

Cost-Aware Escalation from Scalar Reward Models to Generative Models

Stanford CS224N Custom Project

Rui Chen

Department of Computer Science
Stanford University
e0659578@stanford.edu

Landon Choy

Department of Computer Science
Stanford University
landonc@stanford.edu

Cole Yarbrough

Department of Computer Science
Stanford University
cyarb27@stanford.edu

Abstract

The increasing computational demands of reward models for aligning large language models with human preferences call for more efficient inference strategies that balance cost-accuracy trade-offs. While scalar reward models (which output just a numerical score per response) enable lightweight and lower-cost inference, they generally struggle on complex prompts. On the other hand, generative reward models (which output a text judgment with the preferred response) are typically more robust and handle complex prompts better, but come at a greater computational cost. To balance their strengths and shortcomings, we propose *Learn2Defer-RM*: a 2-stage reward modeling framework leveraging a neural router trained on the Skywork dataset, which performs instance-wise escalation from a cheap scalar reward model to a more robust generative model based on explicit scalar features and implicit embeddings. Our goal is to match or surpass the generative reward model’s accuracy while using it sparingly. We found that our learned router with a fixed lambda (where lambda penalizes escalation) outperforms main baselines in overall accuracy on the RM-Bench test set, and escalation rates can be constrained by adjusting lambda. Additional findings show that the wide router (using the explicit scalar features) captures most of the routing signals compared to the deep router (using the implicit embeddings). Furthermore, a Lagrangian method provides more explicit and interpretable cost control, though it faces generalization challenges on the RM-Bench test set.

1 Key Information to include

- Mentor: David Anugraha
- External Collaborators (if you have any): N/A
- Sharing project: N/A
- Late Days (if you are using any):

Team member	Late days remaining
Landon Choy	5
Cole Yarbrough	1
Rui Chen	0
Total pooled	6
Team size	3
Late days applied for the report	2

2 Introduction

Preference judgments are critical for aligning large language models with human values. In this setting, a model is given a prompt and two candidate responses—one labeled preferred by a human annotator—and must predict which response should be chosen across diverse domains and difficulty levels. However, both humans and models can be affected by *style–substance bias*, overvaluing superficial attributes such as length and fluency while undervaluing correctness and substantive reasoning. Generative reward models tend to mitigate this bias more effectively than scalar reward models because they must produce an explanation of the preference decision rather than outputting only a numerical score. For example, the generative judge DeepSeek-R1 achieves 78.9% accuracy on the hard subset of RM-Bench, where style and substance are intentionally in conflict, but at substantially higher inference cost. In contrast, even strong scalar reward models such as Skywork-Reward-Llama-3.1-8B reach only 46.6% accuracy on the same subset [1, 2].

To address this trade-off, we introduce Learn2Defer-RM, which performs instance-wise routing between a cheap scalar reward model (Stage I) and a stronger generative reward model (Stage II). Both evaluators are kept fixed. We train a lightweight neural router to decide whether a given preference pair can be handled by Stage I or should be escalated to Stage II, using features derivable from stage I together with text embeddings of prompts and responses. Our main formulation is a cost-regularized learn-to-defer objective with a fixed escalation penalty λ , while we develop a Lagrangian variant which serves as an extension for explicit budget control.

We evaluate this framework on offline preference-routing data and test whether our proposed framework can improve the preference accuracy / computational cost trade-off over always using either model alone. We further study how the trade-off changes across escalation penalty values, whether constrained optimization yields more interpretable budget control, and which routing signals are most informative through feature ablations. Together, these experiments position learned deferral as a practical mechanism for achieving stronger preference judgment under controllable computational cost.

3 Related work

Reward modeling and preference evaluation Reward modeling is a core component of modern preference-based alignment pipelines, where models are trained to predict which of two candidate responses is preferred by human [3]. Early work in this area largely focused on scalar reward models that assign a single score to each response, enabling efficient pairwise comparison at inference time. More recent work has emphasized that reward-model quality depends strongly on the evaluation setting: RewardBench broadened evaluation across diverse domains, while RM-Bench highlighted a more challenging regime in which preferred and rejected responses differ only subtly with style cues that can mislead scalar models [4, 2]. In parallel, generative or reasoning-based evaluators such as R3 have been proposed to produce more robust and interpretable pairwise judgments [5]. These works motivate our two-stage setup directly: we use a cheap scalar reward model as an efficient first-stage evaluator and a stronger generative reward model as a more reliable but more expensive second-stage verifier. A closely related recent direction routes uncertain examples from a fast reward model to a stronger judge [1].

Learning to defer and selective prediction A second line of work studies when a model should abstain from predicting on its own and instead defer to another decision-maker. Madras et al. [6] introduced learn-to-defer as a framework for jointly optimizing automated predictions and downstream handoff decisions. Related work in selective prediction, such as SelectiveNet, similarly studies how to trade predictive performance against the decision to abstain [7]. More recent extensions consider multi-stage or multi-expert settings, including two-stage defer systems and improved surrogate losses for routing decisions [8, 9]. These works provide the main conceptual foundation for our approach. Our fixed- λ objective can be viewed as a cost-regularized learn-to-defer formulation specialized to pairwise preference judgment, where the router decides whether to keep the scalar reward-model prediction or defer to a stronger generative evaluator.

Budgeted and constrained inference A third relevant line of work studies prediction under explicit resource constraints. Rather than optimizing accuracy alone, these methods aim to control quantities such as abstention rate, coverage, or compute usage. In constrained optimization for machine learning,

Lagrangian or proxy-Lagrangian methods are commonly used to enforce such rate constraints in non-convex settings [10]. This perspective is directly relevant to our budget-control extension. While our main method uses a fixed escalation penalty λ to trace an accuracy–cost trade-off, our Lagrangian variant instead treats heavy-model usage as an explicit average-budget constraint and adapts the dual variable during training. Thus, prior work on constrained inference does not define our main formulation, but it provides the natural framework for interpreting and motivating our budgeted extension.

4 Approach

Problem setup Our starting point is a standard preference dataset

$$\mathcal{D} = \{d_i\}_{i=1}^N = \{(q_i, a_{i1}, a_{i2})\}_{i=1}^N,$$

where for the i^{th} preference pair, q_i is a query, and a_{i1}, a_{i2} are two candidate responses to q_i . Although the raw preference data are stored as (q_i, a_i^+, a_i^-) , with a_i^+ denoting the chosen response and a_i^- the rejected one, our training formulation uses the displayed order (a_{i1}, a_{i2}) after optional swapping. Thus, either a_{i1} or a_{i2} may be the preferred response, and we define a binary label y_i later as whether the first displayed response is preferred.

The underlying task is *preference judgment*: given a query and two candidate responses, the system must determine which response is more aligned with human preference. In our setting, the primary task metric is pairwise accuracy, i.e., the fraction of examples on which the system correctly identifies the preferred response.

2-stage reward modeling Given that scalar RM and generative RM have different strengths and costs, we attempt to approach the problem using a 2-stage reward modeling framework:

- Stage I: a cheap scalar RM (denoted C) handling easier to distinguish examples.
- Stage II: a robust but heavier generative RM (denoted H) handling examples hard to distinguish (i.e. d_i ’s having stronger style bias).
- An originally-designed adaptive router $s_\theta(x_i)$ (where $x_i = \phi(d_i, C)$ is information extracted from d_i and the stage I output) that learns when to escalate from C to H .

Our goal is to leverage the proposed framework to achieve performance comparable to always using heavy generative verifiers on preference judgment tasks, but at a much lower cost. Such a framework has practical applications in various domains by letting the system decide when to allocate more compute, depending on the complexities of tasks to solve.

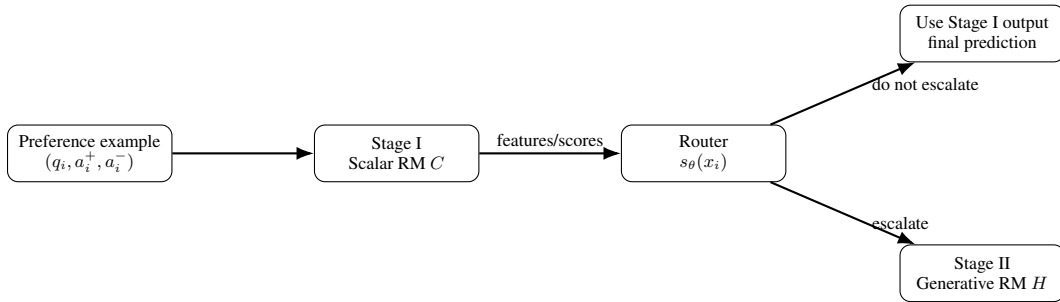


Figure 1: Two-stage routing framework. A scalar reward model first processes the preference pair. The router then decides whether to keep the Stage-I prediction or escalate the example to a stronger generative reward model.

Learn-to-Defer framework: Learn2Defer-RM We model the framework as a learn-to-defer problem [6] on pairwise preference labels $y_i = I(a_{i1} \succ a_{i2}) \in \{0, 1\}$. Our method, Learn2Defer-RM, combines two frozen components: a scalar reward model, *RM-Mistral-7B* [11], and a more computationally expensive generative model, *R3-Qwen3-14B-14k* [5].

Let $p_i^{(C)} = p^{(C)}(a_{i1} \succ a_{i2})$ and $p_i^{(H)} = p^{(H)}(a_{i1} \succ a_{i2})$ denote the preference probabilities generated from the cheap and heavy models, a learned router $s_\theta(x_i)$ produces an escalation probability $\pi_\theta(x_i) \in [0, 1]$, indicating the router’s confidence to defer to the heavy model. During training, we use a soft mixture probability $p_i^{(\text{sys})} = (1 - \pi_\theta(x_i))p_i^{(C)} + \pi_\theta(x_i)p_i^{(H)}$, allowing end-to-end

optimization while keeping both reward models fixed. Following [6], we train the router to balance accuracy and compute via

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^N \mathcal{L}_{\text{BCE}} \left(y_i, p_i^{(\text{sys})} \right) + \lambda \frac{1}{N} \sum_{i=1}^N c_i \pi_{\theta}(x_i),$$

where c_i is the heavy-call cost, and λ is a hyperparameter which controls the accuracy–compute tradeoff. We additionally introduced the cost term comparing to original framework as our main contribution, where we expect a larger λ to discourage escalation, promote reliance on the cheap model unless deferral improves accuracy. In actual implementation, we set all c_i as 1 for better interpretation, so the cost term $\lambda \frac{1}{N} \sum_{i=1}^N c_i \pi_{\theta}(x_i)$ simplifies to $\lambda \frac{1}{N} \sum_{i=1}^N \pi_{\theta}(x_i)$.

At inference time, we convert the soft gate into a deterministic decision by thresholding $\pi_{\theta}(x_i)$ at τ : if $\pi_{\theta}(x_i) > \tau$, the system escalates to the heavy reward model; otherwise, it uses the cheap scalar one. In this way, the adaptive router will be defined as $s_{\theta}(x_i) = \mathbf{1}(\pi_{\theta}(x_i) > \tau)$

Wide-and-deep router features For choice of $x_i = \phi(d_i, C)$, we use a *wide-and-deep* feature design for the router to capture two complementary types of information. The wide features are explicit, low-dimensional signals derived from responses and Stage-I scalar reward model outputs, including the chosen and rejected reward scores, their margin, and response-length differences. These features are inexpensive, interpretable, and closely tied to routing-relevant notions such as confidence and ambiguity. The deep features are dense text embedding representations of the query-response pair, intended to capture higher-level semantic or stylistic patterns that may not be reflected in scalar scores alone. This design allows the router to benefit from both strong hand-crafted routing cues and richer contextual information, while our subsequent ablations evaluate how much each feature family contributes in practice.

Softening generative RM outputs Generative reward models typically produce textual judgments rather than calibrated probabilities. For example, a generative reward model may output statements such as “Response A is better” or “Response B is better”. However, our routing objective requires a probability estimate p_i^H that can be combined with the output of the scalar reward model.

To bridge this mismatch, we first parse the textual output of the generative verifier using the same response parser provided in the R3 model’s codebase [5] to reliably extract which response is preferred. The parsed decision is then converted into a softened probability signal. When the verifier selects response a_{i1} , we map the output to a high preference probability; when it selects response a_{i2} , we map it to a low probability. In cases where the verifier output is ambiguous or cannot be reliably parsed, we assign a neutral probability value.

Formally, we define

$$p_i^{(H)} = \begin{cases} 0.8 & \text{if the verifier prefers } a_{i1}, \\ 0.2 & \text{if the verifier prefers } a_{i2}, \\ 0.5 & \text{if the output is ambiguous.} \end{cases}$$

This softened representation allows us to integrate generative judgments into the probabilistic mixture

$$p_i^{(\text{sys})} = (1 - \pi_{\theta}(x_i))p_i^{(C)} + \pi_{\theta}(x_i)p_i^{(H)},$$

while avoiding unstable gradients that may arise from hard binary targets.

Lagrangian approach: Learn2Defer-RM-lag Rather than treating compute control as a fixed regularization heuristic, we also consider a constrained formulation in which the average escalation rate is explicitly bounded by a target budget ρ . Let $\ell_i(\theta)$ denote the BCE loss of the final mixed prediction on example i . We then solve

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^N \ell_i(\theta) \quad \text{subject to} \quad \frac{1}{N} \sum_{i=1}^N \pi_{\theta}(x_i) \leq \rho.$$

We optimize this problem using a Lagrangian relaxation,

$$J(\theta, \lambda) = L_{\text{BCE}}(\theta) + \lambda \left(\frac{1}{N} \sum_{i=1}^N s_{\theta}(x_i) - \rho \right), \quad \lambda \geq 0,$$

and solve the resulting min-max problem over θ and λ . Intuitively, λ acts as an adaptive price on escalation: when the router exceeds the target budget, the penalty increases; otherwise, it weakens. We treat this formulation as a budget-control extension to our main fixed- λ router. We refer to this variant as Learn2Defer-RM-lag. Refer to Appendix B.1 for more details.

Baselines During evaluation, we compare our learned routers against three kinds of baselines: single-stage baselines (always use cheap models and always use heavy models), simple non-learned combinations of Stage I and Stage II (mean aggregation and learned linear combination), and random escalation policy at fixed probability of 50%. Our main method is the fixed- λ router, where we evaluate the Lagrangian formulation as a budget-control extension. All baselines and learned routers are evaluated using the same inference and evaluation pipeline.

5 Experiments

We evaluate learned routing along four axes: overall comparison with baselines, the accuracy-cost trade-off induced by fixed- λ routing, escalation behavior across domains and difficulty levels, and explicit budget control with the Lagrangian variant. We summarize only the essential experimental setup in the main text and defer implementation details and supplementary results to the appendix.

5.1 Experiment setup

Data We train the router on an offline routing dataset constructed from the *cleaned Skywork v0.1 Reward Data Collection*¹ [12] with an 80/20 train-dev split stratified by source. We evaluate our learned model on *RM-Bench*² [2] and report accuracy and escalation rate across different domains and difficulties, as well as robustness via the *style gap* (Easy-Hard). Each routing example consists of a prompt, two displayed candidate responses after optional swapping, precomputed outputs from the Stage-I scalar model and Stage-II generative verifier, corresponding implicit and explicit features required by router, and a binary target indicating whether the first displayed response is preferred. Refer to Appendix C.1 for more details.

Metrics Our primary metrics are overall pairwise accuracy, i.e., the fraction of examples on which the final routed system identifies the preferred response correctly, and escalation rate, which serves as a proxy for compute cost. On RM-Bench, we also report per-domain / per-difficulty accuracy and style gap (Easy - Hard). For the Lagrangian variant, we additionally track realized vs. target escalation and % of tokens saved. Refer to Appendix C.4 for more details.

Experiment details The offline routing dataset is pre-computed by running both frozen models (the scalar RM-Mistral-7B and the generative verifier R3-Qwen3-14B-14k) over the Skywork dataset. Constructing this offline dataset took approximately 16 hours. We then extracted the explicit wide features and implicit text embeddings using ModernBERT [13], which required an additional 3 hours on a single NVIDIA A100 (80GB) GPU.

Training of learned routers uses AdamW (lr = 0.001, batch size 256) for up to 100 epochs with early stopping on dev accuracy. At inference time, we threshold the escalation probability at $\tau = 0.5$, so that we escalate to heavy reward model if $\pi_\theta(x_i) > 0.5$. Training took roughly 10 minutes on one A100 (80GB) for each baseline and learned router we evaluate. Refer to Appendix C.5 for more details.

5.2 Results

Learn2Defer-RM comparison with baselines We begin by asking whether the learned routing improves over natural baselines. Table 1 shows that the learned router consistently outperforms both single-stage baselines and simple non-learned combinations of Stage I and Stage II. In particular, the always-cheap scalar baseline reaches 0.6632 overall accuracy, while the always-heavy generative baseline R3 reaches 0.8657. By comparison, the fixed- λ learned router achieves 0.8915 at $\lambda = 0.1$, which is the strongest result among the methods shown. Simple static combinations are much less effective: mean and linear combinations of Stage I and Stage II achieve only 0.6679 and 0.6807,

¹Cleaned Skywork Reward Data Collection can be accessed at <https://huggingface.co/datasets/natolambert/skywork-preferences-80k-v0.1-cleaned>.

²RM-Bench dataset can be accessed at <https://huggingface.co/datasets/THU-KEG/RM-Bench>.

respectively, and even random escalation at a 50% rate reaches only 0.7641. These results suggest that the main benefit comes from instance-wise routing rather than from a static combination of the two evaluators. In particular, the gap between the learned router and the simple combination baselines indicates that the cheap and heavy evaluators are complementary, and that this complementarity is best exploited by deciding adaptively when escalation is worthwhile.

Category	Method	Overall Accuracy
Single-stage	Always heavy (R3)	0.865662
Single-stage	Always cheap (RM-Mistral-7B)	0.663212
Learn2Defer-RM	Fixed- λ router ($\lambda = 0.1$)	0.891541
Learn2Defer-RM	Fixed- λ router ($\lambda = 0.2$)	0.870436
Simple combination	Mean(Stage I, Stage II)	0.667923
Simple combination	Linear(Stage I, Stage II)	0.680653
Random escalation	escalation at 50% probability	0.764070

“Always cheap” uses only the Stage-I scalar reward model; “always heavy” uses only the Stage-II generative verifier.

“Mean” and “Linear” denote non-learned combinations of Stage I and Stage II.

Table 1: Overall baseline comparison on RM-Bench. Learned routing (Learn2Defer-RM) outperforms both single-stage baselines and simple non-learned combinations.

Accuracy–Cost trade-off under fixed- λ routing Having established that learned routing outperforms the baselines, we next examine whether this advantage can be controlled through the fixed- λ objective. Figure 2 shows how both overall accuracy and escalation rate vary as the cost penalty λ increases. As λ grows, the router escalates less frequently to the heavy generative verifier, and the escalation rate decreases steadily from 0.3303 at $\lambda = 0.0$ to nearly zero at $\lambda = 2.0$. This reduction in heavy-model usage is accompanied by a gradual drop in overall accuracy, from 0.8857 to 0.6632, tracing out a clear accuracy–cost frontier.

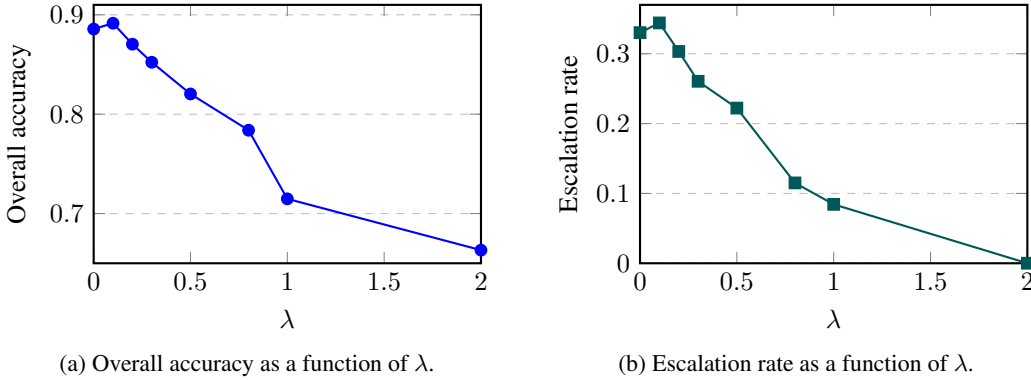


Figure 2: Accuracy–cost trade-off under fixed- λ routing. Increasing λ reduces escalation to the heavy model, while also lowering overall accuracy.

Performance across domains and difficulty levels We next ask where the router chooses to spend its heavy-model budget. Tables 2 and 3 in Appendix A.1 show that the gains from learned routing are concentrated in the more difficult subsets of RM-Bench, and that the router’s escalation behavior is aligned with this pattern. On the accuracy side, Learn2Defer-RM consistently improves over the always-cheap baseline across all domains, with especially large gains on the hard subsets of math, code, and chat. For example, on hard math the scalar baseline achieves 0.296 accuracy, while Learn2Defer-RM reaches 0.890; on hard code the corresponding improvement is from 0.216 to 0.736; and on hard chat it improves from 0.147 to 0.478.

This performance pattern is mirrored by the escalation rates. The learned router escalates much more aggressively on hard chat, code, and math examples than on the easier subsets or the safety domains. For $\lambda = 0.1$, the escalation rate on hard chat, code, and math reaches 0.862, 0.774, and 0.723, respectively, whereas escalation remains much lower for easier subsets and safety-oriented categories. This alignment between performance gains and escalation behavior suggests that the router is allocating heavy-model calls to the parts of the benchmark where the cheap scalar model is most likely to fail.

A similar trend appears in the style-gap columns: relative to the scalar baseline, learned routing substantially reduces the easy-hard gap in math, code, and chat, indicating improved robustness on the more subtle and style-confounded subsets of RM-Bench.

Learn2Defer-RM-lag performance While increasing a fixed compute regularization coefficient λ provides a mechanism to reduce the escalation rate, selecting an appropriate value requires manual tuning and offers only indirect control over the resulting compute budget. To address this limitation, we next consider a Lagrangian formulation that directly optimizes for a desired escalation rate. In this approach, the escalation budget is treated as a constraint and a dual variable dynamically adjusts the effective regularization strength during training. This allows the router to automatically adapt the penalty on escalation so that the resulting policy more closely matches a specified compute budget.

Table 5 in Appendix B.2 summarizes the performance of the Learn2Defer-RM-lag router across the Skywork train/dev splits and the RM-Bench test set as the target escalation budget increases. On RM-Bench, accuracy generally improves as the router is allowed to defer more often, reflecting increased reliance on the stronger generative verifier. Starting from the scalar-only baseline (0.723), performance rises with higher escalation and peaks at 0.877 when the realized escalation rate reaches approximately 0.35, outperforming the generative-only baseline (0.866) while still saving more than half of the generated tokens. Interestingly, the scalar-only baseline slightly outperforms the smallest escalation budget (0.05 target), suggesting that very small amounts of escalation may initially introduce noise before the router learns to defer effectively at higher budgets.

The behavior differs on the Skywork train and development splits. The scalar-only baseline achieves the highest accuracy on both train (0.905) and dev (0.904), while the generative-only baseline performs worst (0.763 train, 0.759 dev), likely reflecting stronger alignment between the scalar model and the Skywork training distribution. Additionally, escalation rates on train and dev closely match the target budgets, indicating that the Lagrangian optimization successfully enforces the constraint during training. In contrast, escalation rates on RM-Bench exceed the targets (e.g., a 0.20 target yielding 0.35 escalation), suggesting that under distribution shift the router defers more frequently because the scalar model is less reliable. Overall, these results indicate that the Lagrangian formulation effectively controls escalation during training while allowing the routing policy to adaptively escalate more often on harder evaluation data.

6 Analysis

Learn2Defer-RM The patterns in Tables 2 and 3 in Appendix A.1 suggest that the router is learning more than a generic tendency to escalate. Instead, it appears to identify cases in which the cheap scalar reward model is likely to fail and the heavy generative verifier is more likely to add value. In this sense, the router is exploiting *instance-wise complementarity* between the two evaluators: the scalar model remains effective on many easy examples, while the generative verifier is most useful on difficult, style-confounded subsets. This helps explain why learned routing outperforms both static single-stage baselines and simple non-learned combinations.

We next analyze which feature family drives routing performance, for which we evaluate 3 variants of our router: using wide features only, using deep features only, and full model using both wide and deep features. We compare their performance with random-escalation baseline.

The ablation results in Table 4 (Appendix A.2) suggest that routing performance is driven primarily by explicit scalar signals rather than by deep text embeddings, which differs from our initial expectations. The wide-only router already achieves 0.880 accuracy, close to the full model’s 0.887, whereas the deep-only variant reaches only 0.7673, only slightly above random escalation (0.7641). We believe this is because the routing task is fundamentally a defer decision –i.e., predicting when the cheap scalar reward model is likely to fail and when the heavy verifier is likely to add value. In our setting, explicit features such as reward scores, score margins, and response-length asymmetries provide direct signals of confidence and ambiguity, which make them highly aligned with the routing objective. By contrast, frozen deep embeddings capture broader semantic information, but are not explicitly optimized to predict scalar-model failure or verifier gain. This makes them useful as a complementary signal, but not as the primary driver of routing performance.

Learn2Defer-RM-lag The training dynamics of Learn2Defer-RM-lag under different target escalation budgets are shown in Figure 3 (Appendix B.3). The Lagrange multiplier λ rises rapidly

early in training and then plateaus, indicating convergence to a regime where the escalation constraint is approximately satisfied; lower escalation targets require larger λ to penalize deferral. The router closely matches the desired escalation rates on the training set, while development metrics exhibit higher variance due to discrete routing decisions evaluated on relatively small validation batches. Across RM-Bench domains (Table 6, Appendix B.4), increasing escalation generally improves accuracy on harder examples by selectively invoking the generative verifier while still achieving substantial token savings. In easier safety domains, performance saturates quickly with minimal escalation (e.g., $< 15\%$ in *safety-response* and $< 10\%$ in *safety-refuse*), whereas reasoning-heavy domains such as *math*, *code*, and *chat* benefit from larger budgets (roughly 40–50%), substantially improving hard-set accuracy and reducing the Easy–Hard style gap.

Table 7 (Appendix B.4) further analyzes routing architectures under the Lagrangian objective using three feature configurations: *wide+deep*, *deep-only*, and *wide-only*. The *wide-only* router uses only scalar reward model features, while *deep-only* relies solely on MODERNBERT embeddings. Although *wide-only* often achieves the highest RM-Bench accuracy at larger escalation budgets, it substantially exceeds the intended escalation rates, indicating weaker constraint control. In contrast, *deep-only* yields the lowest accuracy overall, suggesting that embeddings alone are insufficient for reliable routing. The *wide+deep* configuration provides the best balance between accuracy and escalation control, combining semantic and scalar signals to achieve competitive performance while respecting the compute budget more closely.

7 Conclusion

This work introduced Learn2Defer-RM, a cost-aware learn-to-defer framework that escalates from a cheap scalar reward model to a more robust generative verifier only when the expected accuracy gain justifies the additional cost. By training a lightweight neural router on explicit scalar features and implicit embeddings, we showed that instance-wise escalation can match or exceed the performance of always using the expensive generative model while invoking it only a fraction of the time. Experiments on RM-Bench demonstrate that the learned router produces strong accuracy–cost trade-offs. With a fixed regularization coefficient λ , the router outperforms several strong baselines while allowing controllable compute usage, and ablation studies indicate that explicit scalar features capture most of the effective routing signal, while implicit embeddings contribute less directly to routing decisions but help stabilize cost control.

We further explored a Lagrangian formulation that directly optimizes for a target escalation budget by dynamically adjusting the regularization strength during training. The results show that the router learns meaningful escalation behavior under targeted escalation rates and adapts to task difficulty, though escalation rates can exceed desired targets under distribution shift.

Overall, Learn2Defer-RM demonstrates that learned deferral is an effective strategy for balancing preference judgment accuracy with controllable computational cost. Future work could include alternative definitions of cost, exploring weighted training objectives for different data sources, ensembles of scalar reward models for Stage I routing, and more robust methods for maintaining compute budgets under distribution shift.

Team contributions

Rui Chen: Research on project directions and proposing the current framework; offline dataset construction (scalar RM inference and data cleaning); cost stabilization; ablation experiments; report formulation; and literature review.

Landon Choy: Research on project directions and proposing the current framework; majority of formalized scripts; Lagrangian variant implementation; offline dataset construction (deep embeddings, generative model inference, data cleaning); report formulation; and literature review.

Cole Yarbrough: Research on project directions and proposing the current framework; offline dataset construction (generative model inference); setup for experiments (baselines, learned router, data cleaning); cost stabilization; report formulation; and literature review.

References

- [1] Zhenghao Xu, Qin Lu, Qingru Zhang, Liang Qiu, Ilgee Hong, Changlong Yu, Wenlin Yao, Yao Liu, Haoming Jiang, Lihong Li, Hyokun Yun, and Tuo Zhao. Ask a Strong LLM Judge when Your Reward Model is Uncertain. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [2] Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. RM-bench: Benchmarking reward models of language models with subtlety and style. In *The Thirteenth International Conference on Learning Representations*, 2025.
- [3] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems*, 2022.
- [4] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, Lester James V. Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating Reward Models for Language Modeling. *arXiv preprint arXiv:2403.13787*, 2024.
- [5] David Anugraha, Zilu Tang, Lester James V. Miranda, Hanyang Zhao, Mohammad Rifqi Farhansyah, Garry Kuwanto, Derry Wijaya, and Genta Indra Winata. R3: Robust Rubric-Agnostic Reward Models. *arXiv preprint arXiv:2505.13388*, 2025.
- [6] David Madras, Toni Pitassi, and Richard Zemel. Predict Responsibly: Improving Fairness and Accuracy by Learning to Defer. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- [7] Yonatan Geifman and Ran El-Yaniv. SelectiveNet: A Deep Neural Network with an Integrated Reject Option. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2151–2159, 2019.
- [8] Anqi Mao, Christopher Mohri, Mehryar Mohri, and Yutao Zhong. Two-Stage Learning to Defer with Multiple Experts. In *Advances in Neural Information Processing Systems*, 2023.
- [9] Shuqi Liu, Yuzhou Cao, Qiaozhen Zhang, Lei Feng, and Bo An. Mitigating Underfitting in Learning to Defer with Consistent Losses. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics*, pages 4816–4824, 2024.
- [10] Andrew Cotter, Heinrich Jiang, Maya Gupta, Serena Wang, Taman Narayan, Seungil You, and Karthik Sridharan. Optimization with Non-Differentiable Constraints with Applications to Fairness, Recall, Churn, and Other Goals. *Journal of Machine Learning Research*, 20(172):1–59, 2019.
- [11] Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, KaShun SHUM, and Tong Zhang. RAFT: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023.
- [12] Chris Yuhao Liu, Liang Zeng, Jiakai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-Reward: Bag of Tricks for Reward Modeling in LLMs, 2024.
- [13] Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Griffin Thomas Adams, Jeremy Howard, and Iacopo Poli. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2526–2547, Vienna, Austria, July 2025. Association for Computational Linguistics.

A Learn2Defer-RM experimental results

A.1 Performance across domains and difficulty levels

Domain	Model	Easy	Normal	Hard	Style Gap ($E - H$)
safety-response	RM-Mistral-7B (always cheap)	0.955	0.913	0.586	0.369
	Learn2Defer-RM (Ours at $\lambda = 0.1$)	0.978	0.945	0.690	0.288
	R3 (always heavy)	0.942	0.896	0.660	0.282
	random escalation (50%)	0.940	0.894	0.609	0.331
safety-refuse	RM-Mistral-7B (always cheap)	0.945	0.912	0.878	0.067
	Learn2Defer-RM (Ours at $\lambda = 0.1$)	0.991	0.966	0.952	0.039
	R3 (always heavy)	0.988	0.971	0.947	0.041
	random escalation (50%)	0.973	0.940	0.917	0.056
math	RM-Mistral-7B (always cheap)	0.826	0.561	0.296	0.530
	Learn2Defer-RM (Ours at $\lambda = 0.1$)	0.967	0.912	0.890	0.087
	R3 (always heavy)	0.946	0.916	0.887	0.060
	random escalation (50%)	0.886	0.728	0.596	0.290
code	RM-Mistral-7B (always cheap)	0.832	0.507	0.216	0.615
	Learn2Defer-RM (Ours at $\lambda = 0.1$)	0.925	0.798	0.735	0.190
	R3 (always heavy)	0.725	0.728	0.706	0.019
	random escalation (50%)	0.773	0.617	0.463	0.310
chat	RM-Mistral-7B (always cheap)	0.941	0.695	0.147	0.793
	Learn2Defer-RM (Ours at $\lambda = 0.1$)	0.974	0.817	0.478	0.496
	R3 (always heavy)	0.915	0.817	0.457	0.457
	random escalation (50%)	0.933	0.780	0.326	0.607

Table 2: Accuracy by domain and difficulty on RM Bench. The Easy, Normal, and Hard columns report pairwise preference accuracy on the corresponding difficulty splits.

Domain	Model	Easy	Normal	Hard	Style Gap ($E - H$)
safety-response	Learn2Defer-RM	0.038	0.081	0.364	-0.326
safety-refuse	Learn2Defer-RM	0.058	0.078	0.124	-0.066
math	Learn2Defer-RM	0.246	0.465	0.723	-0.477
code	Learn2Defer-RM	0.157	0.445	0.774	-0.617
chat	Learn2Defer-RM	0.044	0.235	0.862	-0.817

Table 3: Escalation rate of the Learn2Defer router across domains and difficulty levels.

A.2 Wide-and-deep ablations

Model	Overall Accuracy	Escalation Rate
Random escalation (baseline)	0.764070	0.500586
Deep-only router	0.767253	0.382161
Wide-only router	0.880067	0.297487
Wide+deep router	0.887772	0.359883

Table 4: Wide-deep ablation for the learned router. Wide-only features recover most of the gain, while deep-only features showed mild performance comparing to random escalation.

B Lagrangian approach

B.1 Lagrangian formulation for constrained optimization

The appeal of a Lagrangian formulation is the automatic adjusting of the strength of compute regularization to enforce a strict target escalation rate maximum rather than requiring a hand-tuned fixed coefficient. At a high level, the primal variable θ seeks to minimize the task loss, while the dual variable λ enforces the escalation constraint. The primal update therefore solves, for the current dual price, the best trade-off: $\theta \leftarrow \arg \min_{\theta} [\mathcal{L}_{\text{BCE}}(\theta) + \lambda \mathbb{E}[\pi_{\theta}(x)]]$.

The dual update then checks whether this solution violates the escalation budget. If it does, λ is increased so that future primal updates make escalation more expensive; otherwise, λ is decreased. At equilibrium, the router cannot further reduce the BCE loss without either worsening the loss directly or violating the escalation constraint, and the dual variable adjusts so that the budget is satisfied as tightly as beneficial. Under standard regularity conditions, this primal–dual procedure can be shown to converge to a saddle point of the Lagrangian corresponding to the optimal solution of the constrained problem. A formal proof of this convergence result is outside the scope of this paper.

In our implementation, the dual variable is updated using a stochastic gradient ascent step based on the observed escalation rate in each mini-batch. Let $\hat{e}$ denote the empirical batch escalation rate and let α denote the target escalation budget. The dual update performs ascent on the Lagrangian with respect to λ , yielding the update rule $\lambda \leftarrow \text{clip}_{[0, \lambda_{\max}]} (\lambda + \eta_{\lambda} (\hat{e} - \alpha))$, where η_{λ} is the dual learning rate and $\text{clip}(x, 0, \lambda_{\max})$ clamps the value to the interval $[0, \lambda_{\max}]$ to maintain numerical stability. Intuitively, if the observed escalation exceeds the target budget ($\hat{e} > \alpha$), the multiplier increases, raising the effective cost of escalation in subsequent updates. Conversely, if escalation falls below the target ($\hat{e} < \alpha$), the multiplier decreases. Because the multiplier is constrained to remain non-negative, repeated decreases eventually drive λ to zero when the escalation constraint is comfortably satisfied. At this point the compute regularization term disappears and the primal update reduces to minimizing the task loss alone. Thus, the dual mechanism activates when the escalation constraint is violated and becomes inactive once the router operates within the desired compute budget.

B.2 Main Learn2Defer-RM-lag experimental results

Split	Target Escalation	Escalation Rate	Accuracy	% Tokens Saved
Train	–	0.00	0.905	–
	0.05	0.05	0.800	–
	0.10	0.10	0.828	–
	0.15	0.15	0.851	–
	0.20	0.20	0.869	–
	0.25	0.25	0.866	–
Dev	–	1.00	0.763	–
	–	0.00	0.904	–
	0.05	0.05	0.787	–
	0.10	0.10	0.813	–
	0.15	0.15	0.840	–
	0.20	0.20	0.862	–
Test	0.25	0.25	0.873	–
	–	1.00	0.759	–
	–	0.00	0.723	100.00
	0.05	0.10	0.721	89.39
	0.10	0.18	0.785	78.19
	0.15	0.23	0.819	72.70
	0.20	0.35	0.877	55.53
	0.25	0.34	0.871	57.82
	–	1.00	0.866	0.00

Table 5: Escalation behavior and accuracy of Learn2Defer-RM-lag across the Skywork train/dev splits and the RM-Bench test set. Train and dev report escalation rates and accuracy during router training, while the test split additionally reports % Tokens Saved, defined as the percentage of generative tokens that would have been produced by the R3 verifier but were avoided due to routing decisions. An escalation rate of 0 corresponds to using only the scalar reward model (RM-Mistral-7B), while an escalation rate of 1 corresponds to always deferring to the generative verifier (R3).

B.3 Learn2Defer-RM-lag train curves

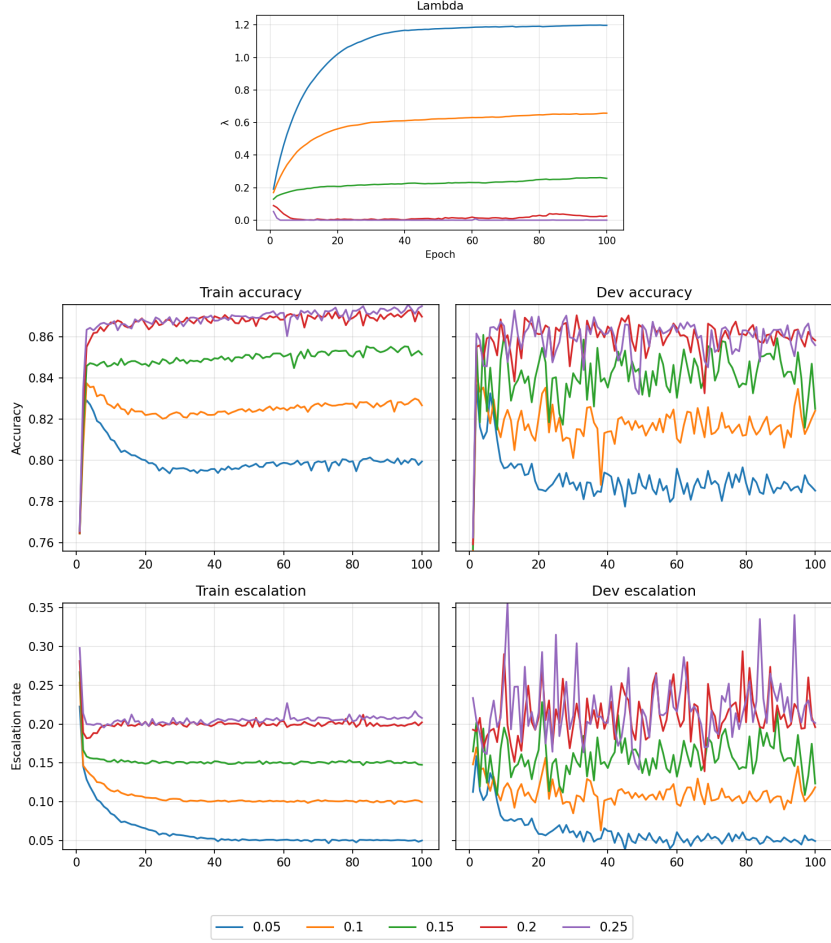


Figure 3: Training dynamics on the SKYWORKS dataset for different target escalation rates $\rho \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$. **Top:** Evolution of the Lagrange multiplier λ during training. The multiplier increases until the system satisfies the escalation constraint, with lower target escalation rates requiring larger λ values to more strongly penalize deferral to the heavy model. **Bottom:** Train and development accuracy (top row) and escalation rates (bottom row). Across settings, the router learns to match the desired escalation budget while maintaining stable accuracy. As expected, allowing higher escalation budgets yields higher accuracy, reflecting increased reliance on the stronger generative verifier. Development curves closely track training behavior, indicating stable optimization under the Lagrangian formulation.

B.4 Learn2Defer-RM-lag ablations

Domain	Escalation Rate	% Tokens Saved	Easy	Normal	Hard	Style Gap (E-H)
safety-response	0.00	100.00	0.955	0.913	0.586	0.369
	0.07	93.25	0.966	0.915	0.620	0.346
	0.07	93.50	0.964	0.919	0.611	0.352
	0.09	91.43	0.972	0.925	0.635	0.337
	0.13	87.67	0.972	0.934	0.656	0.316
	0.14	86.88	0.970	0.934	0.673	0.297
	1.00	0.00	0.942	0.896	0.660	0.282
safety-refuse	0.00	100.00	0.945	0.912	0.878	0.067
	0.03	95.54	0.962	0.924	0.914	0.048
	0.05	94.02	0.971	0.935	0.924	0.047
	0.06	93.34	0.980	0.948	0.925	0.055
	0.08	90.30	0.991	0.951	0.937	0.054
	0.09	89.12	0.991	0.960	0.948	0.042
	1.00	0.00	0.988	0.971	0.947	0.041
math	0.00	100.00	0.826	0.561	0.296	0.530
	0.09	90.27	0.839	0.599	0.463	0.376
	0.19	80.54	0.894	0.691	0.582	0.312
	0.28	70.64	0.895	0.807	0.682	0.213
	0.50	46.24	0.963	0.900	0.852	0.111
	0.45	53.21	0.946	0.873	0.812	0.134
	1.00	0.00	0.946	0.916	0.887	0.060
code	0.00	100.00	0.832	0.507	0.216	0.615
	0.13	87.92	0.836	0.545	0.437	0.399
	0.33	67.42	0.876	0.713	0.651	0.225
	0.34	67.72	0.879	0.718	0.640	0.238
	0.45	55.25	0.912	0.776	0.705	0.208
	0.48	51.51	0.925	0.807	0.724	0.202
	1.00	0.00	0.725	0.728	0.706	0.019
chat	0.00	100.00	0.941	0.695	0.147	0.793
	0.24	76.28	0.946	0.718	0.367	0.579
	0.28	71.29	0.956	0.757	0.401	0.556
	0.33	66.55	0.964	0.791	0.426	0.537
	0.42	56.56	0.969	0.819	0.470	0.499
	0.42	54.65	0.966	0.817	0.457	0.509
	1.00	0.00	0.915	0.817	0.457	0.457

Table 6: Accuracy by domain and difficulty of Learn2Defer-RM-lag on RM-Bench as escalation increases from the scalar-only endpoint to the generative-only endpoint. Bold values denote the best accuracy within each domain block for a given difficulty level.

Metric	Variant	0.05	0.10	0.15	0.20	0.25
Escalation Rate	Wide+Deep	0.095	0.177	0.226	0.351	0.339
	Deep-only	0.114	0.189	0.242	0.284	0.395
	Wide-only	0.066	0.172	0.432	0.479	0.475
RM-Bench Accuracy	Wide+Deep	0.721	0.785	0.819	0.877	0.871
	Deep-only	0.696	0.715	0.710	0.733	0.767
	Wide-only	0.698	0.780	0.889	0.895	0.895

Table 7: Ablation of routing architectures under the Lagrangian formulation with BCE loss. Columns correspond to target escalation budgets. The three router variants differ in their input features: *wide-only* uses only scalar reward model feature values, *deep-only* uses only MODERNBERT embeddings, and *wide+deep* combines both feature types. Values report RM-Bench test escalation rates and classification accuracy. Bold values indicate the lowest escalation rate per column in the escalation block and the highest accuracy per column in the accuracy block.

C Full experiment setup details

C.1 Data

Although the raw preference data are stored as tuples (q_i, a_i^+, a_i^-) , where a_i^+ is the chosen response and a_i^- is the rejected response, the router is trained on preference pairs with arbitrary displayed order. For each example, we optionally swap the response order at 50% probability and define

$$(a_{i1}, a_{i2}) = \begin{cases} (a_i^+, a_i^-), & \text{if } \text{swap}_i = 0, \\ (a_i^-, a_i^+), & \text{if } \text{swap}_i = 1. \end{cases}$$

We then construct the binary target

$$y_i = \mathbf{1}\{a_{i1} \text{ is the preferred response}\} = \mathbf{1}\{a_{i1} \succ a_{i2}\}$$

Thus, $y_i = 1$ when the chosen response is displayed first and $y_i = 0$ when the order is swapped. This order randomization is important for BCE-based training: without swapping, every example would satisfy $y_i = 1$, allowing the router to exploit a trivial positional shortcut instead of learning meaningful routing signals. Accordingly, the offline routing dataset stores `swap_i` together with order-aligned supervision fields, including the scalar-model margin, the softened heavy-model probability, and the final binary target.

Based on this displayed pair representation, we materialize an offline routing dataset and load it through a custom `RouterDatasetWide` class. Each example contains the scalar reward-model outputs for the chosen and rejected responses, token counts for each response, explicit features and frozen implicit representations required by router, and metadata fields used later for grouped evaluation (e.g., `source` for train/dev and `domain`, `difficulty`, and `style` labels for RM-Bench test data). We create separate dataloaders once at startup for train, dev, and test, using batch size 256 for train/test and 128 for dev.

We prepare both explicit and implicit features as inputs to the router. For explicit features, the implemented model uses four scalar inputs derived from the Stage-I reward model: the chosen reward score $r(a_{i1})$, the rejected score $r(a_{i2})$, the absolute score margin $|r(a_{i1}) - r(a_{i2})|$, and the absolute token-count difference between the two responses. For implicit features, we encode (q_i, a_{i1}) and (q_i, a_{i2}) with a lightweight ModernBERT encoder [13] and apply PCA to obtain 16-dimensional vectors per response, which are concatenated into a 32-dimensional embedding vector for each pair.

In addition to the router inputs, we also pre-compute order-aligned supervision fields from the offline heavy-model output. In particular, we derive a scalar-model probability, a softened heavy-model probability $p^{(H)}$, and the final binary target $y_i \in \{0, 1\}$ for the mixed prediction objective. All reward models and encoders remain frozen during training; only the router parameters are updated. This lets us train the router entirely offline without querying either reward model during optimization.

C.2 Router

The router is implemented as a small neural network that maps explicit and implicit example features to an escalation probability $\pi_\theta(x_i)$. Concretely, our router uses a two-head architecture. The wide branch processes four scalar features through a two-layer MLP, producing a 4-dimensional latent vector. The deep branch processes a 32-dimensional ModernBERT embedding through a separate two-layer MLP, producing a 5-dimensional latent vector. These two latent representations are concatenated and passed through a final linear layer with sigmoid activation to produce the router score.

To support feature ablations, we also implement three variants directly in code: `full`, which uses both branches; `wide_only`, which uses only the explicit-feature branch; and `deep_only`, which uses only the embedding branch. This allows us to isolate whether routing gains arise primarily from explicit scalar signals or from implicit semantic representations.

C.3 Training objective

Learn2Defer-RM Our implemented training objective differs from a fixed cost-weighted defer formulation. Instead, we train with binary cross-entropy on the final mixed prediction together with a cost penalty that gives additional control on escalation rate. Let $p_i^{(M)} = \sigma\left(\frac{r(a_{i1}) - r(a_{i2})}{T}\right)$ be the cheap-model preference probability obtained from the scalar margin after temperature scaling, and

let $p_i^{(D)}$ denote the soft heavy-model preference probability extracted from the generative judge. The final mixed system prediction is then

$$p_i^{(\text{sys})} = (1 - \pi_\theta(x_i)) p_i^{(M)} + \pi_\theta(x_i) p_i^{(D)}.$$

We optimize the router using the batch objective

$$\mathcal{L}_{\text{batch}} = \text{BCE}(p_i^{(\text{sys})}, y_i) + \lambda \cdot \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \pi_\theta(x_i),$$

where λ is a regularization parameter that controls the trade-off between prediction quality and escalation cost.

Lagrangian variant For the Lagrangian variant used in our budget-control experiments, λ is adaptively updated to match a target escalation rate ρ . In code, the dual variable is updated as

$$\lambda \leftarrow \text{clip} \left(\lambda + \eta_\lambda \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \pi_\theta(x_i) - \rho \right), 0, \lambda_{\max} \right),$$

so that λ increases when the router exceeds the target escalation rate and decreases otherwise. This gives more direct budget control than a fixed penalty coefficient.

We train for up to 100 epochs using AdamW with learning rate 10^{-3} . We evaluate on the dev set every 5 epochs and checkpoint the best model according to dev accuracy once the target escalation budget is met; before that point, we track the checkpoint whose dev escalation is closest to the target. In our current implementation, we threshold the router score at $\tau = 0.5$ at inference time to obtain a deterministic escalate/not-escalate decision.

C.4 Evaluation method

For each example, the routed system produces a binary preference decision over the displayed response pair. A prediction is counted as correct if this final routed decision matches the ground-truth preferred response. At inference time, we convert the router’s soft escalation score into a hard routing decision using a threshold of 0.5. If an example is not escalated, the final decision is inherited from the scalar reward model; if it is escalated, the final decision is inherited from the generative verifier. This yields a per-example routed correctness indicator that can be averaged over arbitrary subsets.

We compare our learned routers against three kinds of baselines: single-stage baselines (always use cheap models and always use heavy models), simple non-learned combinations of Stage I and Stage II (mean aggregation and learned linear combination), and random escalation policy at fixed probability of 50%. Our main method is the fixed- λ router. We evaluate the Lagrangian formulation as a budget-control extension. Our primary metrics are overall pairwise accuracy and escalation rate, where escalation rate serves as a proxy for compute usage. For RM-Bench, we additionally report grouped accuracies by domain and difficulty level (easy, normal, hard), as well as style-gap statistics when relevant.

The main metrics are accuracy and escalation rate, where escalation rate serves as a proxy for compute usage. During training, we also track train accuracy, train escalation, dev accuracy, dev escalation, loss, and the current dual variable λ , and save these to a CSV log for later analysis. For RM-Bench, we report the *style gap* (Easy–Hard) as a robustness metric, where a smaller gap indicates less degradation on harder examples. Our evaluation pipeline also saves per-example JSON records containing predicted escalation scores, correctness, strategy labels, and metadata fields, enabling downstream analysis by domain, difficulty, and style.

C.5 Experimental details

Unless otherwise noted, the router uses the full feature setting with both wide and deep heads enabled. The explicit-feature head receives 4 inputs, while the deep head receives a 32-dimensional concatenated embedding. The combination network maps the fused latent representation through hidden dimensions 32 and 16 before outputting a single routing logit. The router is trained with AdamW using a learning rate of 10^{-3} , batch size 256 for train/test and 128 for dev, and up to 100 epochs. For the scalar-model probability mapping, we use temperature $T = 1.0$. For the Lagrangian

variant, the dual variable is initialized at 0, updated with step size $\eta_\lambda = 0.005$, and clipped to the range $[0, \lambda_{\max}]$.

Model selection is performed on the dev split. When a target escalation budget is used, we retain checkpoints that satisfy the budget and select the one with the best dev accuracy; if no checkpoint exactly satisfies the budget, we retain the checkpoint whose dev escalation is closest to the target. For fixed- λ runs without an explicit target budget, we select checkpoints based on dev accuracy. At test time, the learned, cheap, heavy, and random policies are all evaluated on both dev and RM-Bench test splits using the same evaluation pipeline, and we save both aggregate metrics and detailed per-example outputs for later statistical and qualitative analysis.