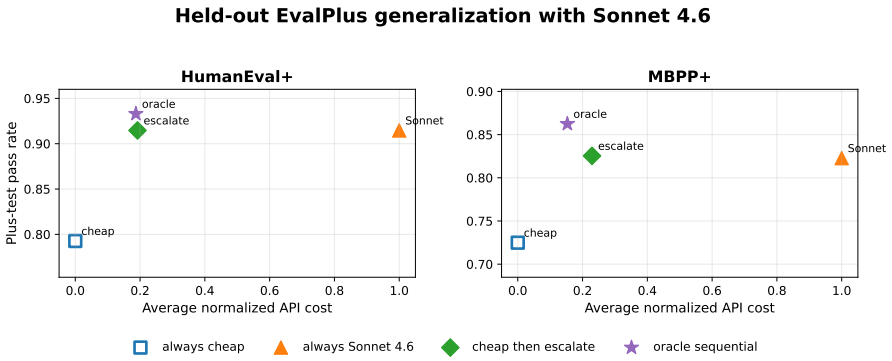

Figure 4: EvalPlus generalization Pareto (HumanEval+ and MBPP+) with Sonnet 4.6 as strong model.

6.2.2 REINFORCE failure modes

Direct policy-gradient training is fragile in this offline, sparse-reward setting. Prompt-only REINFORCE collapses to always-cheap at higher cost penalties ($\lambda \geq 0.5$): sparse expensive-only wins plus a high cost penalty make conservative exploration self-reinforcing. At $\lambda = 0$ it instead over-escalates (always-expensive behavior), wasting cost. Warm-starting from the supervised Qwen3.5-0.8B LoRA router and applying variance-control ablations (lower learning rate, no entropy bonus) does not produce a robust improvement: at $\lambda = 0.125$ the default REINFORCE run is marginally *worse* than supervised SFT on five-seed mean reward (Table 5; 0.923 vs. 0.925), and the best ablation variant we tried stays within bootstrap noise ($|\Delta\text{reward}| < 0.002$). Figure 6 shows regret-to-oracle rising sharply for always-expensive as λ grows, while the oracle stays low across the grid. **Interpretation:** when both models' outcomes are logged, supervised oracle imitation is the appropriate default; naive policy gradients add instability without improving the frontier.

Table 3: EvalPlus held-out results at $\lambda = 0$ with paired-bootstrap 95% CIs on pass rate.

Dataset	Policy	Score	Cost	Esc.
HumanEval+	always cheap	0.793 [0.732, 0.854]	0.000	0%
HumanEval+	always Sonnet	0.915 [0.872, 0.957]	1.000	100%
HumanEval+	cheap-then-escalate	**0.915** [0.872, 0.957]	0.192	14.6%
HumanEval+	oracle sequential	0.933 [0.896, 0.970]	0.187	14.0%
MBPP+	always cheap	0.725 [0.680, 0.772]	0.000	0%
MBPP+	always Sonnet	0.823 [0.786, 0.860]	1.000	100%
MBPP+	cheap-then-escalate	**0.825** [0.788, 0.862]	0.229	17.7%
MBPP+	oracle sequential	0.862 [0.828, 0.897]	0.153	13.8%

Table 4: Qwen3.5-0.8B LoRA router cross-benchmark transfer ($\lambda = 0.125$, five-seed mean). MBPP threshold vs. EvalPlus-recalibrated threshold on held-out test split.

Dataset	MBPP threshold		Recalibrated	
	Score	Reward	Score	Reward
HumanEval+	0.893	0.859	0.932	0.898
MBPP+	0.806	0.774	0.866	0.840
Combined	0.833	0.800	0.889	0.861

6.2.3 Cross-benchmark calibration shift

The Qwen3.5-0.8B LoRA router's main failure mode under distribution shift is calibration, not representation. The same Qwen3.5-0.8B LoRA adapter that transfers well on seed 0 (combined score 0.851) under-escalates badly on seed 3 (0.760 score, 2.6% escalation) because the MBPP-selected probability threshold (0.531) is far too conservative for the EvalPlus score distribution. Recalibrating the threshold on held-out validation tasks restores performance (seed 3: 0.760→0.891), confirming that the router's escalation *ordering* transfers but its operating point does not. This is the key practical caveat for deploying Qwen3.5-0.8B LoRA routers across benchmarks.

7 Discussion

Our results support a clear hierarchy of routing signals for code generation. **First**, executable verifier feedback from a cheap first attempt is the strongest signal: cheap-then-escalate is simple, interpretable, and near-oracle on both development and held-out benchmarks. **Second**, prompt-only features are insufficient even with calibration and nonlinear models. **Third**, the Qwen3.5-0.8B LoRA router can learn a frontier but requires seed aggregation, threshold calibration, and recalibration when benchmarks shift. **Fourth**, REINFORCE is not automatically better in offline sparse-reward routing; supervised methods are more stable. From a course perspective, the project still applies the contextual-bandit toolkit end-to-end—reward design, regret, λ sweeps, and policy-gradient training—but the main scientific conclusion is that offline full-information routing favors supervised oracle imitation over naive REINFORCE.

These findings align with CS224R novelty criteria: we answer an open question about verifier-aware routing, adapt contextual bandits to a non-trivial code-generation domain, and analyze failure modes when learned methods underperform simple rules.

8 Limitations and Future Work

Local cost modeling. We primarily report zero local API cost; sensitivity analysis shows pass rates are unaffected but absolute cost shifts. Future work should include GPU amortization and latency constraints in the objective.

Single strong model. We study binary cheap-vs-expensive routing. A cached three-model oracle over $\{7B, 9B, API\}$ shows that adding local 9B can improve the hindsight frontier on some tasks, but binary 7B-vs-expensive routing already captures most of the achievable gain; full multi-tier learned routing is left for future work.

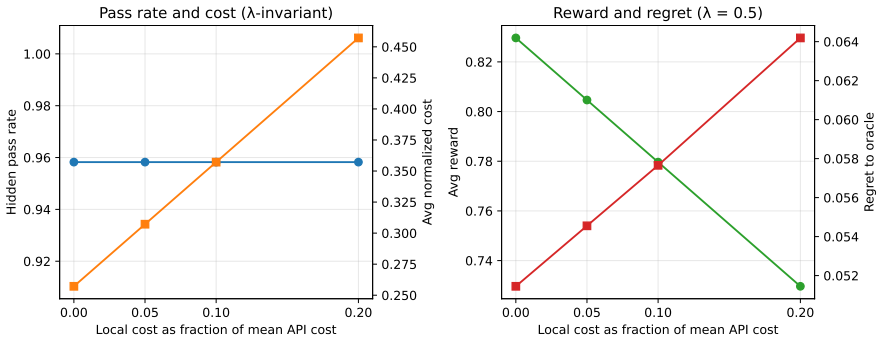

Figure 5: Sensitivity of cheap-then-escalate to nonzero local cost (fraction of mean API cost). Left: hidden pass rate is flat (routing is λ-invariant) while normalized cost scales linearly. Right: at $\lambda = 0.5$, average reward declines and regret-to-oracle grows only modestly, so routing conclusions are robust to a small nonzero local cost.

Figure 6: Regret-to-oracle vs. λ on MBPP-sanitized (7B vs. Qwen 3.7 Max). Always-expensive accumulates regret as cost penalty increases.

Benchmark scope. Results are strongest for Python unit-test benchmarks; domains without executable verifiers may not benefit equally.

Router training. Reward-gap-weighted SFT and multi-benchmark joint training may reduce calibration shift; multi-step sequential RL is an extension beyond this report.

9 Conclusion

This work studied whether a learned router can approach strong-model code-generation quality while paying for the expensive model only when necessary, and whether the executable feedback unique to code makes routing easier than in the prompt-only LLM-routing literature. Framing the problem as a cost-sensitive contextual bandit over a frozen cheap model and a frozen strong model, we compared deterministic baselines, prompt-only and verifier-aware supervised routers, a compact ModernBERT classifier, a Qwen3.5-0.8B LoRA router, and REINFORCE policy gradients on 427 MBPP-sanitized tasks and on held-out EvalPlus benchmarks.

Our central finding is a clear hierarchy of routing signals. Executable *visible* verifier feedback from a cheap first attempt is the dominant signal: the simple cheap-then-escalate rule nearly matches the hindsight oracle on MBPP (0.958 hidden pass rate at 0.257 normalized cost) and matches Claude Sonnet 4.6 quality on EvalPlus at roughly 19–23% of API cost, while dominating every prompt-only learned router. Static prompt features are insufficient even with nonlinear models and threshold calibration. The Qwen3.5-0.8B LoRA router can trace a competitive cost–quality frontier but is seed-sensitive, calibration-dependent, and—after cross-benchmark threshold recalibration—still does not

Table 5: Supervised vs. REINFORCE Qwen3.5-0.8B LoRA router at $\lambda = 0.125$ (five-seed mean on MBPP held-out test). Prompt-only REINFORCE is not in the pre-registered grid at this λ; it collapses to always-cheap at $\lambda \geq 0.5$.

Method	Pass rate	Avg. cost	Reward	Esc. rate
Supervised Qwen3.5-0.8B LoRA	0.964	0.317	0.925	0.21
Qwen3.5-0.8B LoRA + REINFORCE	0.963	0.326	0.923	0.21

beat the simple verifier rule on reward. Direct REINFORCE either collapses to always-cheap under a high cost penalty or, when warm-started from the supervised router, fails to produce a robust gain, a useful negative result for sparse-reward offline routing.

These results carry a concrete practical recommendation: *deploy verifier-aware escalation first*, and treat a learned router such as the Qwen3.5-0.8B LoRA model as an optional refinement that must be seed-aggregated and re-calibrated per benchmark before it is trusted. More broadly, the project illustrates a transferable lesson for applying reinforcement-learning machinery to a new domain: when an environment exposes a cheap, reliable verifier and both actions' outcomes can be logged offline, exploiting that verifier and supervised oracle imitation can outperform direct policy-gradient optimization. Future work should incorporate GPU-amortized local cost and latency into the objective, extend the action space to a learned multi-tier router over {7B, 9B, API} (a setting where the hindsight oracle already shows room to halve API cost), apply reward-gap-weighted or multi-benchmark joint training to reduce calibration shift, and test verifier-aware routing in domains beyond Python unit tests.

Team Contributions

Mauricio Berlanga (solo project). All work—problem formulation, data collection, evaluation harness, baselines and learned routers (ModernBERT, Qwen3.5-0.8B LoRA, REINFORCE), EvalPlus experiments, figures, and this report—was completed individually.

AI Tools Disclosure

ChatGPT and Claude were used for boilerplate scaffolding, debugging suggestions, and writing assistance; the bandit formulation, experiments, analysis, and primary writing were developed independently and reviewed by the author.

References

Jacob Austin et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. In *International Conference on Learning Representations*, 2024.

Mark Chen et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Jiawei Liu, Chunqiu Steven Xia, Yuxuan Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 2023.

Isaac Ong et al. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.