IF YOU KNOW, YOU KNOW: SELF-SUPERVISED UNCERTAINTY ESTIMATION FOR LLM JUDGES OVER BINARY CHOICES

Anonymous authors
Paper under double-blind review

ABSTRACT

'LLM-as-a-judge' systems are increasingly relied upon for numerous tasks, such as labeling training data for AI models, and evaluating LLM generations in industry applications. However, a pervasive problem of these systems is that LLMs exhibit an alarming tendency to *reverse* their preference over binary choices, when the presentation order of the choices is inverted. This behavior is patently irrational, and calls into question whether it makes sense for the field to rely on LLM judges, absent a reliable *measure of uncertainty* over their verdicts. To that end, we propose a simple, scalable, and self-supervised method of training LLM judges to measure their own uncertainty in preferences over a binary choice. We show that our method is more accurate than any leading baselines, and generalizes surprisingly well across data domains. We also show that our method scales gracefully across model-sizes, and transfers across model families. Finally, we show that our method can enable downstream use-cases, such as adaptive inference compute for more accurate LLM-as-a-judge pipelines.

1 INTRODUCTION

Who would go to a doctor who gives conflicting diagnoses, depending on the order in which the evidence was presented? Unfortunately, across a host of model sizes and families, LLMs are observed to have remarkably unstable preferences, even over binary choices: presenting the same two choices in A versus B order gives a different outcome compared to presenting them in B versus A order. How can we rely on such an LLM?

Given the rapidly increasing prevalence of LLM-as-a-judge systems in various settings, and given the observed unreliability of the judgments produced by these systems, it's clear that uncertainty estimates over judgments would be quite useful. Prior work proposes various methods of calibrating the uncertainty of LLM judges (henceforth referred to as 'judges'). However, many of these methods require training an auxiliary model to estimate uncertainty from the judge's latent state. Deploying an additional model at inference time obviously adds cost and latency to judge pipelines. Additionally, many existing uncertainty calibration methods rely on ground-truth labels of 'correctness' of judgments for training. Generating these datasets is slow, and expensive.

In this paper, we address both of these problems. We finetune *the judge model itself* to express its own uncertainty over judgments, via a trained 'uncertainty' token. This removes the need to deploy an additional uncertainty model at inference time. Further, we do so using data collected via an entirely self-supervised procedure, simply by fine-tuning on judgments for which the judge's preference switched, when the presentation order of the choice was inverted.

We show that our method results in an uncertainty estimator that has superior predictive power over all evaluated baselines. Our method generalizes well across diverse data domains, scales across model sizes, and transfers across all tested model families. Our method is compute efficient, and we would argue, very simple to implement into any LLM-as-a-judge pipeline. Our method results in no material change in existing model capabilities for models of 7B parameters and above. Finally, we show how training models to output reliable uncertainty estimations can enable more efficient adaptive-inference LLM-as-a-judge systems.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2 Related Work

We organize existing uncertainty estimation methods along two dimensions: (1) whether the method requires training a model, and (2) how many full judge-model passes the method requires at inference.

2.1 Training-free, single-pass approaches

Methods which estimate uncertainty from the output distribution. The simplest training-free approach is to treat the log-probabilities the judge model assigns over output tokens as a 'confidence'. Other slightly-more-involved methods involve estimating uncertainty from summary statistics of the output token distribution (Kadavath et al., 2022; Plaut et al., 2025). However, the uncertainty estimates inferred from output distributions of LLMs are empirically observed to be quite poorly calibrated (Plaut et al., 2025).

Methods which prompt the model itself to output an uncertainty estimate. Another family of methods prompt models to output a numerical 'confidence' over their verdict, in text, in the same forward pass which generates the verdict (Tian et al., 2023; Xiong et al., 2024).

2.2 Trained, single-pass approaches

Auxiliary trained estimators. A separate line of work leaves the judge unchanged, and trains another (usually much smaller) model to predict whether the model's output is reliable. Most methods in this family reuse information produced during the judge's ordinary verdict pass, such as its hidden states (Liu et al., 2024b; Radharapu et al., 2026; Azaria & Mitchell, 2023; Orgad et al., 2025; Kossen et al., 2024), or only the input and output text (Ulmer et al., 2024). LitCab instead learns a lightweight head over the model's final hidden states that rescales its output logits so that its own answer probability is better calibrated (Liu et al., 2024d). A related approach, CCPS (Khanmohammadi et al., 2025), applies perturbations of increasing magnitude to the judge's final hidden state, and trains a classifier to predict correctness from how far and how quickly the answer distribution shifts.[1]

Integrated uncertainty generation. A different line of work adapts the judge model itself to communicate confidence. Models have been fine-tuned to express uncertainty verbally (Lin et al., 2022; Li et al., 2025), to output the probability that a given answer is correct (Kapoor et al., 2024), self-trained on sampling-based uncertainty estimates (Liu et al., 2024c), and taught to produce confidence-bearing rationales (Xu et al., 2024). Similar work trains the model itself to refuse or delegate on questions labeled as outside its knowledge (Zhang et al., 2024; Zhu et al., 2025a;b; Yang et al., 2026), although this can also lead to excessive refusal (Wen et al., 2025).

2.3 Training-free, multi-pass approaches

Post-hoc self-evaluation. This approach requires two forward passes of the judge. The first pass generates a verdict. The second pass feeds that verdict back into the judge and prompts it to assign an uncertainty score to its verdict (Kadavath et al., 2022).

Repeated-generation consistency. A related family of training-free multi-pass methods estimates uncertainty by generating multiple responses and measuring how much they agree (Manakul et al., 2023; Kuhn et al., 2023; Farquhar et al., 2024; Duan et al., 2024). These approaches avoid uncertainty-specific compute at training-time, but shift the compute burden to inference-time.

2.4 Trained, multi-pass approaches

Parameter-distribution methods. An interesting line of work uses variational inference to estimate a posterior distribution over the parameters of LLMs themselves, and subsequently leverages this posterior to estimate the predictive uncertainty. Given a training dataset $\mathcal{D}$, their posterior predictive distribution is obtained by marginalizing the parameter posterior:

$$P(\boldsymbol{y} \mid \boldsymbol{x}, \mathcal{D}) = \int P(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{D})\, d\boldsymbol{\theta}.$$

[1] We classify this method as single-pass, though one could argue either way, as although it does not require rerunning the complete judge model, it does require several evaluations of the judge's output layer.

This approach is neat in theory, but in practice comes with implementation hurdles. Firstly, the exact posterior is generally intractable, and secondly, modeling parameter posterior distributions over all of the parameters of an LLM requires substantial compute resources. A pragmatic solution to the above is to (a) use an approximate inference method for the true posterior distribution (Blundell et al., 2015), and (b) do the approximate inference procedure over a low-rank subset of the model's parameters (Wang et al., 2024b). Similar themed approaches train several independently initialized models or adapters and aggregate their predictions at inference time, at the cost of one forward pass per ensemble member (Lakshminarayanan et al., 2017; Balabanov & Linander, 2025).

Auxiliary learned estimators over multiple passes. Another branch of methods requires both multiple passes and a trained model. MACR trains a small estimator on the judge's preference probabilities under several different in-context demonstration sets, using human labels (Jin et al., 2026), and proxy-based uncertainty-aware reward models run multiple stochastic dropout passes of a separately trained reward model to obtain an uncertainty over its scores (Lee et al., 2024).

3 METHODOLOGY

The logic for our approach is based on three observations:

Many training approaches discard good training data. Prior work has documented that pairwise LLM judges can change their preference over a binary choice when the presentation order is reversed (Wang et al., 2023a). The literature generally treats this as a failure mode to be mitigated (Zheng et al., 2023; Badshah et al., 2026; Wang et al., 2023b; Zhu et al., 2023; Li et al., 2024). Rather than discard these inconsistent examples, we use them as supervision for training an uncertainty estimate.

There's something in the latents. We observe that linear probes over a judge's latent state can yield uncertainty estimators with impressive predictive power Radharapu et al. (2026); Liu et al. (2024b). Despite this, however, this internal uncertainty signal somehow fails to propagate to the final layer of the judge: explicitly prompting a judge to estimate its uncertainty yields a poor estimator, and similarly, estimating uncertainty from summary statistics of the judge's distribution over outputs, also yields a poor estimator (see results below). We hypothesize that judge's may actually *know* internally when they are uncertain, but lack the capacity to 'verbalize', or act on that signal.

Single call estimators are best. It is obviously much more efficient to have single call methods. These return both a judgment *and* an estimate of that judgment's uncertainty in the same forward pass, in contrast to multiple-call methods. In our binary choice setting, single call estimation is crucial because it does not admit the trivial solution of simply querying the judge twice, in both orders.

3.1 SELF-SUPERVISED DATA COLLECTION

Judgments which 'flip'

Let x be the beginning of a prompt which is passed to the model, a "judge" model parameterized by θ. Let $\{y_i, ..., y_j\}$ denote a set of candidate completions to x, which we shall ask the judge model to output its preference over. Concretely, we evaluate our method using data from the UltraFeedback dataset (Cui et al., 2023).

By way of an example from the UltraFeedback dataset, x could look like: *"I am going to Cairo in June of this year, thinking of 4-5 days, what are the best things I could do?"*.

The set of candidate completions $\{y_1, \ldots, y_n\}$ might contain some exciting suggestions of things to do involving pyramids and camels. In our method, we take each unordered pair $\{y_i, y_j\}$ with $i < j$, and construct two judging prompts, one for each presentation order.

The prompt P_π, passed to the judge model in both orders reads: *"You are an impartial judge. Compare the two responses to the instruction and decide which one is better. Answer with just a single letter."*[2].

The 'user message' which is passed on each call to the judge is templated as follows:

"Instruction: x Response A: y_1 Response B: y_2 Which response is better? Answer with just 'A' or 'B'."

[2]Note that in this data *collection* stage, the model is *forced* to choose between A or B, and is *not* given the option to declare it is uncertain.

These two prompts are passed to the judge model in turn, with the KV cache cleared between generation so as to treat each call independently. For each presentation order $\pi \in \{ij, ji\}$, we sum the next-token probabilities of the single-token spellings of each answer letter $\ell \in \{\mathrm{A}, \mathrm{B}\}$, with and without a leading space. Writing this sum as $m_{\pi,\ell}$, we normalize over A and B and choose the more probable letter:

$$p_{\pi,\ell} = \frac{m_{\pi,\ell}}{m_{\pi,\mathrm{A}} + m_{\pi,\mathrm{B}}}, \qquad \hat{\ell}_\pi = \arg\max_{\ell \in \{\mathrm{A},\mathrm{B}\}} p_{\pi,\ell}. \tag{1}$$

We map $\hat{\ell}_\pi$ back to the response it references. This leaves the content-level choice $w_\pi \in \{y_i, y_j\}$. We define the pair's *flip label* as: $\varphi = \mathbf{1}[\, w_{ij} \neq w_{ji} \,]$.

For a judge to choose 'A' in both orders means that the judge has chosen y_i in one presentation order and y_j in the inverted presentation order, which renders $\varphi = 1$. Even without saying anything about the *correctness* of either candidate completion, we can say that the judge's preference was *unstable*. We use this as a label of uncertainty over the judge's output: it's clear the judge's preference was not certain, given it was not invariant to permutation of the presentation order.

Dataset construction

The full UltraFeedback dataset contains 250,000 instructions. For compute efficiency reasons, we use a randomly sampled subset of the data, comprising 11,200 instructions, with four responses each. Four candidate responses yields $\binom{4}{2} = 6$ unordered pairs of responses. This yields $11{,}200 \times 6 \times 2 = 134{,}400$ judge calls, with the 2 coming from presenting the choice in both orders. We split this data subset by instruction prompt: 80% for training and 20% for held-out evaluation.

From the resultant 107,520 training examples, each of our training runs for smaller models (Qwen2.5-1.5B and 7B (Qwen Team, 2024), Llama-3.1-8B, and Mistral-7B) are trained using a seeded random subset of 32,000 ordered examples. For larger models (Qwen2.5-14B, Qwen2.5-32B, and Qwen2.5-VL-3B), we limit the training to 4,000 instructions.

For each judge model tested, we generate our 'flip' labels by first passing the judge all response-pairs in both presentation-orders. We label each ordered call with whether it 'flipped' or not. Next we will use these examples to train models to better identify when they are uncertain.

3.2 Training

3.2.1 LoRA Fine-tuning

We finetune the judge to express its uncertainty about its own output distribution, via the ordinary vocabulary token 'U' (uncertainty). We hypothesize that this way we can train the judge to learn a non-linear mapping between whatever signal appears to exist (in the related work) that linear probes pick up on, and the output token distribution. We do not fit any auxiliary model or separate uncertainty head. To make our implementation tractable, we keep the judge's source parameters θ fixed and learn a Low-Rank Adaptation (LoRA) adapter (Hu et al., 2021).

If a call resulted in a 'flip', we set its target t_π as U, so both presentations of a flipped pair receive target U. If the call did not result in a flip, the target remains the judge's original A/B verdict for that call[3]. During this finetuning stage, we augment both the system prompt and the user prompt with a third alternative. The judge can instead choose the letter 'U' denoting that it is 'uncertain'.

Formally, let $\widetilde{P}_\pi$ be the user prompt like the prompt used in collection P_π, but with 'U' added to its set of permitted answers. Our evaluations compare our trained model to an untrained baseline model that *also* had this 'U' as an explicit third option in the prompt[4]. Finetuning using LoRA keeps each pretrained weight matrix $\boldsymbol{W}_0$ fixed and learns a low-rank update $\boldsymbol{BA}$. For an input activation $\boldsymbol{h}$, the adapted layer computes

$$\boldsymbol{z} = (\boldsymbol{W}_0 + \boldsymbol{BA})\boldsymbol{h}, \qquad \boldsymbol{A} \in \mathbb{R}^{r \times d_{\text{in}}}, \quad \boldsymbol{B} \in \mathbb{R}^{d_{\text{out}} \times r}. \tag{2}$$

[3]This regularizes training and helps avoid collapse to a degenerate model which only outputs 'U'.

[4]During data *collection* we force the judge to choose between A or B, without giving it option U, but when evaluating our method, we of course must compare with a baseline judge which also *could* select option U, despite not being fine-tuned to be able to do so.

The update has rank at most r and requires $r(d_{\text{in}}+d_{\text{out}})$ trainable parameters per projection, compared with $d_{\text{in}}d_{\text{out}}$ for full fine-tuning. In our experiments, we set $r = 16$ on the query, key, value, and output projections in every self-attention block. We collect these trainable weight matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ into an *adapter*: η. Disabling the adapter recovers the source judge, giving our 'adapter-off control'. Appendix C gives the full training configuration.

Let $q_{\theta,\eta}(\cdot \mid \widetilde{P}_\pi)$ be the adapted model's next-token distribution for the three-way prompt $\widetilde{P}_\pi$, where θ are the frozen source-judge parameters and η contains the LoRA matrices. We minimize first-token cross-entropy:

$$\mathcal{L}(\eta) = -\frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\widetilde{P}_\pi, t_\pi)\in\mathcal{D}_{\text{train}}} \log q_{\theta,\eta}(t_\pi \mid \widetilde{P}_\pi). \tag{3}$$

3.2.2 Uncertainty Measure

After training, we can extract an estimate of our trained judge's uncertainty from one forward pass. Writing the summed next-token probabilities of each letter's spellings as m_{A}, m_{B}, and m_{U}, as in Equation 1, the uncertainty measure is defined as

$$P(\text{U}) = \frac{m_{\text{U}}}{m_{\text{A}} + m_{\text{B}} + m_{\text{U}}}. \tag{4}$$

Similar to the untrained setting, we obtain a verdict by choosing whichever of A or B is more probable. If we allow abstention, we abstain when U is more probable than either.

4 Results

4.1 Our trained $P(\text{U})$ measure is the strongest predictor

We compare the main uncertainty estimates on the same target: judgments from the original, adapter-off source judge which 'flip' on presentation order inversion. We measure estimator fidelity by Area Under ROC curve, or AUROC. An AUROC of 0.5 corresponds to chance, and 1.0 corresponds to perfect separation. Performance across all methods is shown below. Trained $P(\text{U})$ reaches 0.842 ± 0.002 AUROC (Figure 1), beating all evaluated competing methods. Color distinguishes trained from training-free estimators, while hatching identifies methods requiring more than one full-model call. Appendices A and B give full implementation details of every competing method.

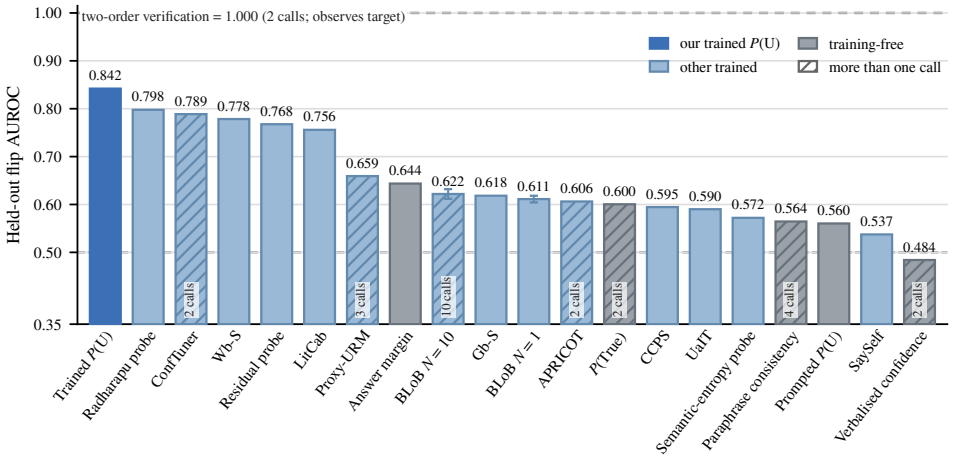

Figure 1: Predictive performance and its inference cost. AUROC for frozen source-judge flips on 26,880 held-out UltraFeedback calls. Trained $P(\text{U})$ is the mean over three seeds, its reported $\pm$ is sample standard deviation. Hatching marks additional full-model calls. BLoB error bars show sample standard deviation over three seeds. Implementation details: Appendices A and B.

Could the gain come simply from further training on judging data? To test this, we train an A/B-only control on the same calls, LoRA setup, optimizer, data budget, and schedule. The only difference

is that it never receives a U target. Its A/B margin, $|P(\mathrm{A}) - P(\mathrm{B})|$ with the sign reversed to serve as an uncertainty score (Appendix A), reaches 0.601 ± 0.012 flip AUROC, below the 0.644 of the untrained judge, whereas after our training the A/B margin reaches 0.761 ± 0.007 and $P(\mathrm{U})$ reaches 0.842 ± 0.002. The gain comes from the U supervision, not from extra training on the prompts.

4.2 Is $P(\mathrm{U})$ actually calibrated?

While AUROC can tell us how well an uncertainty estimate ranks judgments which 'flip', it does not tell us whether the value of the estimate is meaningful as a probability measure. We therefore also evaluate the raw probability assigned to U on the held-out test split: 26,880 judging calls from 2,240 UltraFeedback prompts.

To evaluate this, we group predictions into ten discrete probability intervals, and compare the predicted and observed flip rates within each group. Figure 2 compares trained and untrained models with $P(\mathrm{U})$ as an available option. Across three training seeds, the average gap, weighted by group size, is 1.8 percentage points (expected calibration error 0.018 ± 0.008). Overall, the model predicts a flip rate of 36.2%, close to the observed rate of 37.4%.

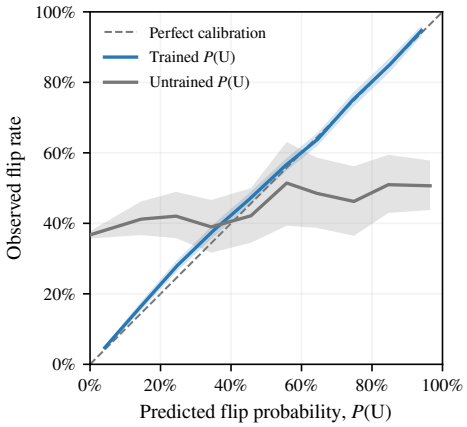

Figure 2: Training makes $P(\mathrm{U})$ better calibrated. Trained (three-seed mean) versus untrained $P(\mathrm{U})$ for Qwen2.5-7B-Instruct judge's flips on held-out test prompts excluded from training. Shading shows 95% prompt-bootstrap intervals. The dashed line shows how perfect calibration would appear.

We also measure the accuracy of individual probability measures. The Brier score averages the squared differences between each predicted probability and what happened (1 for a flip, 0 otherwise). It is 0.156 ± 0.001 across seeds. Across the three training seeds, this error is on average 33.4% lower than the degenerate solution of always predicting a 37.4% chance of flipping for every judgment (the global flip percentage). Untrained (but explicitly prompted) $P(\mathrm{U})$ scores 0.363 on the same calls. That is worse than constant prediction and the mean mass it assigns to U is just 3.7%. These results suggest that $P(\mathrm{U})$ does more than identify which judgments are more likely to flip. It actually yields a well calibrated *measure* of uncertainty, at least in distribution. Appendix E gives the per-seed reliability data.

4.3 Trained $P(\mathrm{U})$ generalizes across data domains

Despite training *only on UltraFeedback comparisons*, our trained $P(\mathrm{U})$ estimator reaches 0.800 AUROC on GSM8K mathematical solutions, 0.874 on TriviaQA factual answers, and 0.839 on HH-RLHF dialogue responses. These transfer scores also match or beat every competing estimator we evaluated across all three domains (Table 2). The corresponding untrained $P(\mathrm{U})$ scores are 0.633, 0.746, and 0.620 (top row of Figure 3). Appendix F audits every transfer number under the single-presentation protocol.

4.4 Uncertainty training scales across model sizes and families

Our method scales across all evaluated Qwen models, from 1.5B up to 32B. We observe that the uncertainty predictive power of a trained 1.5B model even exceeds that of an untrained 32B model, after just one hour of training. Applying the same method to Llama-3.1-8B, Qwen2.5-VL-3B (on text), and Mistral-7B yields flip AUROC of 0.85–0.93 (Figure 4). The 14B, 32B and VL-3B judges are trained on a separate 4,000-instruction set with its own held-out split (Appendix C), so their AUROCs are measured on a different evaluation set from the 1.5B and 7B judges. Our method is also effective when we extend the modality to images, Qwen2.5-VL-3B $P(\mathrm{U})$ AUROC reaches 0.946 on GQA and 0.973 on VQAv2, versus 0.847 and 0.936 for the untrained judge model.

The predictive power of $P(\mathrm{U})$ continues to improve as we add more training examples. On Qwen2.5-7B, held-out flip AUROC rises from 0.773 with 2k ordered training rows to 0.843 with 32k (seed 0)

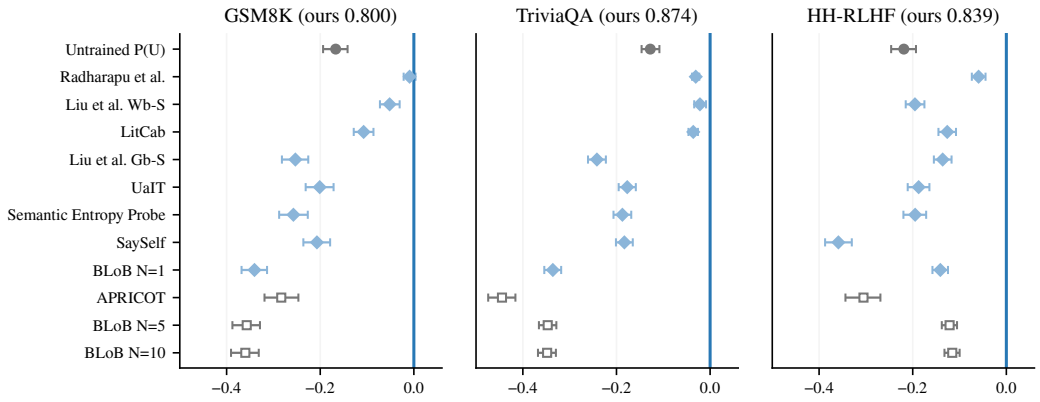

Figure 3: Zero-shot cross-domain transfer. AUROC of each evaluated method minus the AUROC of our method. AUROC differences below zero favor our method. For example, the generalization performance of our method outperforms that of Radharapu et al. (2026) by a small amount, and outperforms that of SaySelf (Xu et al. (2024)) by quite a large amount. Bars are paired 95% confidence intervals across three adapter seeds. The filled gray circle is untrained $P(\mathrm{U})$ from the same judge with the adapter off. Gray, open markers identify methods requiring extra model evaluations.

and 0.856 on the complete 107,520-row training split, though returns are seen to diminish (Appendix Figure 9). While the 32k setting used throughout our main experiments was chosen for compute reasons, future work can look deeper into the scaling limits of this method, potentially in pipelines for post-training LLMs to be more metacognitive.

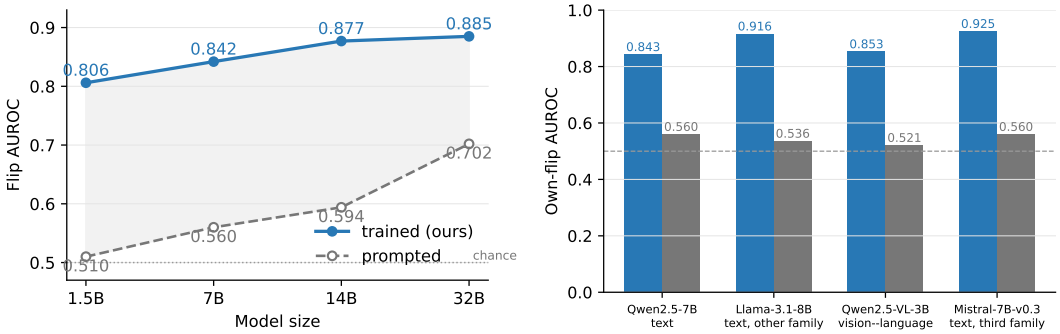

Figure 4: Left: the Qwen2.5 size sweep. Size helps, but our method with trained P(U) helps more. Right: family and modality replications. Blue is trained $P(\mathrm{U})$, gray is adapter-off $P(\mathrm{U})$. Each model is trained on its own source judge's flips.

4.5 Uncertainty awareness does not erode existing capabilities

The uncertainty estimation capability gain does not come at the expense of sacrificing the model's general capabilities. Training with our method increases Qwen2.5-7B's judging accuracy from 79.70% to $83.00 \pm 0.47\%$ on RewardBench. Llama-3.1-8B sees a similar increase from 70.52% to $71.89 \pm 0.33\%$ (mean and standard deviation across three training seeds). The untrained judge is scored under its original A/B prompt and the trained judge under the A/B/U prompt, taking the more probable of A and B. Table 12 shows that the untrained judge scores the same under either prompt. Every seed improves over the untrained judge (Figure 5). See Appendix Table 9 for all seed scores.

We also test Qwen2.5-7B on seven standard benchmarks, using their normal answer formats (no U option). Figure 6 reports each benchmark separately. MMLU, MMLU-Pro, GSM8K, BBH, TriviaQA and SQuAD2 change by less than one percentage point. GPQA-diamond rises from 24.24% to 30.81%. These evaluations are single-seed due to compute constraints. Appendix Table 10 gives the question counts and paired intervals for each benchmark.

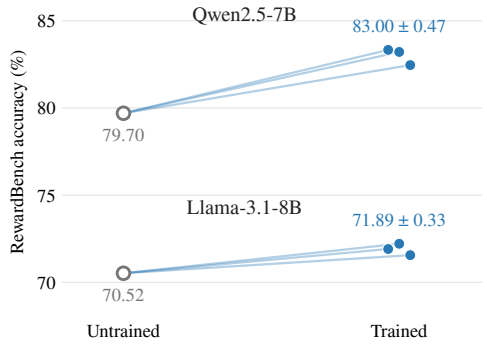

Figure 5: RewardBench accuracy improvement. Blue dots show all three adapter seeds and labels give mean ± SD. Accuracy measured across 5,970 choices. Baseline uses A/B, trained judges use A/B/U and return the higher-probability A/B verdict.

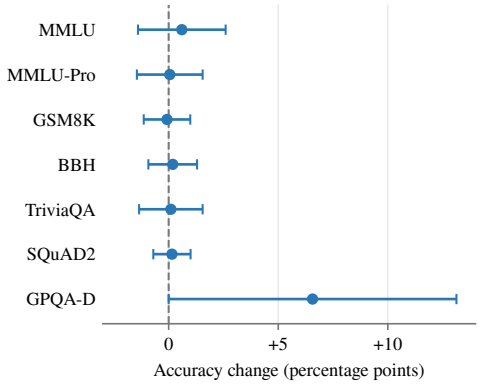

Figure 6: Mostly negligible changes in ordinary benchmark accuracy. Trained minus untrained accuracy for Qwen2.5-7B, seed 0. Bars are 95% paired question-bootstrap intervals. Both conditions use identical questions, greedy decoding, and no U option.

Appendix Figure 10 also includes the individual benchmark results for Qwen 1.5B, 14B and 32B and Llama-3.1-8B. No atrophy is seen across the board, except at 1.5B - at the 1.5B scale the effects are mixed. We imagine this may be because at that scale, the same rank-16 update changes a larger fraction of the model's parameters.

4.6 Trained $P(\mathrm{U})$ tracks judge-specific uncertainty

It's important to understand whether $P(\mathrm{U})$ identifies *judge-specific uncertainty*, or simply identifies questions which are *difficult to answer*, for many judges. We evaluate this by using a $P(\mathrm{U})$ trained on Qwen2.5-7B-Instruct to predict flips by adapter-off Qwen2.5-7B-Instruct, Llama, and Mistral judges (Table 14). Across three adapter seeds, using a pair-level score (Appendix I), Qwen's score predicts Qwen's flips with 0.862 AUROC, compared with 0.683 for Llama and 0.716 for Mistral. The lag for Llama and Mistral indicates a judge-specific uncertainty model, rather than gauging generic question-difficulty.

4.7 Training makes judges more consistent, and more appropriately confident

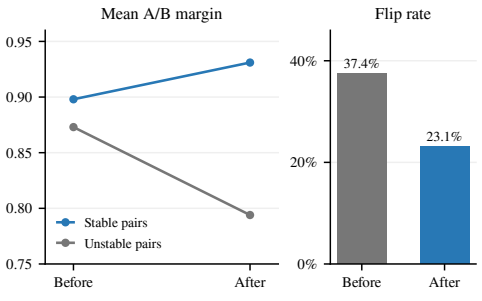

Figure 7: Training changes what the judge is confident about. A/B-only prompting: U is unavailable.

Even when we remove U from the prompt, and force an A/B choice, training redistributes confidence in the desired direction. Evaluated on Qwen2.5-7B-Instruct, the mean A/B margin rises from 0.898 to 0.931 on stable pairs and falls from 0.873 to 0.794 on unstable pairs (Figure 7). The trained Qwen2.5-7B-Instruct model's flip rate across the dataset also fell *overall* from 37.4% to 23.1%, while Llama 3.1-8B-Instruct's flip-rate also falls from 54.2% to a mean of 32.8% across three seeds. Training therefore yields more than a reliable indicator: it makes the judge more confident on stable decisions. The score also survives that change, predicting the trained judge's own remaining flips at 0.816 ± 0.010 AUROC (Appendix J.2). Appendix J.1 gives further details.

4.8 Online routing approaches 32B judging accuracy with fewer FLOPs

We test whether a small judge can use its learned $P(\mathrm{U})$ to decide whether a judgment should be routed to a more capable model. For a set of questions, Qwen2.5-7B first returns its A/B/U probabilities. If an 'escalation rule' fires, the choice is routed to a Qwen2.5-32B expert, otherwise we keep the 7B model's higher-probability A/B choice. We evaluate three 'escalation rules': escalate when $P(\mathrm{U}) > 0.5$, escalate when U wins the three-way softmax, $P(\mathrm{U}) > \max\{P(\mathrm{A}), P(\mathrm{B})\}$, or escalate when $P(\mathrm{U}) > \tau$, with τ calibrated on 100 unlabeled comparisons[5]. All three rules are evaluated on the same remaining 5,770 RewardBench comparisons over 2,885 questions. Results average three 7B adapter seeds.

Figure 8 shows the accuracy–compute tradeoff as we vary the calibrated $P(\mathrm{U})$ threshold, keeping both models' verdicts fixed. The standalone 7B and 32B judges provide the always-off and always-on expert references.

The fixed $P(\mathrm{U}) > 0.5$ rule sends 16.61% of calls to 32B and reaches 85.90% accuracy while saving 58.02% of estimated inference FLOPs. Escalating whenever U wins sends 18.31% of calls to 32B, reaches 86.18% accuracy, and saves 56.27%. Neither rule needs calibration. With 100 calibration calls, accuracy rises to 87.58% at 30.75% expert use and 43.21% FLOP savings, including calibration. For comparison, 7B alone reaches 82.99% and 32B alone reaches 88.87% on the same evaluation set. Thus the calibrated cascade is within 1.29 percentage points of always using 32B while reducing estimated inference compute by more than two-fifths (Figure 8)[6]. See Appendix K for further details.

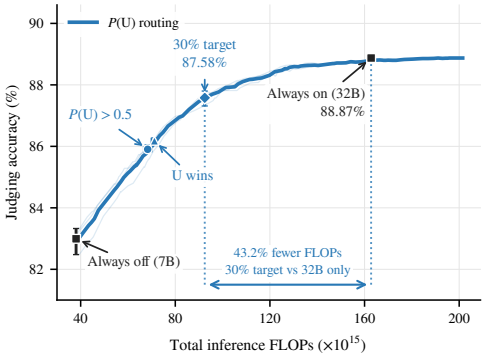

Figure 8: Our routing method versus an always-on or always-off expert. Total estimated FLOPs include all judge calls, and calibration where used. Circle: $P(\mathrm{U}) > 0.5$; triangle: U wins; diamond: calibrated 30% target. Dotted guides connect the 30% routing and 32B-only points to the 43.2% compute-saving. Bold curve averages three seeds, faint curves and marker ranges show seed variation.

5 Conclusion

We present a simple, scalable, and self-supervised method of training an uncertainty estimator over binary choices. Our method learns a well-calibrated uncertainty score, which supersedes existing methods, and transfers surprisingly well across various data domains. Our method requires no training of an additional model, and has been shown to significantly reduce the footprint of LLM-as-a-judge pipelines. Our method scales gracefully across model sizes, and provides benefits across all tested model families.

Our results also point to two directions for future work, which we explore in preliminary form in the appendix. First, trained $P(\mathrm{U})$ carries information beyond the questions on which it is measured: measuring it on a small sample of questions and averaging over each remaining question's nearest neighbors in the judge's activation space predicts where the judge errs better than the same procedure with untrained $P(\mathrm{U})$ (Appendix L). This suggests $P(\mathrm{U})$ can be used to map gaps in a model's capabilities, not just to score individual judgments. Second, $P(\mathrm{U})$ can filter self-labeled preference data for recursive fine-tuning: in a preliminary experiment, doing so improves DPO outcomes over unfiltered self-labeling under all six evaluators we tested (Appendix M). We outline our initial results in the appendix, and hope they inspire future work in these research directions.

Reproducibility statement

Every prompt is specified verbatim: the judging prompt and two-way question (Section 3.1), the three-way U prompt (Appendix A), and all uncertainty baselines (Appendix A). U is the ordinary

[5]For this rule, we use one presentation from each of 100 calibration questions and fix τ at the 70th percentile of their scores, targeting approximately 30% expert use

[6]FLOPs are estimated from each call's recorded input length, rather than the fraction of expert calls alone.

vocabulary token "U". Scores are first-token probabilities re-normalized over the allowed letters, summing with and without-space variants. The training set is a seeded uniform subsample of 32,000 of the 107,520 training calls after an 80/20 split by instruction. The held-out set is all 26,880 calls from held-out instructions. The learned-comparator sweep uses 28,850 training and 3,150 validation calls, matching the same 32,000-training-row budget, and scores the identical held-out calls. Each primary comparator sees one order presentation (the reverse presentation order is used only to construct the 'flip' label). Of the 2,985 RewardBench prompts, 5 (0.17%, all adversarial subsets) share an instruction with UltraFeedback. The responses are RewardBench's own, and removing those pairs changes no reported error rate by more than 0.002. For §4.8, 100 single-presentation calibration calls fix the threshold before evaluating 5,770 disjoint RewardBench calls. Inference FLOPs use the explicit parameter-token estimate in Appendix K.

AI USE STATEMENT

Generative AI tools were used to write and debug experiment and analysis code, and audit frozen results. All AI-assisted code was reviewed and its outputs checked against the research log by the authors. The authors take responsibility for the final content, claims, and artifacts.

REFERENCES

Amos Azaria and Tom Mitchell. The internal state of an LLM knows when it's lying. *Findings of EMNLP*, 2023.

Sher Badshah, Ali Emami, and Hassan Sajjad. SCOPE: Selective conformal optimized pairwise LLM judging. *arXiv preprint arXiv:2602.13110*, 2026.

Oleksandr Balabanov and Hampus Linander. Uncertainty quantification in fine-tuned LLMs using LoRA ensembles. *arXiv preprint arXiv:2402.12264*, 2025.

Felix J. Binder, James Chua, Tomek Korbak, Henry Sleight, John Hughes, Robert Long, Ethan Perez, Miles Turpin, and Owain Evans. Looking inward: Language models can learn about themselves by introspection. *arXiv preprint arXiv:2410.13787*, 2024.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pp. 1613–1622, 2015.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. UltraFeedback: Boosting language models with high-quality feedback. *arXiv preprint arXiv:2310.01377*, 2023.

Jinhao Duan, Hao Cheng, Shiqi Wang, Alex Zavalny, Chenan Wang, Renjing Xu, Bhavya Kailkhura, and Kaidi Xu. Shifting attention to relevance: Towards the predictive uncertainty quantification of free-form large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5050–5063, 2024. doi: 10.18653/v1/2024.acl-long.276.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630:625–630, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Gaojie Jin, Yong Tao, Lijia Yu, and Tianjin Huang. Margin-adaptive confidence ranking for reliable LLM judgement. *arXiv preprint arXiv:2605.15416*, 2026. doi: 10.48550/arXiv.2605.15416. Accepted to the International Conference on Machine Learning.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. URL https://arxiv.org/abs/2001.08361.

Sanyam Kapoor, Nate Gruver, Manley Roberts, Katherine Collins, Arka Pal, Umang Bhatt, Adrian Weller, Samuel Dooley, Micah Goldblum, and Andrew Gordon Wilson. Large language models must be taught to know what they don't know. *Advances in Neural Information Processing Systems*, 2024.

Reza Khanmohammadi, Erfan Miahi, Mehrsa Mardikoraem, Simerjot Kaur, Ivan Brugere, Charese Smiley, Kundan S. Thind, and Mohammad M. Ghassemi. Calibrating LLM confidence by probing perturbed representation stability. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10448–10514, 2025. doi: 10.18653/v1/2025.emnlp-main.530.

Jannik Kossen, Jiatong Han, Muhammed Razzak, Lisa Schut, Shreshth Malik, and Yarin Gal. Semantic entropy probes: Robust and cheap hallucination detection in LLMs. *arXiv preprint arXiv:2406.15927*, 2024.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *International Conference on Learning Representations*, 2023.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Joonho Lee, Jae Oh Woo, Juree Seok, Parisa Hassanzadeh, Wooseok Jang, Juyoun Son, Sima Didari, Baruch Gutow, Heng Hao, Hankyu Moon, Wenjun Hu, Yeong-Dae Kwon, Taehee Lee, and Seungjai Min. Improving instruction following in language models through proxy-based uncertainty estimation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 27009–27036, 2024.

Haitao Li, Junjie Chen, Qingyao Ai, Zhumin Chu, Yujia Zhou, Qian Dong, and Yiqun Liu. CalibraEval: Calibrating prediction distribution to mitigate selection bias in LLMs-as-judges. *arXiv preprint arXiv:2410.15393*, 2024.

Yibo Li, Miao Xiong, Jiaying Wu, and Bryan Hooi. ConfTuner: Training large language models to express their confidence verbally. *Advances in Neural Information Processing Systems*, 2025.

Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. *Transactions on Machine Learning Research*, 2022.

Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-reward: Bag of tricks for reward modeling in LLMs. *arXiv preprint arXiv:2410.18451*, 2024a.

Linyu Liu, Yu Pan, Xiaocheng Li, and Guanting Chen. Uncertainty estimation and quantification for LLMs: A simple supervised approach. *arXiv preprint arXiv:2404.15993*, 2024b.

Shudong Liu, Zhaocong Li, Xuebo Liu, Runzhe Zhan, Derek F. Wong, Lidia S. Chao, and Min Zhang. Can LLMs learn uncertainty on their own? expressing uncertainty effectively in a self-training manner. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 21635–21645, 2024c. doi: 10.18653/v1/2024.emnlp-main.1205.

Xin Liu, Muhammad Khalifa, and Lu Wang. LitCab: Lightweight language model calibration over short- and long-form responses. In *International Conference on Learning Representations*, 2024d.

Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Hadas Orgad, Michael Toker, Zorik Gekhman, Roi Reichart, Idan Szpektor, Hadas Kotek, and Yonatan Belinkov. LLMs know more than they show: On the intrinsic representation of LLM hallucinations. *International Conference on Learning Representations*, 2025.

Benjamin Plaut, Khanh Xuan Nguyen, and Tu Trinh. Probabilities of chat LLMs are miscalibrated but still predict correctness on multiple-choice QA. *Transactions on Machine Learning Research*, 2025.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Bhaktipriya Radharapu, Eshika Saxena, Kenneth Li, Chenxi Whitehouse, Adina Williams, and Nicola Cancedda. Calibrating LLM judges: Linear probes for fast and reliable uncertainty estimation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 193–216, San Diego, California, USA, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-industry.14. URL https://aclanthology.org/2026.acl-industry.14/.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D. Manning. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. *Proceedings of EMNLP*, 2023.

Dennis Ulmer, Martin Gubri, Hwaran Lee, Sangdoo Yun, and Seong Oh. Calibrating large language models using their generations only. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15440–15459, 2024. doi: 10.18653/v1/2024.acl-long.824.

Haoxiang Wang, Wei Xiong, Tengyang Xie, Han Zhao, and Tong Zhang. Interpretable preferences via multi-objective reward modeling and mixture-of-experts. *Findings of EMNLP*, 2024a.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*, 2023a.

Yibin Wang, Haizhou Shi, Ligong Han, Dimitris Metaxas, and Hao Wang. BLoB: Bayesian low-rank adaptation by backpropagation for large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024b.

Yidong Wang, Zhuohao Yu, Zhengran Zeng, Linyi Yang, Cunxiang Wang, Hao Chen, Chaoya Jiang, Rui Xie, Jindong Wang, Xing Xie, Wei Ye, Shikun Zhang, and Yue Zhang. PandaLM: An automatic evaluation benchmark for LLM instruction tuning optimization. *arXiv preprint arXiv:2306.05087*, 2023b.

Bingbing Wen, Jihan Yao, Shangbin Feng, Chenjun Xu, Yulia Tsvetkov, Bill Howe, and Lucy Lu Wang. Know your limits: A survey of abstention in large language models. *Transactions of the Association for Computational Linguistics*, 2025.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs. In *International Conference on Learning Representations*, 2024.

Tianyang Xu, Shujin Wu, Shizhe Diao, Xiaoze Liu, Xingyao Wang, Yangyi Chen, and Jing Gao. SaySelf: Teaching LLMs to express confidence with self-reflective rationales. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5985–5998, 2024. doi: 10.18653/v1/2024.emnlp-main.343.

Haoyan Yang, Reza Shirkavand, Yukai Jin, Jiawei Zhou, Shangqian Gao, and Heng Huang. Capability self-assessment: Teaching LLMs to know their limits. *arXiv preprint arXiv:2606.00251*, 2026.

Hanning Zhang, Shizhe Diao, Yong Lin, Yi R. Fung, Qing Lian, Xingyao Wang, Yangyi Chen, Heng Ji, and Tong Zhang. R-tuning: Instructing large language models to say 'I don't know'. In *Proceedings of NAACL*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 2023.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. JudgeLM: Fine-tuned large language models are scalable judges. *arXiv preprint arXiv:2310.17631*, 2023.

Runchuan Zhu, Zinco Jiang, Jiang Wu, Zhipeng Ma, Jiahe Song, Fengshuo Bai, Dahua Lin, Lijun Wu, and Conghui He. GRAIT: Gradient-driven refusal-aware instruction tuning for effective hallucination mitigation. *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025a.

Runchuan Zhu, Zhipeng Ma, Jiang Wu, Junyuan Gao, Jiaqi Wang, Dahua Lin, and Conghui He. Utilize the flow before stepping into the same river twice: Certainty represented knowledge flow for refusal-aware instruction tuning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025b.

A Baseline Uncertainty Estimator Implementations

Every signal in Figure 1 and Table 1 is scored by how well it predicts, on the same 26,880 held-out calls, that a judgment will flip when the two responses are presented in the reverse order.

A.1 Method Details

Answer-token margin. $-|p_A - p_B|$ after re-normalizing over A and B.

P(True) self-evaluation (Kadavath et al., 2022). The model is shown the instruction, both responses, and its own verdict, and then asked a binary question:

> Proposed judgment: Response A is better. Is this judgment correct? Answer with just 'True' or 'False'.

We do not use the generated text. The score is the model's implicit probability, read from the logits as $p(\text{True})/(p(\text{True}) + p(\text{False}))$, negated.

Verbalized confidence (Tian et al., 2023). The model is shown the same information and asked to report a number instead:

> You judged that Response A is better. How confident are you in that judgment? Answer with a single integer from 0 (a pure guess) to 100 (certain), and nothing else.

Here the score *is* the generated text: we parse the integer and negate it. The reports average 87 out of 100, and a third of all judgments are rated at least 90. Those highly-rated judgments reverse more often than the model's average, which is why this signal scores below chance.

Paraphrase consistency ($k = 4$). We present the same ordered comparison under the original prompt and three stylistic variants:

> Which of the two responses is better? Reply with only A or B.
>
> Compare Response A and Response B. Which one is superior? Answer with a single letter, A or B.
>
> Judge which response better addresses the instruction. Output exactly one letter: A or B.

The score is the negated majority share across four votes. It costs four passes and gains 0.064 over chance, versus 0.342 for single-pass adapted-model $P(\text{U})$. The variants do not re-order the options, so this baseline tests stylistic rather than order invariance.

Chain-of-thought self-consistency ($k = 5$). On 3,000 seeded held-out pairs, the adapter-off model receives, in both orders:

> You are an impartial judge. Compare the two responses to the instruction and decide which one is better. Think step by step, then end with exactly FINAL ANSWER: A or FINAL ANSWER: B.

Generation is capped at 320 tokens. One greedy chain per order gives the reasoned verdict. Because 22% of these chains reach the cap without a verdict, 2,026 pairs have a verdict in both orders. Five further chains per order, sampled at $T = 0.7$, give the agreement score: in each order, the fraction of chains that pick the most common verdict. A pair's score is the lower of its two orders' agreements. Results are in Table 15.

Table 1: Predicting held-out adapter-off flips (26,880 calls, flip rate 0.374). Each signal is defined in Appendix A. The adapted-model row is mean ± s.d. over three seeds. Every adapted-model-versus-training-free difference is positive in all 2,000 bootstrap resamples. The supervised probe uses the same flip labels on a common held-out split.

	Method	AUC
adapted	$P(\mathrm{U})$ **with** adapter	$\mathbf{0.842 \pm 0.002}$
supervised probe	linear residual probe (adapter-off model)	0.768
adapter off	$P(\mathrm{U})$ **without** adapter (identical prompt)	0.560
	answer-token margin	0.644
	P(True) self-evaluation	0.600
	paraphrase consistency (k=4)	0.564
	verbalized confidence 0–100	0.484

B Trained Uncertainty Estimator Implementations

Every baseline sees one presentation order from the UltraFeedback dataset. The dataset is split by instruction, and comprises 28,850 training and 3,150 validation calls.

Many of the cited methods estimate answer correctness, or human agreement, and not order invariance. Repeating their original experiments exactly is therefore not directly applicable to our question. As such, we amend the baseline implementations in the most minimal way possible, such to render them comparable to our method. The below outlines the changes made in each instance.

By and large, the main change we needed to make in each instance was to re-frame the supervision target from 'answer-correctness' to 'verdict flip under presentation order inversion'. In general, no changes were made to architecture, loss, optimizer, schedule, or method-specific feature construction, for any model. We believe our adaptations are source-faithful endpoint adaptations, but would welcome any feedback on how we could improve our adaptations.

B.1 Method Details

Radharapu et al. Brier-trained linear judge-state probe. Radharapu et al. fit one affine readout of a reasoning judge's last-token residual state to verdict correctness (Radharapu et al., 2026). The scalar is mapped to $[0, 1]$ and trained with MSE/Brier loss. Concretely, for residual state h and binary target y, the probe minimizes $\mathcal{L}_{\mathrm{Brier}} = (\sigma(w^\top h + b) - y)^2$ while the judge itself remains frozen. Their target is verdict correctness. Their recipe uses 4,000 examples, learning rate 10^{-4}, weight decay .01, batch four, and ten epochs. The main method says the affine output is passed through a sigmoid, whereas Appendix A.5 says the logit is clamped to $[0, 1]$. We follow the main description.

We make the unavoidable endpoint substitution $y = \mathbf{1}[\text{verdict correct}] \to \mathbf{1}[\text{order inversion}]$ and read the judge's ordinary decision state. We do not add the paper's reasoning trace or verbal-confidence preface because that would change the judge whose instability defines our endpoint.

LitCab. LitCab learns a vocabulary-wide linear calibration layer on top of a frozen language model and adds its output to the base next-token logits (Liu et al., 2024d). The layer is initialized at zero and trained with the released max-margin objective, full-vocabulary normalizer, learning rate 10^{-5}, batch 128, and validation early stopping. Native LitCab promotes a correct answer token over an incorrect one. We make the endpoint substitution that, on stable calls, the recorded verdict is positive and U is negative, whereas on reversing calls U is positive and the recorded verdict is negative. The base judge remains frozen, and the calibration layer reuses its final state, so deployment still requires one full-model context. No reversed test presentation is shown.

Semantic Entropy Probe. Kossen et al. (2024) replace sampling and NLI clustering with a logistic probe that reads one hidden state and predicts whether a generation's semantic entropy is high or low. Two steps of that recipe are unnecessary in our adaptation, which generates a single A/B/U token. First, the three candidate outputs are already distinct meanings, so NLI clustering would only recover the classes we start from. Second, one forward pass yields the whole A/B/U distribution, so

the ten categorical draws we take from it actually share a single prefix computation, so we count those 10 draws as just a single model call. We compute the entropy of that distribution in closed form, normalized to $[0, 1]$ by $\log 3$. We then binarize the entropy with the released method's two-level reconstruction criterion: the split minimizing within-group squared error, which is fit on the training and validation entropies. The fit converges at iteration 2,067 of a 5,000-iteration cap. Both departures make our adaptation cheaper than the original implementation (which is reflected in our analysis).

UaIT. UaIT self-trains a model to emit its answer and verbal confidence together (Liu et al., 2024c). Its teacher computes uncertainty, maps it through $\exp(-u)$, and retains high-confidence correct or low-confidence incorrect examples before supervised LoRA training. We replace correctness by judge stability and replace SAR with exact categorical A/B/U entropy, since the declared verdict classes are one-token semantic outcomes. We retain the released 0.5 filter, percentage output format, rank-64 LoRA configuration, four-epoch schedule, optimizer, and effective batch, while using the common Qwen judge rather than the released Mistral checkpoint.

SaySelf. SaySelf teaches a model to emit an answer, a self-reflective rationale, and an integer confidence, then optimizes confidence with a correctness-dependent quadratic reward (Xu et al., 2024). In the released pipeline, repeated long-form answers and an external summarizer construct the reflection and confidence teacher. For our binary endpoint, exact A/B predictive entropy replaces 100 long-form samples and the summarizer: the teacher confidence is $\mathrm{round}(10e^{-H})$ clipped to $[1, 10]$, and the two declared reflections state whether sampled verdicts are largely consistent or in disagreement. The SFT target contains the judge's original verdict, reflection, and confidence.

We retain rank-64, scale-16 LoRA with dropout 0.1, restricted to the query/key/value/output projections rather than the release's attention-plus-MLP set, three SFT epochs at 7×10^{-5}, effective batch 128, 3% linear warm-up, and the released 3,000-example, two-epoch confidence-only PPO stage at 10^{-5}. The unavoidable endpoint substitution replaces correctness by stability in the native reward, $r = 1 - 2(s - c)^2$, where $s = 1 - \mathrm{flip}$ and $c \in \{.1, \ldots, 1\}$. A common Qwen judge replaces the released Mistral checkpoint. Qwen tokenizes confidence '10' as two tokens, so the corrected implementation scores its exact autoregressive log probability rather than treating 10 as one token. The other nine actions are atomic (singular tokens). All 26,880 test outputs parse, giving 0.537 AUROC in-domain and 0.593/0.691/0.480 on GSM8K/TriviaQA/HH-RLHF with one adapted-judge evaluation.

ConfTuner. ConfTuner represents confidence as a token distribution (Li et al., 2025). For $T_N = \{0, \ldots, N\}$ it selects those token logits, normalizes across the grid, and minimizes $\sum_i q_i(y - i/N)^2$. Its released Qwen configuration uses T_9, rank-8 LoRA with scale 32 on query/value projections, learning rate 10^{-5}, effective batch 24, three epochs, and epoch-wise LR multiplier 0.85. The released Qwen command also includes answer-generation SFT loss.

We teacher-force the judge's original A/B verdict and place confidence directly after it: i.e. $y = 1$ means stable, and flip score is one minus expected confidence. The SFT component trains the model to reproduce the original verdict token, preserving the native answer loss without letting the scorer change the judge's verdicts. The reported run otherwise keeps that released configuration unchanged: T_9, rank-8 LoRA with scale 32 on query/value projections, learning rate 10^{-5}, effective batch 24, three epochs, the 0.85 epoch-wise multiplier, and the verdict SFT term.

APRICOT. APRICOT embeds questions with all-mpnet-base-v2, standardizes embedding dimensions, clusters with HDBSCAN (minimum cluster size three, minimum samples one), and assigns each training answer its cluster's empirical accuracy (Ulmer et al., 2024). DeBERTa-v3-base then receives question plus answer and regresses to that soft target with MSE. The released main TriviaQA run uses 600 steps, batch 32, cosine decay with 10% warm-up, gradient cap 10, learning rate 3.369×10^{-5}, and weight decay 0.008936.

Our adaptation clusters instruction text only, and replaces cluster accuracy by cluster *stability* (not flipping)

Supervised uncertainty estimation (Gb-S and Wb-S). Liu et al. formulate uncertainty estimation as ordinary supervised regression from a labeled output to its correctness score (Liu et al., 2024b). For a multiple-choice output, their gray-box model (Gb-S) normalizes logits over the declared answer tokens and uses predictive entropy plus the sorted choice probabilities. Their white-box model (Wb-S) retains every Gb-S feature, and also includes the middle-layer state at the final answer token (the

shades of 'gray' here denote the level of access the auxiliary model has to the hidden state of the judge model). Gb-S fits a 100-tree random-forest regressor with depth four. Wb-S fits 150 trees with depth eight and 45 candidate features per split. Both use random seed zero.

Here the labeled output is the adapter-off judge's verdict and its target is flip rather than correctness. Qwen2.5-7B is both target and tool model, as required for Wb-S on the white-box judge whose uncertainty is being estimated. The judge has three possible answer tokens, A/B/U, rather than the paper's four MMLU choices. As a result, Gb-S has entropy plus three sorted probabilities. Wb-S reads Qwen's middle block after teacher-forcing the cached A/B verdict.

Proxy-based uncertainty-aware reward model (Proxy-URM). Lee et al. train a Llama-2-7B reward model for one epoch with the Bradley-Terry preference loss (Lee et al., 2024). Its uncertainty comes from the scoring head: at inference the transformer is frozen in eval mode, but the head's two dropout layers stay active, so twenty-five passes over a pair give twenty-five reward gaps. Through the Bradley-Terry sigmoid those gaps become a logit-normal distribution over the preference probability, which the primary Balanced Entropy summarizes by combining predictive, aleatoric, and epistemic uncertainty. We also retain the epistemic (mutual information) and aleatoric (expected entropy) components separately. We take the implementation from the authors' released repository,[7] and change nothing in it: its reward-model command fixes rank-8 LoRA (scale 32, dropout 0.1) on query/key/value/output projections, learning rate 10^{-4}, effective batch eight, and cosine decay, its loader builds the token-level head ReLU–Dropout(0.2)–Linear(d, d, no bias)–ReLU–Dropout(0.2)–Linear(d, 1), and its trainer fixes the twenty-five dropout samples.

We make one endpoint substitution: chosen and rejected come from the adapter-off judge's A/B verdict rather than a human preference label. Proxy-URM therefore trains on preference labels alone (never sees a flip label). We score its Balanced Entropy against held-out flips. The comparison asks whether preference-trained uncertainty transfers to judge instability without instability supervision. As dropout is confined to the head, the backbone is deterministic and the release's per-draw reruns of it are redundant. We encode each response once and draw the same 25 masks through the unchanged head, so inference costs two response encodings and 50 lightweight head evaluations rather than 50 full backbone passes, leaving the sampled distribution unchanged.

LoRA deep ensemble. Balabanov and Linander interpret independently fitted LoRA adapters as a mixture posterior and use ensemble predictive entropy and mutual information for uncertainty quantification (Balabanov & Linander, 2025). Their uncertainty experiments use five Mistral-7B members. Every member has rank-8, scale-32 adapters only on query and value projections, with LoRA-B initialized to zero and LoRA-A by Kaiming uniform. It trains for ten epochs using Adam, learning rate 5×10^{-6}, and batch eight. The objective is next-token cross-entropy plus the authors' selected Gaussian-prior penalty $\sum_\ell \|\Delta W_\ell\|_F^2$ (their $\lambda/2 = 1$). Their reported uncertainty distribution retains the declared answer-token probabilities and aggregates the rest of the vocabulary into an "other" class.

Our five members predict S/U rather than a five-way multiple-choice answer. The residual vocabulary remains one "other" class, so posterior averaging, predictive entropy, and mutual information are unchanged. We additionally report ensemble-mean $p(\mathrm{U})$ because U is now the directly supervised event. UltraFeedback response pairs are much longer than CommonsenseQA. We therefore use the paper's released 512-token MMLU setting rather than its 128-token CQA setting, with a fixed 120/160/160 allocation to preserve the instruction, both responses, and the S/U query. The paper does not state numerical precision or an attention backend. Our H200 jobs use bfloat16 SDPA and gradient checkpointing. These are memory/execution choices, not changes to the fitted parameters or loss, but they can change floating-point rounding so are therefore disclosed for exact reproduction purposes.

BLoB. BLoB (Bayesian LoRA by Backpropagation) learns a Gaussian variational posterior throughout fine-tuning rather than Bayesianizing a fitted adapter post hoc (Wang et al., 2024b). Our corrected adaptation applies BLoB directly to the same Qwen2.5-7B judge, rather than training a second Llama classifier. The input is the ordinary pairwise-judging prompt with one visible A/B presentation, and the target is the adapter-off judge's verdict. It never sees a flip label during training, and we use flips only to evaluate whether posterior-predictive uncertainty forecasts a switch. This is deliberately distinct from our method: training BLoB to place mass on U would amount to Bayesianizing our own objective.

[7] https://github.com/P-B-U/proxy_based_uncertainty, commit b631707.

We retain the released variational recipe: an 8-bit frozen backbone, rank-8, scale-16, zero-dropout LoRA on query, value, and LM-head projections, and a Gaussian posterior over each LoRA-A matrix. The released low-rank Flipout estimator draws one posterior sample per training forward. Mean parameters minimize A/B cross-entropy with AdamW (10^{-4}, $\epsilon = 10^{-6}$), while the posterior parameters receive the separately scheduled KL step with SGD at 0.01. We retain $\rho \sim \mathcal{U}(.05/\sqrt{2}, .05)$, standard deviation ρ^2, prior standard deviation 0.2, dataset-rescaling exponent $\pi/8$, the released implementation's cyclical $2^i/(2^{M+1} - 1)$ KL weights, 6% linear warm-up, batch four, and 5,000 updates.

Two implementation changes are required. First, Qwen replaces the released Llama-2 backbone because the competitor is uncertainty in the judge being evaluated, not uncertainty from an additional model. Second, the maximum length is 2,304 tokens (up to 1,024 instruction and 400 tokens per response) rather than the release's 300-token multiple-choice setting. All posterior, optimizer, schedule, rank, and sampling choices remain fixed.

At inference, every posterior draw processes the full prompt and scores the single next-token distribution over A and B. If $p_s(A)$ is draw s's probability, the primary score is the binary entropy of $\bar{p}(A) = N^{-1} \sum_s p_s(A)$. We also report $N \in \{1, 5\}$, posterior mutual information, predictive variance, and the posterior-mean-weight diagnostic. Because the adapted Qwen is itself the judge, N means exactly N full-model evaluations, each with one output token, with no separate judge call. Each seed evaluates all 26,880 held-out calls. Ten draws buy only 0.011 AUROC over one while remaining 0.220 below our one-pass score. Zero-shot transfer is evaluated as for every other estimator: one presentation per prediction on all three datasets, with intervals clustered by question or response pair.

CCPS. CCPS perturbs the final hidden state in a targeted adversarial direction and uses output-distribution changes as confidence evidence (Khanmohammadi et al., 2025). Its multiple-choice path records 15 unperturbed features and, for 15 perturbed quantities, their minimum, maximum, mean, and standard deviation over five radii: 75 features. These include token probability/logit, entropy, margin, Jacobian norm, radius to argmax flip, perturbation-effect integral, KL/JSD, cosine similarities, and hidden distance. The released pipeline standardizes on training data, learns a 64–32–16 ELU contrastive encoder with eight-dimensional output, then a 48–24–12 classifier. We train both stages at LR 10^{-4} and weight decay .01. Large-model commands use 5,000 steps per stage.

Our port probes the judge's A/B verdict token, at radii 4, 8, 12, 16, and 20, and trains on the flip label. A generated rationale or answer sequence would change the judge being evaluated. Inference adds five full-vocabulary LM-head projections. Both training stages use the released 5,000 steps.

Table 2: Trained uncertainty estimators, grouped by inference cost. Entries are flip AUROC from one presentation per prediction. UF is held-out UltraFeedback. GSM8K, TriviaQA, and HH-RLHF use the same single-presentation scoring rule. Transfer scores apply each estimator unchanged, with no fitting on the target domain. A dash means the method cannot be transferred this way. GPU times cover complete train–validate–test jobs and upper-bound fit time. Wb-S and Gb-S share one feature-generation job. The trained-$P(\mathrm{U})$ and BLoB transfer rows are means across their respective seeds 0, 1, and 2. BLoB cells also show sample SD. The inference column counts full-model evaluations per prediction, and no score combines the two presentation orders. Extra-context methods occupy the lower block and use open markers in Figure 1.

Method	UF	GSM8K	TriviaQA	HH-RLHF	measured job	inference	adaptation
One full-model evaluation							
trained $P(\mathrm{U})$ (3 seeds)	.842 ± .002	.800	.874	.839	∼ 1 A100-h	1: adapted judge	ours
Radharapu et al. linear probe (Radharapu et al., 2026)	.798	.791	.843	.779	0.701 H200-h	1: judge + affine head	endpoint adaptation
Liu et al. Wb-S (Liu et al., 2024b)	.778	.749	.852	.643	0.755 H200-h shared	1: judge + state/forest	endpoint adaptation
LitCab (Liu et al., 2024d)	.756	.693	.838	.712	0.588 H200-h	1: judge + calibrated logits	endpoint adaptation
Liu et al. Gb-S (Liu et al., 2024b)	.618	.547	.632	.703	0.755 H200-h shared	1: judge + logits/forest	endpoint adaptation
BLoB N=1 (Wang et al., 2024b)	.611 ± .007	.460 ± .028	.538 ± .011	.698 ± .006	17.774 H200-h / 3 seeds	1: sampled Bayesian judge	verdict-only adaptation
CCPS (Khanmohammadi et al., 2025)	.595	–	–	–	0.528 H200-h	1: judge + 5 projections	endpoint adaptation
UaIT (Liu et al., 2024c)	.590	.599	.697	.651	3.349 H200-h	1: adapted judge	endpoint adaptation
Semantic Entropy Probe (Kossen et al., 2024)	.572	.543	.686	.644	0.662 H200-h	1: judge + logistic probe	endpoint adaptation
SaySelf (Xu et al., 2024)	.537	.593	.691	.480	4.336 H200-h	1: adapted judge	endpoint adaptation
Additional full-model context(s); open markers in figures							
ConfTuner (Li et al., 2025)	.789	–	–	–	5.418 H200-h	2: judge + external Qwen	endpoint adaptation
Proxy-URM (Lee et al., 2024)	.659	–	–	–	2.426 H200-h	3: judge + 2 Llama-2 encodings	endpoint adaptation
BLoB N=5 (Wang et al., 2024b)	.620 ± .010	.443 ± .030	.527 ± .011	.717 ± .007	shared above	5: sampled Bayesian judge	verdict-only adaptation
BLoB N=10 (Wang et al., 2024b)	.622 ± .010	.440 ± .032	.526 ± .013	.723 ± .006	shared above	10: sampled Bayesian judge	verdict-only adaptation
APRICOT (Ulmer et al., 2024)	.606	.517	.429	.533	0.253 H200-h	2: judge + DeBERTa	endpoint adaptation

C TRAINING DETAILS

Table 3 gives the full training configuration for our main judge, Qwen2.5-7B-Instruct (Section 3.2). Each run trains the LoRA adapter for one epoch over 32,000 examples, which is 2,000 optimizer updates at an effective batch size of 16. We evaluate the final checkpoint.

Table 3: Training configuration for the Qwen2.5-7B-Instruct judge.

Setting	Value
Objective	Cross-entropy on the answer token (A, B, or U), full vocabulary
Trainable weights	LoRA on q/k/v/o in every attention block
Rank / LoRA scale / dropout	16 / 32 / 0.05
Optimizer	AdamW, learning rate 10^{-4}
Adam betas / epsilon / weight decay	$(0.9, 0.999)$ / 10^{-8} / 0.01
Learning-rate schedule	Linear decay to 0.05 of initial rate
Microbatch / accumulation / effective batch	4 / 4 / 16
Epochs / optimizer updates	1 / 2,000
Numerical precision	bfloat16 weights; float32 softmax for scoring
Memory	Gradient checkpointing
Prompt / response truncation	1,024 tokens / 400 tokens per response
Training seeds	0, 1, 2 (train/held-out split fixed with seed 0)

Each example is formatted with the model's chat template. The training target is a single answer token, A, B, or U, placed directly after the prompt, and the loss ignores the prompt and padding tokens. We use the system prompt from Section 3.1 for both the trained judge and the adapter-off control, and read out the verdict and $P(\mathrm{U})$ as in Equation 4.

Training data. We use 11,200 instructions sampled from UltraFeedback, each with four responses. Each instruction gives six response pairs, and judging every pair in both orders gives 134,400 judge calls. We split by instruction (seed 0), holding out 2,240 instructions (26,880 calls) for evaluation. The remaining 107,520 calls form the training pool, from which each run draws a random subset of 32,000. The larger judges (Qwen2.5-14B, Qwen2.5-32B, and Qwen2.5-VL-3B) use a separate set of 4,000 instructions with its own split, so their held-out sets are smaller.

Models and compute. We use `Qwen/Qwen2.5-1.5B-Instruct`, `Qwen/Qwen2.5-7B-Instruct`, `Qwen/Qwen2.5-14B-Instruct`, `Qwen/Qwen2.5-32B-Instruct`, `meta-llama/Llama-3.1-8B-Instruct`, `mistralai/Mistral-7B-Instruct-v0.3`, and, for image inputs, `Qwen/Qwen2.5-VL-3B-Instruct`. Each judge gets its own adapter. We will release the trained adapters, evaluation outputs, and our code upon publication. Training one adapter takes about one A100 GPU-hour, and the complete run for the main 7B judge took about 2.5 A100-80GB GPU-hours.

D WHY USE FLIPPED JUDGMENTS AT ALL?

We argue that this method of collecting uncertainty labels is more useful than methods which overtly focus on collecting judgments which are 'correct', or not. If one limits label collection to domains with ground truth 'correctness' labels, one (a) requires (expensive) human-labels over each judgment for 'correctness', and (b) omits an entire class of uncertainty examples over two *incorrect* completions, which, nonetheless, produced a flip. Furthermore, our method of collecting uncertainty targets in the absence of a measure of 'correctness' admits the possibility of doing uncertainty estimation in entirely open-ended domains, such as preference over works of art, where a notion of 'correctness' is infeasible.

E PROBABILITY CALIBRATION

Calibration is measured on the same 26,880 held-out calls used for flip AUROC. We use the raw U-token probability after normalizing over A/B/U and do not fit a post-hoc calibration map. Expected calibration error uses ten equal-width bins, and confidence intervals resample whole prompts, retaining both presentation orders together.

Table 4: Raw held-out calibration by training seed. The empirical flip rate is 0.3737 in every row.

Seed	mean $P(\mathrm{U})$	AUROC	Brier	NLL	ECE
0	.3538	.8432	.1554	.4674	.0200
1	.3565	.8396	.1573	.4729	.0247
2	.3754	.8422	.1552	.4675	.0083
mean	.3619	.8417	.1560	.4692	.0176

Figure 2 uses the same ten equal-width bins, with each prediction scored from a single presentation. For the trained curve, each bin's mean prediction and observed flip rate are computed per seed and then averaged over the three seeds. The untrained curve does not depend on the seed and appears once. Shading gives pointwise 95% intervals from 2,000 bootstrap resamples of whole prompts, with both presentation orders kept together and the same resamples used for both curves. For untrained $P(\mathrm{U})$, 93.3% of predictions fall in the lowest bin (0–10%).

For seed 0, 10,000 prompt-cluster bootstrap resamples give AUROC $[.836, .850]$, Brier score $[.152, .159]$, negative log-likelihood $[.458, .477]$, and expected calibration error $[.013, .028]$. The Brier score of the constant-prevalence predictor is .2341.

F CROSS-DOMAIN TRANSFER

Here every estimator predicts a flip from a single presentation of each pair. Methods that require additional model evaluations for one presentation still incur that cost, as listed in Table 2. Table 5 gives the trained and untrained $P(\mathrm{U})$ scores in each domain with paired intervals.

The evaluation sets contain 1,319 GSM8K questions (2,638 presentations), 2,000 TriviaQA questions (4,000 presentations), and 1,500 HH-RLHF pairs (3,000 presentations).

Table 5: Cross-domain transfer from one presentation. Trained AUROC is mean ± sample standard deviation across three adapter seeds. Intervals are paired 95% intervals for trained minus untrained $P(\mathrm{U})$.

Dataset	Trained $P(\mathrm{U})$	Untrained $P(\mathrm{U})$	Difference, 95% CI
GSM8K	$.800 \pm .004$	.633	.167 $[.142, .194]$
TriviaQA	$.874 \pm .005$	.746	.128 $[.108, .146]$
HH-RLHF	$.839 \pm .003$	.620	.219 $[.193, .246]$

Table 6: Trained $P(\mathrm{U})$ single-presentation AUROC for each existing adapter seed. The same held-out calls and original source-judge labels are used for all three seeds.

Dataset	Seed 0	Seed 1	Seed 2
GSM8K	.8005	.7967	.8037
TriviaQA	.8771	.8676	.8768
HH-RLHF	.8382	.8422	.8354

Confidence intervals are 95% bootstrap intervals from 2,000 resamples of whole questions, with both presentation orders of a question kept together and the same resamples used for every method, so differences between methods are paired. For trained $P(\mathrm{U})$ we report the mean of the three seeds' AUROCs, not the AUROC of their averaged predictions. The intervals reflect uncertainty from the test sample only, not from retraining, and are not corrected for multiple comparisons.

The strongest competing point estimate is Radharapu et al.'s probe on GSM8K (0.791), Liu et al.'s Wb-S on TriviaQA (0.852), and Radharapu et al.'s probe on HH-RLHF (0.779). The trained-$P(\mathrm{U})$ advantages are respectively 0.0091 ($[-0.0036, 0.0218]$), 0.0217 ($[0.0087, 0.0340]$), and 0.0593 ($[0.0445, 0.0737]$). Thus, the trained score is numerically highest in all three domains, but the GSM8K comparison does not establish a clear advantage over the probe.

Scores are similar in the two presentation orders: trained $P(\mathrm{U})$ reaches 0.813 and 0.787 on GSM8K, 0.880 and 0.867 on TriviaQA, and 0.842 and 0.835 on HH-RLHF. Per-order and per-seed results will be released with our code.

G Model sizes, families and modalities

Table 7 lists the held-out flip AUROC behind Figures 4 and 9, for every judge and for each training-set size.

Table 7: Held-out flip AUROC for every judge and for the training-set size sweep. Top: each judge is trained on its own flips and evaluated on its own held-out calls. Untrained $P(\mathrm{U})$ and A/B margin are adapter-off scores under the same A/B/U prompt. All rows except Qwen2.5-7B are a single adapter (seed 0). The 14B, 32B and VL-3B judges use the separate 4,000-instruction split (Appendix C). Bottom: Qwen2.5-7B trained on subsets of the 107,520-row training pool.

Judge	Held-out calls	Flip rate	Untrained $P(\mathrm{U})$	A/B margin	Trained $P(\mathrm{U})$
Qwen2.5-1.5B	26,880	55.5%	0.510	0.510	0.808
Qwen2.5-7B (3 seeds)	26,880	37.4%	0.560	0.644	0.842±0.002
Qwen2.5-14B	9,276	22.5%	0.594	0.750	0.879
Qwen2.5-32B	9,276	17.5%	0.702	0.789	0.884
Llama-3.1-8B	26,880	54.2%	0.536	0.619	0.916
Mistral-7B-v0.3	26,880	79.6%	0.560	0.519	0.925
Qwen2.5-VL-3B (text)	9,276	43.2%	0.521	0.565	0.853

Training rows (Qwen2.5-7B)	Trained $P(\mathrm{U})$ AUROC
2,000	0.773
8,000	0.808
32,000	0.842±0.002 (seeds 0.843, 0.840, 0.842)
107,520 (all)	0.856

G.1 Training-set size

Figure 9 shows held-out flip AUROC on Qwen2.5-7B as the number of ordered training rows grows from 2k to the complete 107,520-row training split (Section 4.4).

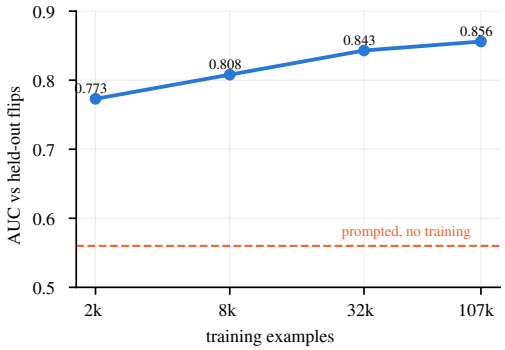

Figure 9: AUC against training-set size. The $P(\mathrm{U})$ score strengthens smoothly with labels and is unsaturated at the full label set.

G.2 Image-conditioned results

One fixed Qwen2.5-VL-3B training seed evaluates 1,000 questions from each dataset, with both response orders kept together in every resample.

Table 8: Image-conditioned flip AUROC. Brackets give 95% confidence intervals from resampling whole questions, and Δ is adapter on minus off.

Dataset	flip rate	adapter on	adapter off	Δ
GQA	3.2%	.946 [.924, .963]	.847 [.786, .900]	.098 [.054, .148]
VQAv2	1.2%	.973 [.954, .987]	.936 [.898, .968]	.037 [.014, .064]

H Capability retention and label ablations

H.1 Training also improves held-out preference accuracy

We evaluate both the trained and the untrained judges' preference accuracy on RewardBench. Here, the adapted judge retains the A/B/U prompt, though to obtain an A/B verdict, we ignore $P(\mathrm{U})$ and select whichever of A or B receives more probability. Against the adapter-off model under the original A/B prompt, Qwen2.5-7B's single-call accuracy rises from 79.70% to a mean of 83.00%, and Llama-3.1-8B's from 70.52% to 71.89%. Table 9 reports all three seeds for each model.

Table 9: RewardBench accuracy (%) for every adapter seed. Each presentation is scored separately; no verdict combines the two orders. The baseline uses the original A/B prompt; trained judges use A/B/U and select the higher-probability A/B verdict, including calls where U is most probable. The final column is the mean and sample standard deviation across seeds.

Model	Untrained	Seed 0	Seed 1	Seed 2	Mean $\pm$ SD
Qwen2.5-7B	79.70	83.33	83.22	82.46	83.00 ± 0.47
Llama-3.1-8B	70.52	71.91	72.21	71.56	71.89 ± 0.33

Table 10: Ordinary benchmark accuracy (%) for Qwen2.5-7B before training and with the seed-0 adapter. Both conditions use the same questions, greedy decoding and normal answer formats, with no U option. Intervals are 95% paired question-bootstrap intervals for the change in percentage points (20,000 draws). They describe test-set uncertainty for seed 0, not variation across training seeds. BBH is BIG-Bench Hard; GPQA-D is GPQA-diamond.

Benchmark	Questions	Before	After	Change	95% interval
MMLU	1,000	67.90	68.50	$+0.60$	$[-1.40, +2.60]$
MMLU-Pro	2,000	44.65	44.70	$+0.05$	$[-1.45, +1.55]$
GSM8K	1,319	91.43	91.36	-0.08	$[-1.14, +0.99]$
BBH	2,700	29.78	29.96	$+0.19$	$[-0.93, +1.30]$
TriviaQA	2,000	49.35	49.45	$+0.10$	$[-1.35, +1.55]$
SQuAD2	2,000	25.85	26.00	$+0.15$	$[-0.70, +1.00]$
GPQA-D	198	24.24	30.81	$+6.57$	$[+0.00, +13.13]$

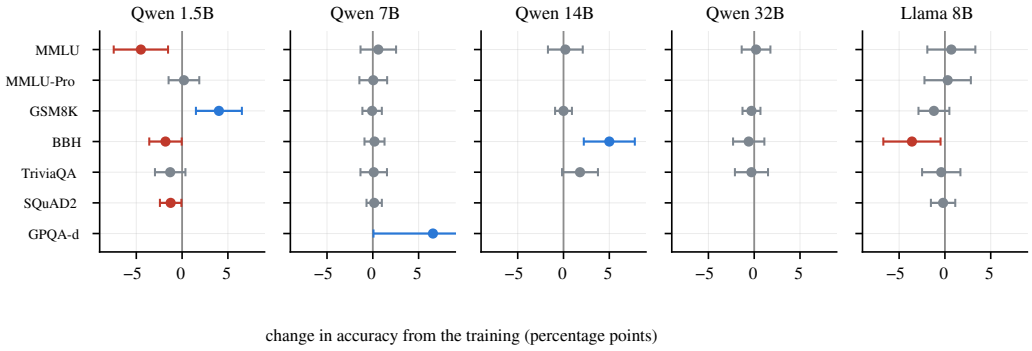

Figure 10: Capability retention after training. These are single-seed evaluations (seed 0), not averages across training seeds. Change in accuracy with the adapter attached, on the ordinary form of each benchmark with no abstention permitted, greedy decoding, and identical questions. Bars are 95% intervals on the paired difference and color marks intervals excluding zero *before* multiplicity correction (the text applies Holm within each model). From 7B upward nothing degrades within the Qwen family. Llama-3.1-8B is flat in aggregate. At 1.5B the same rank-16 adapter is a far larger relative perturbation of the model and the effects are mixed and small.

H.2 Label ablations: the gain does not come from generic training on judging data

Could the improvement observed on RewardBench simply come from the additional training of the judge on the same prompts? To test this, we train an A/B-only control using the same calls, LoRA setup, optimizer, data budget, and training schedule. The only difference is that this control never receives a U target. Its A/B margin reaches 0.601 ± 0.012 flip AUROC, below the 0.644 achieved without any adapter. By comparison, the A/B margin after our training reaches 0.761 ± 0.007, and $P(\mathrm{U})$ reaches 0.842 ± 0.002. We conclude that training on judging data alone does not explain the gain (Table 11).

Table 11: Flip AUROC on identical held-out calls. Values are mean ± sample standard deviation across three training seeds, and intervals use the seed-0 paired prompt-cluster bootstrap.

Training and score	mean ± SD	seed-0 95% CI
flip-placed U, $P(\mathrm{U})$ (ours)	.842 ± .002	[.836, .850]
flip-placed U, inverse A/B margin	.761 ± .007	[.748, .764]
adapter off, inverse A/B margin	.644	[.635, .653]
A/B-only training, inverse A/B margin	.601 ± .012	[.578, .598]

Label-content ablations. An A/B-only self-distillation control uses the same calls but always targets the letter originally selected by the adapter-off judge. RewardBench accuracy changes by only +0.001, so another training epoch cannot explain the gain. A second control assigns the same number of U targets to randomly selected calls. It recovers about one third of the accuracy gain, perhaps by learning how often U occurs overall, but not which judgments are unstable. Its difference in mean $P(\mathrm{U})$ between unstable and stable pairs is 0.001, versus 0.327 when U is assigned to actual order inversions (Figure 11).

Table 12: RewardBench accuracy (2,985 gold-labeled pairs, out-of-distribution for the tuning) under label ablation. Same data, compute, and prompt. Only the targets differ.

Adapter	call acc.	pair acc. (both orders)	U-rate
none (adapter off)	.797	.676	—
A/B only (self-distilled)	.798	.678	0%
random-U (matched volume)	.808	.698	0.15%
flip-placed U (ours)	**.833**	**.757**	17.8%

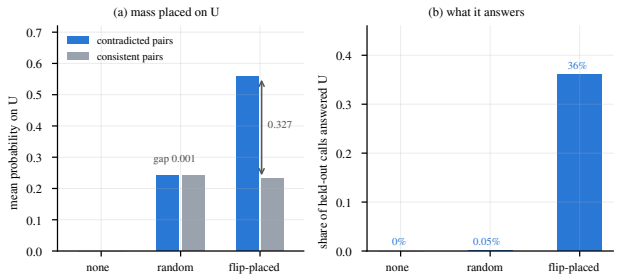

Figure 11: Effect of U-target placement at fixed target count. **(a)** Mean U probability on order-inconsistent and order-consistent pairs: random placement yields nearly constant mass, whereas flip placement conditions U on instability. **(b)** Resulting predictions. Constant mass of 0.24 never exceeds the winning letter.

Table 13 compares U labels taken from order inversion with two label sources built from sampled consistency, at a matched budget of 32k judgments. The exact within-order control assigns U to the 36.4% of calls with the largest disagreement probability, $2p_Ap_B$, without using any per-example order-flip label. It therefore removes differences in row count and U-label frequency from the comparison.

Table 13: Matched 32k-judgment comparison of U-label sources. AUROC is mean $\pm$ sample standard deviation across three training seeds.

U-label source	training rows	judgments	U targets	AUROC
order inversion (ours)	32k	32k	36.4%	$.842 \pm .002$
four samples in one order	8k	32k	14.6%	$.633 \pm .004$
exact within-order disagreement	32k	32k	36.4%	$.636 \pm .003$

I JUDGE SPECIFICITY OF THE $P(\mathrm{U})$ SCORE

We hold 13,440 response pairs from 2,240 held-out prompts fixed and ask whether Qwen's trained $P(\mathrm{U})$ better predicts Qwen's own flips than Llama-3.1-8B's or Mistral-7B's flips on the same pairs (Binder et al., 2024). Each judge's flip is defined per pair, so to compare the three judges on identical pairs we need one score per pair. We take S to be the larger of the two orders' $P(\mathrm{U})$ values. This uses both presentations of a pair, but neither call sees the other order's verdict, so S does not observe the flip. The question here is whose uncertainty the score tracks, and the same S is compared against all three sets of flips, so the cost of two calls plays no part in the comparison. Table 14 reports mean AUROCs across three adapter seeds.

Trained $P(\mathrm{U})$ predicts Qwen's own flips better than Llama's by 0.180 AUROC (95% CI $[0.167, 0.192]$), and better than Mistral's by 0.146 ($[0.132, 0.160]$).

Intervals use 2,000 bootstrap resamples of prompts, keeping all pairs from a sampled prompt together and averaging the three fixed-seed AUROCs within each resample.

Table 14: Model specificity against pre-training flips (pair level). Qwen labels are held-out flips from the adapter-off Qwen model whose training-split flips supervise S. Other labels come from adapter-off Llama and Mistral judges on the same pairs. The symmetric difference restricts to pairs where exactly one model flips. Results average three adapter seeds, and intervals use 2,000 paired prompt-cluster bootstrap resamples.

	o = Llama-3.1-8B	o = Mistral-7B
$\mathrm{AUC}(S, \varphi_{\mathrm{Qwen}})$	**0.862**	**0.862**
$\mathrm{AUC}(S, \varphi_o)$	0.683	0.716
Δ_o (95% CI)	0.180 [.167, .192]	0.146 [.132, .160]
AUC within symmetric difference	0.766	0.767

J Consistency and confidence

J.1 Forced-choice consistency and confidence

We know that our training method improves performance in detecting *unstable* judgments, however, what we'd also like to know, is how our training method impacts models' decision boundaries on *stable* decisions.

To assess this, we replace the A/B/U question in the judge prompt with an A/B-only question, and, as an extra safeguard, we force the model to choose only between A and B at the decision position[8]. We run the same evaluation with the uncertainty adapter on and off and compare the resulting A/B margins.

We see that training the model to have an uncertainty adapter redistributes confidence in exactly the desired direction (Figure 7). On pairs the judge handles consistently, the mean A/B probability margin *with the uncertainty adapter on* rises from 0.898 to 0.931. On pairs where it contradicts itself, the margin falls from 0.873 to 0.794. The separation (in margin) between stable and unstable pairs therefore grows more than five-fold, from 0.025 to 0.137, i.e. even in induced circumstances where the judge can no longer output the 'U' token it was trained to output, we can still observe that the judge's output distribution over the choice has less entropy over stable pairs, and higher entropy over unstable pairs.

The judge's order consistency improves as well. Under this forced-choice protocol, the primary Qwen adapter's flip rate falls from 37.4% to 23.1%, a 38% relative reduction. Llama's flip rate similarly falls from 54.2% to a mean of 32.8% across three adapter seeds.

J.2 Does the trained $P(\mathrm{U})$ predict the trained judge's own flips?

Fine-tuning can bring about a change in the judge's decisions, so we also test whether the score predicts a judge's flips *after* training. For each of the three Qwen2.5-7B training seeds, we use the same A/B/U prompt that produces the uncertainty score, taking whichever of A or B is more probable as the verdict. Under this same prompt, the flip rate falls from 41.0% with the adapter off to 26.2% after training (averaged across seeds). Despite this change, trained $P(\mathrm{U})$ predicts the trained judge's own flips at 0.816 ± 0.010 AUROC, compared with 0.562 ± 0.003 for untrained $P(\mathrm{U})$ and 0.738 ± 0.005 for uncertainty measured by the trained judge's A/B probability margin. Each prediction uses $P(\mathrm{U})$ from a single presentation order, and both orders are needed only to establish the flip label. The trained score therefore remains useful for identifying the judge's remaining inconsistencies, even after training makes it more consistent. Here $\pm$ is the sample standard deviation across the three training seeds.

J.3 Does reasoning help reduce flips?

Generating reasoning tokens before the verdict does not stop a judge's judgments from flipping. We prompt the adapter-off Qwen2.5-7B to reason step by step before answering, on 3,000 held-out pairs

[8]We extract the next-token probabilities of A and B, renormalize over those two tokens, and select the more probable one (U therefore cannot be returned in this forced-choice evaluation)

(Appendix A). On the 2,026 pairs where its reasoning reaches a verdict in both orders, those verdicts still flip on 34.4% of pairs, against 35.4% for its single-token verdicts on the same pairs.

Nor does reasoning give a good uncertainty signal. We sample five reasoning chains in each order and score each pair by how well its chains agree, taking the lower agreement of the two orders. This predicts whether the reasoned verdict flips at only 0.648 AUROC, despite roughly 2,600 generated tokens per pair. Trained $P(\mathrm{U})$, which generates no extra tokens, reaches 0.755 (Table 15).

Table 15: Predicting when the reasoned verdict flips. The adapter-off Qwen2.5-7B reasons step by step before answering (one greedy chain per order, at most 320 tokens), and a pair flips if its reasoned verdicts pick different responses in the two orders. AUC is over the 2,026 of 3,000 held-out pairs where both chains reach a verdict (flip rate 34.4%). Every score combines the pair's two presentation orders. Chain agreement is the lower of the two orders' agreement among five chains sampled at $T = 0.7$, answer margin is the smaller of the two A/B margins, and both $P(\mathrm{U})$ rows take the larger of the two orders' $P(\mathrm{U})$.

Signal	AUC	extra tokens per pair
reasoning-chain agreement	0.648	∼2,600
answer margin (adapter off)	0.651	0
untrained $P(\mathrm{U})$ (adapter off)	0.540	0
trained $P(\mathrm{U})$	**0.755**	0

J.4 Output probabilities are a poor estimator of answer 'confidence'

A judge's probability on its chosen answer is a poor guide to whether that answer is stable. Flips remain common even among its most confident answers (Figure 12): above 95% confidence, Qwen2.5-7B's answers still flip 34% of the time, barely below its 37% overall rate.

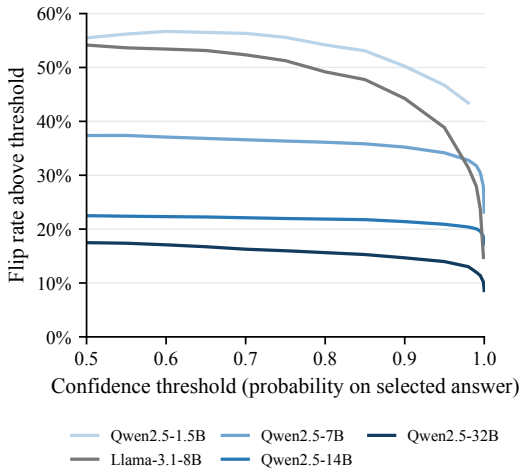

Figure 12: Output probabilities do not identify unreliable judgments. Flip rate above each answer-probability threshold for five text-only models. Curves are drawn only where at least 5% of calls qualify.

K Online routing protocol and inference FLOPs

In the dynamic routing setting for the judge model(s), every non-escalated verdict is the adapted 7B model's forced A/B choice. Escalation replaces that verdict with the expert's (32B) A/B choice. U wins only when its probability strictly exceeds both A and B. Ties retain the small-model verdict.

Calibration and evaluation split. We calibrate the escalation threshold on 100 RewardBench questions sampled without replacement, one random presentation each, setting it to the 70th percentile of their $P(\mathrm{U})$ scores (0.263, 0.243 and 0.269 for the three adapter seeds), which needs no ground

truth labels or calls to the larger model. These questions are then excluded in both presentation orders, leaving 2,885 questions and 5,770 calls for every reported accuracy and compute comparison.

FLOP accounting. We use the leading-order parameter–token approximation to a dense forward pass, counting a multiply and an add as two operations (Kaplan et al., 2020). For nominal model size P_j and recorded unpadded input length $T_{j,i}$, define

$$\widehat{F}_j(T_{j,i}) = 2P_jT_{j,i}, \qquad P_7 = 7.6 \times 10^9, \quad P_{32} = 32.5 \times 10^9. \tag{5}$$

The first-token A/B/U readout uses one forward pass. For evaluation calls $\mathcal{E}$ and per-call escalation indicator r_i, the cascade estimate is

$$\widehat{F}_{\text{cascade}} = \sum_{i \in \mathcal{E}} [2P_7T_{7,i} + r_i\, 2P_{32}T_{32,i}] + \widehat{F}_{\text{cal}}. \tag{6}$$

Here $\widehat{F}_{\text{cal}}$ is the cost of the 100 additional 7B calibration calls for the calibrated rule and zero for the fixed rules. The reference is $\widehat{F}_{32} = \sum_{i \in \mathcal{E}} 2P_{32}T_{32,i}$, with saving $1 - \widehat{F}_{\text{cascade}}/\widehat{F}_{32}$. Both models process 2,504,304 input tokens over the evaluation set, and which tokens reach 32B depends on the route mask. The reference therefore costs 1.628×10^{17} estimated FLOPs. Table 16 amortizes total compute over the 5,770 evaluation calls, including calibration startup where applicable. These are parameter-count FLOP estimates, not profiler measurements: they omit explicit attention, padding and kernel overhead, and exclude training and data-collection costs.

Accuracy–compute sweep. For Figure 8, we reuse the same calibration/evaluation split and evaluate target expert fractions from 0% to 100% in one-percentage-point increments. For each interior target b and adapter seed, we set the threshold to the $(1 - b)$ quantile of the 100 calibration $P(\text{U})$ scores, with the same `higher` convention, then escalate evaluation calls whose scores exceed it. The endpoint cascades explicitly route none or all. No evaluation labels or scores enter threshold selection. Every point on the sweep includes the 100-call calibration cost. The separately marked fixed rules and standalone models require no calibration. Routing every call to 32B still pays for the 7B pass, so this cascade endpoint costs more than using 32B alone.

The always-off reference runs only the 7B judge, and the always-on reference runs only the 32B judge, without a 7B screening pass. Faint curves and marker ranges show the spread across the three adapters. Each x-coordinate is the compute for one evaluation workload at that threshold, averaged across seeds.

Table 16: Online Qwen2.5-7B-to-32B routing on the same 5,770 held-out comparisons. Means over three adapter seeds. Estimated TFLOPs per evaluation call include the 7B pass, any 32B pass, and the calibrated rule's 100-call startup amortized over evaluation. Savings are relative to using 32B alone on every evaluation call.

Rule	32B calls (%)	Accuracy (%)	Est. TFLOPs/call	Saved (%)
7B only	0.00	82.99	6.60	76.62
$P(\text{U}) > 0.5$	16.61	85.90	11.84	58.02
U wins A/B/U	18.31	86.18	12.34	56.27
100-call calibration	30.75	87.58	16.02	43.21
32B always	100.00	88.87	28.21	0.00

Table 17: Uncertainty in the online routing results. Question-clustered 95% intervals condition on the fitted adapters and the fixed calibration set.

Rule	Accuracy (%), 95% interval	FLOPs saved (%), 95% interval
$P(\text{U}) > 0.5$	$[84.78, 87.04]$	$[56.75, 59.30]$
U wins A/B/U	$[85.06, 87.29]$	$[54.93, 57.60]$
100-call calibration	$[86.49, 88.63]$	$[41.64, 44.78]$

Uncertainty and calibration sensitivity. Intervals are 95% bootstrap intervals from 3,000 resamples of the 2,885 evaluation questions, with both presentations kept together and results averaged over the three adapter seeds. They reflect uncertainty from the evaluation sample for a fixed calibration sample. To check sensitivity to that sample, we repeat the calibration/evaluation split 200 times. With

100 calibration calls, mean accuracy is 87.45% and mean expert use 29.94%, with central 90% ranges of 86.83–87.95% and 22.67–38.59%. With 200 calibration calls, mean accuracy is 87.47% and the expert-use range narrows to 24.51–35.21%. These overlapping replays describe calibration-sample sensitivity rather than independent replications. A fixed threshold permits immediate routing but does not guarantee an exact expert-call quota on future traffic.

L Mapping Uncertainty Across Questions

Can a judge's uncertainty on a few questions tell us where it is likely to be wrong on the rest? Our premise is that if a model represents questions similarly in its latent space, it may have similar levels of uncertainty about them.

We proceed as follows. First, we measure $P(\mathrm{U})$ directly on a small sample of questions. Then we *estimate* $P(\mathrm{U})$ for every other question by interpolating from its nearest neighbors in the model's representation space. Finally, we ask whether the resulting map of uncertainty predicts where the model actually errs, and whether training improves that prediction.

L.1 Method: measure a few questions, estimate their neighbors

We use Qwen2.5-7B, Llama-3.1-8B and Mistral-7B-v0.3, each with one trained uncertainty adapter (seed 0), on the full MMLU test split (14,042 questions in 57 subjects) and MMLU-Pro test split (12,032 questions in 14 categories). Three models and two benchmarks give six comparisons. The method has four steps, outlined below.

1. Give every question a location. With the adapter disabled, the model answers each question in its ordinary multiple-choice format. We save its answer (its highest-probability option), which is what we later check to determine whether the model answered correctly, or not. We also save the model's second-most-likely option. We'll use this second-most-likely option in step 3. Finally, we also save the model's final-layer activation at the last prompt token. This activation vector is taken as the question's *location* in the model's latent space. Two questions are neighbors when their activation vectors are close in cosine distance. All computations use these full-dimensional vectors, the two and three-dimensional plots below are for illustrative purposes only.

2. Choose which questions to measure. We hold out 20% of the questions as a test set. From the remaining 80%, we pick a small set of questions, the *anchors*, on which we measure $P(\mathrm{U})$ directly. Every other question will instead receive an estimated $P(\mathrm{U})$, borrowed from its nearest anchors (step 4). We use 10% of the benchmark's questions as anchors. We want them to span the full range of question content, so we embed each question and its answer choices with `BAAI/bge-small-en-v1.5` and pick anchors by greedy k-center selection, each time adding the question furthest from those already chosen. This text embedding is used only to pick anchors.

3. Measure uncertainty at the anchors. For each anchor we ask the model to compare its answer with its runner-up using the A/B/U judging prompt, in both presentation orders, and take the larger $P(\mathrm{U})$ as the anchor's measured uncertainty u_i. We do this twice, once with the trained adapter active and once with it disabled, so that we can build two maps (a trained uncertainty map, and an untrained uncertainty map) from the same anchors and the same candidate pairs.

4. Fill in the unmeasured questions. For every question j, find its 32 nearest anchors and take a weighted average of their measured uncertainties, giving closer anchors more weight:

$$\widehat{u}_j = \frac{\sum_{i \in \mathcal{N}_{32}(j)} w_{ji} u_i}{\sum_{i \in \mathcal{N}_{32}(j)} w_{ji}}, \qquad w_{ji} = \frac{1}{\max\{1 - \cos(h_j, h_i), 10^{-6}\}}, \tag{7}$$

where h_j is the saved activation and $\mathcal{N}_{32}(j)$ is the set of nearest anchors. In plain terms, a question receives high estimated uncertainty when nearby measured questions have high uncertainty. We call the collection of these estimates the *uncertainty map*.

What differs between the trained and adapter-off maps? Only the $P(\mathrm{U})$ value measured at each anchor. Both maps share the same questions, answers, anchors, activations, neighbors and weights. If the trained map predicts errors better, that is because training made the uncertainty values more informative, not because it moved anything in representation space.

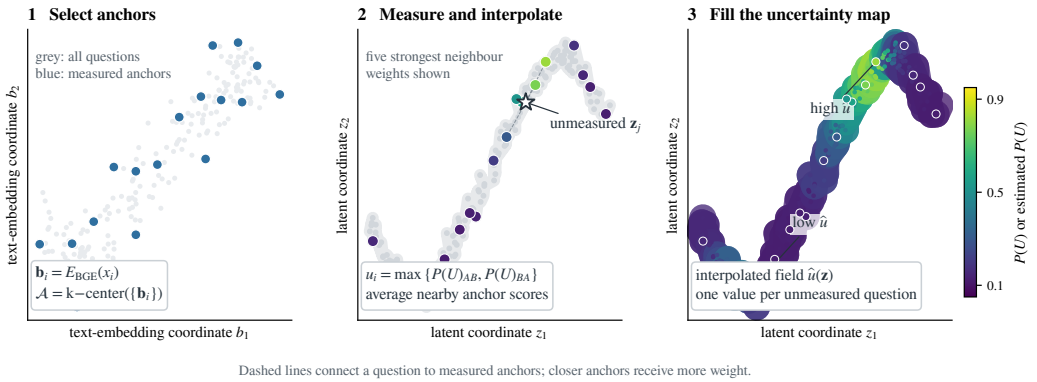

Figure 13: Measure uncertainty at a few questions, then interpolate. Left: select anchors that cover the question content using text embeddings. Middle: measure their $P(\mathrm{U})$ and find neighbors using the model's fixed adapter-off activations. Right: estimate uncertainty at unmeasured questions by averaging nearby anchor values. The drawing is schematic. The experiment uses 32 neighbors in the full activation space and 10% of questions as anchors.

L.2 Result: the trained map identifies where errors concentrate

We now check the model's answers on the held-out 20% against the correct answers, and ask whether the questions the map calls uncertain are the questions the model actually answered incorrectly. The trained map ranks held-out errors at 0.722 AUROC against 0.663 for the adapter-off map, and it wins all six comparisons (Table 18). Averaged across the six model–benchmark configurations, the model is wrong on 25.6% of the questions in the least uncertain quarter of the trained map and on 71.5% of those in the most uncertain quarter. Uncertainty measured on one set of questions therefore says something informative about errors on nearby questions that were never even measured.

Table 18: Training improves the map's error ranking in all six comparisons. AUROC for predicting which held-out questions the model got wrong, using the uncertainty map built from 10% of questions. Only the measured $P(\mathrm{U})$ values differ between the adapter-off and trained columns.

Model	Benchmark	Adapter off	Trained	Δ
Qwen2.5-7B	MMLU	0.726	0.757	+0.031
Qwen2.5-7B	MMLU-Pro	0.662	0.717	+0.055
Llama-3.1-8B	MMLU	0.704	0.740	+0.036
Llama-3.1-8B	MMLU-Pro	0.648	0.682	+0.033
Mistral-7B-v0.3	MMLU	0.642	0.749	+0.106
Mistral-7B-v0.3	MMLU-Pro	0.596	0.686	+0.091
Mean		0.663	0.722	+0.059

M Uncertainty filtering improves self-labeled DPO data

Does the judge's uncertainty tell us which of its own judgments are worth training on? To find out, we let the judge label its own preference data for DPO and ask whether discarding the pairs it is uncertain about produces a better policy.

We start from the 2,240 held-out UltraFeedback prompts and their response pairs. For every pair the judge itself picks the winner and the loser. No human and no stronger model labels anything. We then build five training sets of exactly 6,000 pairs each, one per selection rule, and train a DPO model on each, with three seeds per rule and every model starting from the same Qwen2.5-7B checkpoint. The rules are:

1. **Unfiltered**: 6,000 pairs, with no selection by any score.
2. $P(\mathrm{U})$ **filter, adapter-off labels**: the 6,000 pairs with the lowest trained $P(\mathrm{U})$, labeled by the adapter-off judge.

3. **Prompted-confidence filter**: the 6,000 pairs the judge rates most confident when asked directly.
4. **Probe filter**: the 6,000 pairs a hidden-state probe rates most stable.
5. $P(\mathrm{U})$ **filter, adapted labels**: as in rule 2, except that the adapted judge also supplies the labels. It uses its own $P(\mathrm{U})$ to choose which pairs to keep and its own A/B verdict to label them.

In rules 1 to 4 the labels come from the original, adapter-off judge, so those four differ only in which pairs are kept. Rule 5 differs from rule 2 only in which judge labels the pairs which are retained, so comparing the two isolates the effect of the adapted judge's verdicts. We also train a sixth, diagnostic model on the pairs whose verdict is the same in both presentation orders. Since that requires two calls per pair it is not a deployable filter, and it appears only in Appendix M.1. For condition a, let

$$\mathcal{D}_a = \{(x_i, y_i^+, y_i^-)\}_{i=1}^{6000},$$

where x_i is an instruction and y_i^+ and y_i^- are that arm's selected winner and loser. DPO can be derived from the KL-regularized reward objective

$$J(\pi) = \mathbb{E}_{x,\, y\sim\pi(\cdot|x)}[r(x,y)] - \beta\, \mathbb{E}_x[D_{\mathrm{KL}}(\pi(\cdot \mid x) \,\|\, \pi_{\mathrm{ref}}(\cdot \mid x))]\,. \tag{8}$$

Optimizing over the policy distribution for each x gives

$$\pi^*(y \mid x) = \frac{\pi_{\mathrm{ref}}(y \mid x)\exp(r(x,y)/\beta)}{Z(x)}, \qquad r(x,y) = \beta \log \frac{\pi^*(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} + \beta \log Z(x). \tag{9}$$

Under the Bradley–Terry preference model, the partition term cancels between the two responses. Replacing π^* by the trainable policy π_ϕ yields

$$\Pr_\phi(y^+ \succ y^- \mid x) = \sigma(\beta\Delta_\phi)\,, \quad \Delta_\phi = \log \frac{\pi_\phi(y^+ \mid x)}{\pi_{\mathrm{ref}}(y^+ \mid x)} - \log \frac{\pi_\phi(y^- \mid x)}{\pi_{\mathrm{ref}}(y^- \mid x)}, \tag{10}$$

where $\sigma(z) = (1+\exp(-z))^{-1}$. Maximum likelihood on the observed preferences therefore gives the loss used in our experiment:

$$\mathcal{L}_{\mathrm{DPO}}(\phi; \mathcal{D}_a) = -\frac{1}{|\mathcal{D}_a|} \sum_{(x,y^+,y^-)\in\mathcal{D}_a} \log \sigma(\beta\Delta_\phi)\,. \tag{11}$$

We compute each response log-probability by summing over response tokens while masking the prompt,

$$\log \pi_\phi(y \mid x) = \sum_{t=1}^{|y|} \log \pi_\phi(y_t \mid x, y_{<t}). \tag{12}$$

For one preference pair, differentiation gives

$$\nabla_\phi \ell = -\beta\, \sigma(-\beta\Delta_\phi)\left[\nabla_\phi \log \pi_\phi(y^+ \mid x) - \nabla_\phi \log \pi_\phi(y^- \mid x)\right]. \tag{13}$$

The frozen reference contributes no gradient. Thus a violated or low-margin preference receives the largest multiplier, and the update increases the chosen response's log-probability relative to the rejected response.

Only a fresh rank-16 LoRA adapter on the query, key, value, and output projections is updated, and the base and reference weights remain frozen. With microbatch size one and four-step gradient accumulation, the effective batch gradient is $g_k = \frac{1}{4}\sum_{i\in B_k} \nabla_\phi \ell_i$. AdamW then applies

$$\begin{aligned} m_k &= b_1 m_{k-1} + (1-b_1) g_k, & v_k &= b_2 v_{k-1} + (1-b_2)(g_k \odot g_k), \\ \widehat{m}_k &= \frac{m_k}{1-b_1^k}, & \widehat{v}_k &= \frac{v_k}{1-b_2^k}, \\ \phi_{k+1} &= (1-\alpha\lambda)\phi_k - \alpha \frac{\widehat{m}_k}{\sqrt{\widehat{v}_k}+\epsilon}. \end{aligned} \tag{14}$$

We use $\beta = 0.1$, $\alpha = 10^{-5}$, $(b_1, b_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, weight decay $\lambda = 0.01$, maximum sequence length 1,024, and one epoch. The LoRA coefficient is $\alpha_{\mathrm{LoRA}} = 32$ and dropout is 0.05. Every condition uses these same updates across three seeds.

To score the five policies, six independent reward models compare each one against the base policy on 400 fresh prompts (Liu et al., 2024a; Wang et al., 2024a). The two-order oracle is left out of this ranking because it is a diagnostic rather than a deployable filter. Its reward shifts are in Appendix M.1.

Table 19: Rank of each non-oracle training-data condition by win fraction against the base policy. Win fractions are averaged over three DPO seeds and 400 fresh held-out prompts before ranking. Ranks are recomputed over the five conditions shown, with 1 best. Mean rank averages the six evaluator ranks. PairPref is the pairwise preference evaluator.

	Evaluator-wise rank						
Training-data condition	Mistral	Skywork	FsfairX	ArmoRM	Eurus	PairPref	Mean rank
$P(\mathrm{U})$ **filter, adapted labels**	**1**	**1**	**1**	4	**1**	**1**	**1.50**
$P(\mathrm{U})$ filter, adapter-off labels	2	2	3	3	2	3	2.50
prompted self-report	3	3	2	2	3	2	2.50
hidden-state probe filter	4	4	4	**1**	4	4	3.50
unfiltered self-labeling	5	5	5	5	5	5	5.00

Every filtered dataset beats unfiltered self-labeling under every one of the six evaluators, and the $P(\mathrm{U})$ filter with adapted labels ranks first on five of the six and by mean rank.

M.1 Win rates and reward shifts

Wins, losses, and ties against the base policy. Table 19 ranks the conditions by how often each evaluator prefers the DPO model's response to the base policy's response on the 400 evaluation prompts. That ranking counts only strict wins. An evaluator can also score the two responses equally, which for a scalar reward model means a reward difference of exactly zero and for PairPref a preference probability of exactly 0.5. Table 20 therefore reports the win, loss and tie rates themselves, with the tie rate $T = 1 - W - L$, together with $W + \frac{1}{2}T$, the win rate when a tie counts as half a win.

Table 20: Win, loss and tie rates against the base policy. Percentages average three DPO seeds and six evaluators, each on the same 400 evaluation prompts.

Condition	Wins	Losses	Ties	Wins + half ties
$P(\mathrm{U})$ filter, adapted labels	45.92	43.28	10.81	51.32
$P(\mathrm{U})$ filter, adapter-off labels	44.96	42.74	12.31	51.11
Prompted self-report	44.74	41.33	13.93	51.70
Hidden-state probe filter	43.42	42.82	13.76	50.30
Unfiltered self-labeling	40.00	41.63	18.38	49.19
Two-order oracle (diagnostic)	44.11	42.64	13.25	50.74

The $P(\mathrm{U})$ filter with adapted labels wins more often than it loses once ties are accounted for, even though its strict win fraction sits below 50%.

All conditions in §M draw on the same 2,240 held-out prompts, and each contains exactly 6,000 pairs. They differ only in which pairs are kept and which model supplies the label. Table 21 gives each condition's mean reward minus the base policy's mean reward, in each scalar reward model's own units. The base policy is the reference throughout, so its own row would be zero in every column, just as its win rate against itself in Tables 19 and 20 would be 50%. The units span orders of magnitude, so do not average across columns. PairPref produces only a pairwise preference, so it appears in Table 19 but not here.

Table 21: Mean reward of each condition minus that of the base policy (3 DPO seeds, 400 fresh held-out prompts), in each evaluator's native units. The base policy is the zero point.

	Mean reward shift (native units)				
Condition	Mistral	Skywork	FsfairX	ArmoRM	Eurus
Unfiltered	$+0.001$	-0.029	-0.002	$+0.0001$	-21.4
$P(\mathrm{U})$ filter, adapter-off labels	$+0.035$	$+0.412$	$+0.003$	$+0.0000$	$+20.8$
prompted self-report	$+0.031$	$+0.247$	$+0.031$	$+0.0004$	-6.2
probe filter	$+0.015$	$+0.347$	$+0.019$	$+0.0008$	$+11.8$
$P(\mathrm{U})$ filter, adapted labels	$+0.053$	$+0.440$	$+0.024$	$+0.0003$	$+32.3$
Two-order oracle (diagnostic)	$+0.027$	$+0.340$	$+0.025$	$+0.0003$	-1.3