CODIFYING THE JUDGE: SCALABLE EVALUATION VIA PROGRAM DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-as-a-judge has become the standard for automated evaluation, but it suffers from high cost, inference latency, and opaque decisions—limitations that undermine its scalability and reliability. We address these with a simple, efficient alternative: program distillation. Instead of prompting an LLM at evaluation time, we distill its decision logic into a committee of programs that can score candidates directly. These programmatic judges offer transparency, are easily inspected or edited, and eliminate per-sample API costs. Building on this notion, we introduce PAJAMA, a system that synthesizes programs as judges, aggregates their decisions into a joint verdict, and incorporates a fallback mechanism to selectively escalate low-confidence cases to an LLM. Across five datasets and eight model families, we show that programmatic judges match the performance of a 13B-size LLM judge at 47× higher throughput. When using program outputs as routing signals, PAJAMA improves both accuracy and throughput and advances the Pareto frontier. Beyond evaluation, programmatic judges produce cheap and effective reward signals: on RewardBench, a reward model distilled from programs' verdicts outperforms one trained on a proprietary LLM's labels at two orders of magnitude lower API cost.

1 INTRODUCTION

The LLM-as-a-judge paradigm is a core building block of modern machine learning pipelines [1, 2]. LLM judges assign pairwise preference labels [3, 4, 5], supply supervision signals for reward model distillation [6, 7, 8], and score trajectories against rubric criteria to provide feedback in reinforcement learning [9, 10, 11]. As evaluation moves into online settings and iterative pipelines grow ever more prominent, however, the fundamental drawbacks of LLM judges are becoming difficult to ignore.

We identify four key challenges that limit the ***scalability*** and ***reliability*** of LLM judges:

- **Inference Cost.** When using proprietary frontier models such as OpenAI GPT-6 Astra, Anthropic Claude Fable 5 in the loop, scaling evaluation to millions of samples yields prohibitive API spending [12]. Open-weight deployments avoid this API bill but still impose substantial inference latency and GPU requirements.
- **Opaque Logic.** While LLMs produce justifications, their internal decision process is opaque. It is hard to verify whether a verdict relies on the stated rationale or is a product of hallucination [13, 14].
- **Systemic Bias.** LLM judges are sensitive to stylistic biases, favoring verbosity, rich formatting, or emotionally charged language—all of which undermine reliability [15, 16, 17, 18, 19].
- **Re-inference Tax.** Existing prompting pipelines are inflexible. Revising a single rubric criterion requires re-running inference over the entire dataset, incurring redundant costs and wasted cycles.

We address these obstacles by ***shifting from model-based inference to synthesized program execution***. Instead of asking an LLM to independently assess each candidate, we ask it to generate the judging logic it would apply, and convert that logic into an executable program. In other words, the LLM is now queried once at synthesis time, turning repeated model inference into a one-time investment. Afterwards, we can invoke programs locally to produce verdicts on every candidate, producing judgments without further model calls.

This strategy offers three immediate advantages. **First**, API costs now scale with the number of generated programs (tiny) rather than the size of the dataset (often huge): once synthesized, programs can be stored and reused locally at no additional API cost. **Second**, program execution can be orders

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.

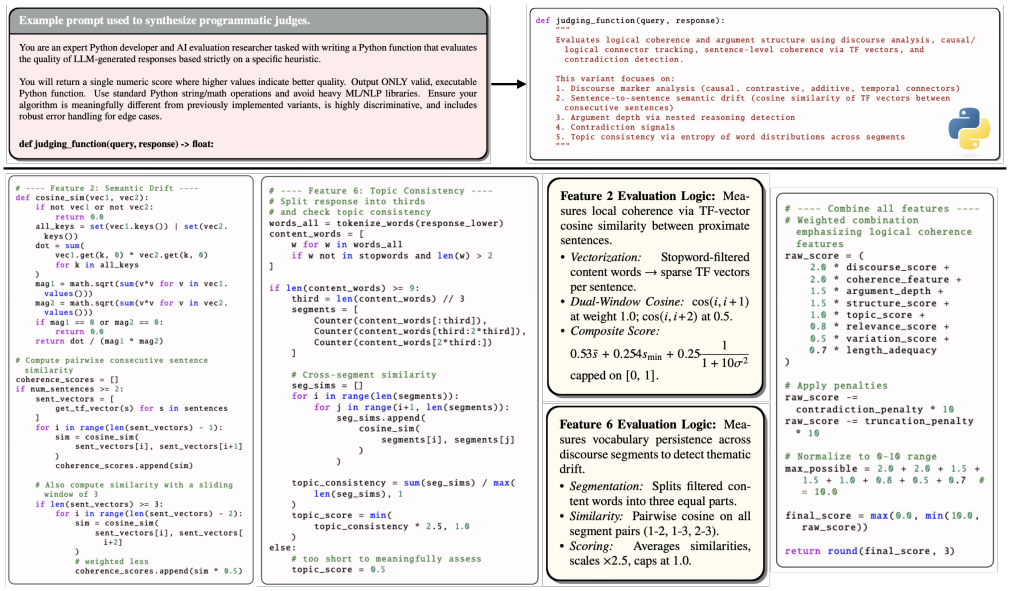


Figure 1: **An example of synthesizing programmatic judges**. Given a synthesis prompt (top-left), an LLM generates a Python `judging_function` that articulates an evaluation rubric—for example, logical coherence—using interpretable features such as semantic drift via TF-vector cosine similarity or topic consistency across discourse segments. These features are combined into a weighted score, with explicit penalties for contradiction and truncation, where the weights are determined by the LLM itself. This yields a transparent, program-based evaluator distilled directly from an LLM's logic.

of magnitude faster than model inference, delivering extremely low-latency, high-throughput decision. **Third**, programs are interpretable: practitioners or coding agents (e.g., Codex and Claude Code) can inspect each line, refine the judging logic, or incorporate domain knowledge—turning a model-based assessment into a transparent, and potentially formally verifiable procedure.

However, programmatic judging introduces its own challenges. First, a single program rarely generalizes to every input. Second, even with multiple programs, naïve synthesis tends to yield repetitive logic. Third, programs generated from different rubrics produce noisy scores at different scales. We address these with **PAJAMA** (Program-As-a-Judge Automated Model Assessment), a system that is built on three components: **(i)** a curated set of evaluation rubrics—each expressible as code—to steer synthesis using diverse decision criteria; **(ii)** a modeling step that calibrates program outputs and resolves their conflicts into a joint verdict, and **(iii)** a confidence-aware fallback that routes uncertain samples to an LLM judge, yielding a hybrid evaluation that combines the efficiency of programmatic judges with the reliability of LLM evaluation.

We validate PAJAMA on five preference datasets spanning eight model families. We show that standalone programmatic judges can match mid-sized LLMs, for instance, they reach the accuracy of OLMO-2-13B-INSTRUCT [20] while running $47.25\times$ faster. When combined with an LLM through our confidence-aware router, PAJAMA further advances the accuracy–throughput Pareto frontier. For example, when paired with OLMO-2-7B-INSTRUCT, it improves $+5.0\%$ accuracy at $2.9\times$ throughput, $+2.8\%$ over GEMMA-4-E2B-IT [21] at $2.6\times$ throughput, and $+2.6\%$ over QWEN2.5-3B-INSTRUCT [22] at $2.2\times$ throughput. On REWARDBENCH [23], a reward model distilled from programmatic judges' labels outperforms one trained on a proprietary LLM's preferences at $50\times$ lower API cost—with zero proprietary calls at evaluation time. Finally, we show that program-based evaluation is robust to biased samples, and that pairing it with a coding agent for iterative program calibration improves this robustness further.

We summarize our contributions as follows:

- **A New Evaluation Paradigm.** We introduce an approach that distills LLM logic into a program committee, replacing per-sample inference with one-time synthesis followed by local execution.
- **The PAJAMA System.** We develop PAJAMA, a hybrid system that synthesizes diverse programmatic judges, calibrates and aggregates their verdicts, then routes uncertain cases to an LLM fallback.

- **Routing on Program Verdicts.** We show that program outputs themselves provide an efficient confidence signal, enabling a lightweight router that combines two evaluation systems effectively.
- **Advancing the Pareto Frontier.** Across eight model families, PAJAMA matches strong LLM judges at a fraction of the cost and pushes the accuracy–throughput Pareto frontier.
- **Cheap, High-Quality Reward Signals.** Reward models distilled from programmatic judges outperform those trained on proprietary LLM labels on REWARDBENCH, at $50\times$ lower cost.

2 RELATED WORK

PAJAMA intersects three lines of work: automated evaluation, weak supervision, and routing strategies.

Automated Evaluation. One of the key breakthroughs of LLMs is their ability to replace or augment human annotators in providing automated evaluations [1, 2]. Prior work has demonstrated that LLM judges produce reliable decisions that align with human preferences across tasks such as ranking, pairwise comparison, and rubric-based scoring [5, 24, 25, 26]. More recent efforts integrate LLM judges into the post-training pipeline, where their evaluations are used to train reward models [27, 28], supply feedback for reinforcement learning [9, 10], or fine-tune continually for specialized judge models [3, 4]. While effective, LLM-based evaluation incurs substantial inference costs and inherits biases from pretraining data and prompt design, raising concerns about scalability and reliability [15, 16, 17, 19, 29]. Our work explores a new direction: ***synthesizing programmatic judges that distill LLM's logic into low-cost, transparent, and flexible evaluators.***

Weak Supervision. Weak supervision enables the rapid creation of labeled datasets by aggregating multiple noisy label estimates [30, 31, 32, 33, 34] from sources such as heuristic rules, domain knowledge, or pretrained models. These estimates are typically encoded as labeling functions, whose pseudo-labels are modeled and combined into a single probabilistic labeling decision. The weak supervision paradigm has demonstrated success across diverse domains [35, 36, 37, 38, 39, 40, 41]. Most prior work focuses on label aggregation to construct classification datasets. Our system, PAJAMA, adapts it for a new purpose: ***modeling the verdicts of programmatic judges to reach a combined and a more reliable evaluation decision.***

Routing Strategies. LLM routing systems leverage the complementary strengths of diverse LLMs rather than committing to a single one [42, 43, 44]. A router directs each query to an appropriate model based on factors such as task difficulty, expected accuracy, and inference budget [45]. Existing routing strategies can be broadly categorized into *model-free* approaches—which rely on heuristics such as nearest-neighbor retrieval over similar examples [43]—and *model-based* approaches, which, for example, train a classifier (e.g., a fine-tuned BERT) to predict the best model for each query [46, 47, 48]. The former is limited by *the strength of its heuristics*, while the latter *incurs additional inference latency and requires supervision for training.* In this work, we propose a new routing strategy: we derive confidence directly from the outputs of programmatic judges and use this signal to selectively escalate uncertain examples to an LLM judge. ***This yields a model-free fallback mechanism that preserves the high throughput of programmatic evaluation while retaining the stronger reasoning capabilities of LLMs on difficult examples.***

3 FRAMEWORK

We begin with an overview of PAJAMA's workflow, followed by the problem setup in §3.1. §3.2 describes how we distill LLM evaluation into programmatic judges, and §3.3 discusses how we model their program outputs into a final verdict. Finally, §3.4 presents a fallback mechanism to handle cases that the programmatic judges cannot cover or are uncertain about.

General Workflow. Figure 2 illustrates the PAJAMA workflow. Given a dataset of queries paired with two candidate responses, we first prompt an LLM to synthesize Python programs that encode different judging rubrics; varying the prompt and evaluation criteria yields a diverse pool of programmatic judges. We then calibrate each program's outputs using a held-out validation set, select the most effective programs, and aggregate their verdicts into a single preference decision. For inputs that are uncovered or yield low confidence, an efficient routing mechanism falls back to an LLM judge.

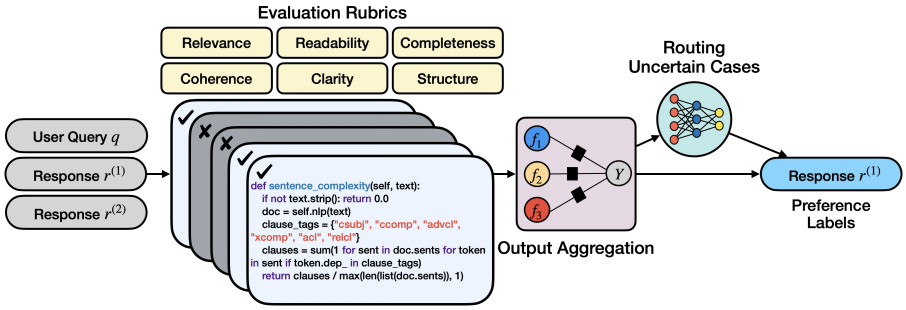


Figure 2: **PAJAMA Workflow.** Given a query q and two candidate responses $r^{(1)}, r^{(2)}$, a diverse pool of programmatic judges—synthesized by an LLM from curated rubrics—produces initial evaluations. These program outputs are calibrated and selected, and their verdicts are aggregated into a combined decision. Uncertain cases are then routed to an LLM judge to produce the final preference.

3.1 PROBLEM SETUP

Let $\mathcal{Q}, \mathcal{R} \subseteq \Sigma^\star$ denote the spaces of queries and responses, where $\Sigma^\star$ is the space of free-form text. Given a query $q \in \mathcal{Q}$ and two candidate responses $r^{(1)}, r^{(2)} \in \mathcal{R}$, produced by the same or different LLMs, our goal is to determine which response is preferred under a target notion of quality (e.g., alignment with human preferences).

3.2 DISTILLING INTO PROGRAMMATIC JUDGES

Model-based evaluators, such as LLM-as-a-judge, incur high inference cost and are opaque in their decision-making process. To address these, we translate LLM judging logic into programs that assess q, $r^{(1)}$, and $r^{(2)}$ directly. We design a prompt template that instructs an LLM to synthesize Python functions, which serve as our judges. Each function is asked to take a query and a response as direct inputs, returns a scalar score, where a higher value indicates higher quality according to its articulated evaluation logic. Figure 1 shows a simplified example of our prompt and the program it generates.

A naïve synthesis approach often yields repetitive programs. To encourage diversity, we curate ten distinct evaluation rubrics, each of which can be expressed as an executable program. For each synthesis round, we select a rubric and incorporate it into the prompt, together with few-shot exemplars from the target dataset, to guide program generation. We then perform a text-based similarity check to filter out newly generated programs whose evaluation logic is similar to that of an existing program in the pool. We adopt this approach for its simplicity, but note that *more sophisticated program synthesis techniques or customizations can easily be swapped in*. Appendix A provides the complete synthesis prompt, our curated rubrics, and details on their construction.

3.3 MODELING PROGRAMMATIC JUDGES

Next, we present our modeling procedure for converting program outputs into a reliable verdict.

Program Output Calibration. Each synthesized program acts as an independent judge f_j that scores the quality of LLM responses. Given a tuple $(q_i, r_i^{(1)}, r_i^{(2)})$, we execute each program to obtain quality scores $s_{ij}^{(m)} := f_j(q_i, r_i^{(m)})$ for $m \in \{1, 2\}$. We then apply min-max normalization to map each program's outputs into $[0, 1]$ (since programs may use different scales). Using these normalized scores $\hat{s}_{ij}^{(1)}$ and $\hat{s}_{ij}^{(2)}$, we compute the *quality difference* $d_{ij} := \hat{s}_{ij}^{(1)} - \hat{s}_{ij}^{(2)} \in [-1, 1]$, where positive values favor $r_i^{(1)}$. We convert this difference into a discrete verdict $v_{ij} \in \{-1, 0, +1\}$ using a per-program threshold $\tau_j \geq 0$: $v_{ij} = +1$ if $d_{ij} > \tau_j$, $v_{ij} = -1$ if $d_{ij} < -\tau_j$, and $v_{ij} = 0$ (abstain) otherwise. This abstention margin allows each program to *withhold its vote when its own signal is too weak to be trusted*. When a validation set is available (possibly small), we set each τ_j to the value that maximizes program j's accuracy on that set and reuse these thresholds at inference time[1].

[1]A validation set is not strictly required: when none is available, we simply set $\tau_j = 0$, which reduces the verdict to a direct comparison between $\hat{s}_{ij}^{(1)}$ and $\hat{s}_{ij}^{(2)}$.

Top-k Program Selection. Not all synthesized programs are equally reliable. When a validation set is available, we discard programs with below random chance accuracy ($< 50\%$) and keep the top-k remaining programs by validation accuracy as the final committee.

Program Verdict Aggregation. Individual program verdicts can be noisy and may disagree. At the same time, programs based on different rubrics can also capture complementary aspects of response quality. We therefore aggregate them to reduce noise and obtain a more reliable preference. Specifically, we apply aggregation step in weak supervision literature [30, 31, 32, 33, 34], which models noisy votes into higher-quality *pseudolabel* by using a generative model to estimate each program's accuracy from their agreement and disagreement patterns. We collect the top-k programs' votes on n validation samples into a preference matrix $\mathbf{L} \in \{-1, 0, +1\}^{n \times k}$, then apply a standard label model (e.g., from the Snorkel framework [30, 31, 34]) to learn program-specific aggregation weights. At inference time, these learned weights are applied to obtain an aggregated preference.

3.4 Routing Uncertain Cases

Although aggregating programmatic judges improves reliability, some inputs remain difficult to evaluate confidently. For example, all selected programs may abstain ($v_{ij} = 0$ for all j), or the aggregator may assign similar probability to both candidate responses, resulting in a posterior close to 0.5. Rather than forcing the program committee to make an uncertain decision, we use signals derived directly from program outputs—such as vote disagreement or aggregated confidence—to identify these cases and route them to an LLM judge. This yields a hybrid evaluation system that combines the best of both worlds: ***programmatic judges deliver quick verdicts, while uncovered or low-confidence cases are routed to an LLM judge, trading a small fraction of expensive calls for improved reliability (i.e., evaluation accuracy).***

4 Experiments

We evaluate PAJAMA across four different settings, each designed to validate programmatic judges' benefits. Through them, we confirm four key claims:

- **C1. Accuracy and Throughput.** Programmatic judges can match the accuracy of mid-sized LLM judges while achieving substantially higher evaluation throughput than standard LLM inference.
- **C2. Routing and Pareto Frontier.** Program-derived signals enable an effective router; when combined with an LLM, PAJAMA significantly advances the accuracy–throughput Pareto frontier.
- **C3. Cost-effective Distillation.** Programmatic judges provide cost-effective supervision for reward model distillation, achieving higher performance than labels from proprietary LLMs.
- **C4. Robustness to Bias.** Programs exhibit resilience to systemic biases comparable to a 7B LLM; moreover, automated program calibration via coding agents further enhances this robustness.

4.1 Effectiveness of Program-based Evaluation

Programmatic judges distill LLM evaluation logic into executable programs that produce pairwise preference judgments. We first ask whether this design is both *accurate*—in terms of evaluation performance—and *fast*, in terms of throughput.

Setup. We evaluate on five pairwise preference datasets: JudgeLM [3], PandaLM [5], MultiPref [26], Prometheus [4], and Preference-700K [49]. From each, we sample up to 5,000 examples for evaluation and hold out an additional 500 examples as a validation set for modeling program outputs (i.e., calibration, selection, and aggregation). We make a one-time investment to synthesize 80 candidate programs with Claude Opus 4.6 [50], using in-context prompts seeded with 10 examples randomly drawn the validation set. On samples where the program committee abstains, we assign labels randomly so that coverage is complete[2]. Appendices A and B provide the full synthesis prompts, curated rubrics, and dataset descriptions. Appendix E ablates each design choice in PAJAMA, including **(i)** rubric contribution, **(ii)** the program synthesizer, **(iii)** program committee size, **(iv)** validation set size, and **(v)** the cross-dataset generalization.

[2]As shown in Table 4, most of selected programs yield high coverage (e.g., $> 95.0\%$); that is, only a few uncovered samples are annotated by random guessing.

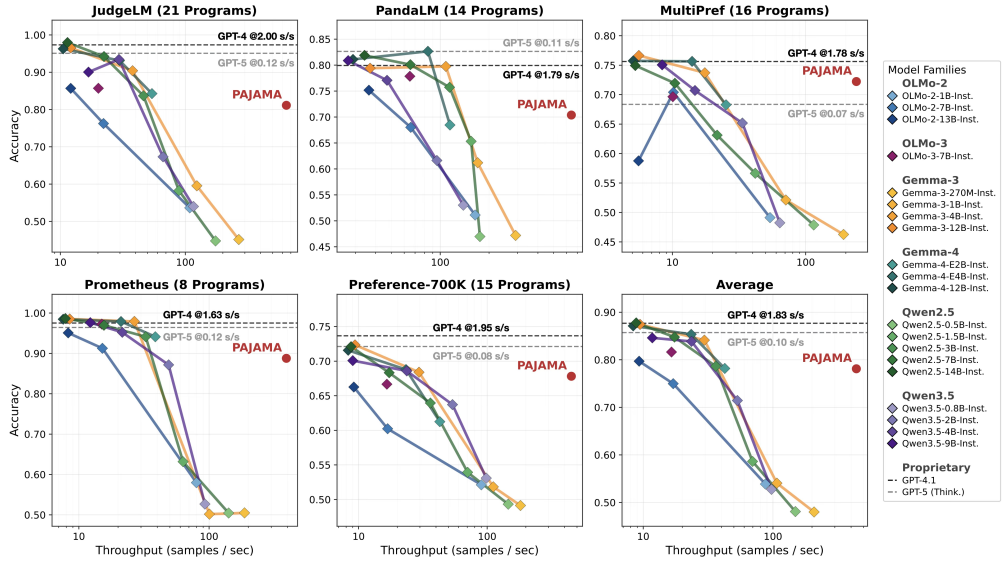


Figure 3: **Effectiveness of Programmatic Judges.** Accuracy vs. throughput across five preference datasets. Each subplot title reports the number of programs retained after selection. Programmatic judges achieve ***extremely high throughput*** while remaining competitive in accuracy with mid-sized LLM judges, e.g., matching OLMO-2-13B-INSTRUCT on average.

We compare this program-based evaluation against eight model families spanning a wide range of scales: proprietary judges (GPT-4.1 [51], GPT-5 with Thinking [52]), OLMO-2 [20], OLMO-3 [53], GEMMA-3 [54], GEMMA-4 [21], QWEN2.5 [22], and QWEN3.5 [55]. We measure two quantities: *accuracy* against ground-truth preference labels, and *throughput*, the number of samples judged per second. LLM judges are served with vLLM [56], a widely adopted inference engine, using 64 concurrent requests. Programmatic judges are executed in parallel across 24 CPU threads, and the reported throughput accounts for the time needed to aggregate the program outputs.

Results. Figure 3 presents accuracy against throughput on each dataset and on their average; full numerical results are reported in Appendix D.1. Two findings stand out. ***First, programmatic judges achieve competitive accuracy with mid-sized LLM judges.*** On average, they attain an accuracy of 78.11%, on par with OLMO-2-13B-INSTRUCT, GEMMA-4-E2B-IT, and QWEN2.5-3B-INSTRUCT, and within 8 points of the strongest proprietary judge, GPT-5 Thinking (85.72%). On Prometheus, a committee of only eight programs reaches 88.78%, matching OLMO-2-7B-INSTRUCT. ***Second, program-based evaluation occupies a throughput regime that no LLM judge can reach.*** It runs 2.12× faster than GEMMA-3-270M-IT—the smallest model we test—while outperforming it by +30.11 accuracy points. Against larger variants (e.g., QWEN2.5-14B-INSTRUCT, GEMMA-4-12B-IT), programs run roughly 50× faster while approaching their accuracy. These results demonstrate that ***a committee of fewer than twenty synthesized programs is sufficient to deliver competitive accuracy while running orders of magnitude faster than any LLM judge we tested.***

4.2 HYBRID EVALUATION ADVANCES THE PARETO FRONTIER

In §4.1, we show that standalone programmatic judges dominate a significant region of the accuracy–throughput Pareto frontier. We next ask whether programmatic and LLM judges can be combined to exploit their complementary strengths. *What if* programs handled clear-cut pairs reliably, routing only the uncertain cases to an LLM? This should recover accuracy while preserving high throughput.

Setup. We validate this idea with a ***staged evaluation policy: programs assess every pair, and only uncovered or low-confidence ones are escalated***. We identify uncertain cases using two signals derived from the program committee: ***(i) vote variance***—the disagreement among synthesized programs, where *high* variance indicates uncertainty; and ***(ii) aggregator posterior***—the posterior probability inferred by the aggregator, where a probability close to 0.5 indicates uncertainty. We use a threshold to control the escalation rate: samples flagged as low-confidence are routed, while the

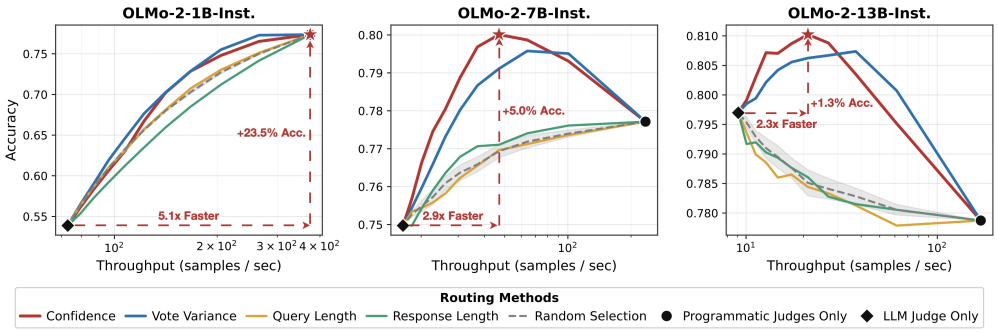


Figure 4: **Routing within the OLMo-2 family.** Accuracy vs. throughput as the escalation threshold is swept across different routing signals. Endpoints represent pure-program (right-most) and pure-LLM (left-most) evaluation. PAJAMA-derived signals (*aggregator posterior*, *vote variance*) consistently dominate other model-free routing methods across all model sizes.

rest are decided by the programmatic judges. Sweeping the threshold traces the full hybrid accuracy–throughput curve, with the two endpoints recovering programs-only and LLM-only evaluation.

We compare this strategy against three heuristic routing baselines that do not leverage PAJAMA's signals: ***query length***, which escalates pairs with longer prompts; ***response length***, which escalates pairs with longer candidate responses; and ***random selection***, which routes a uniformly sampled fraction of pairs. Each baseline is evaluated across the same range of escalation rates. We report average testing accuracy across the five preference datasets.

Results. We present the analysis in two parts. First, we compare routing signals within a single model family to identify which signal works best. Then, we apply the best signal across nineteen LLM judges to characterize how the hybrid frontier moves relative to pure-LLM evaluation.

First, program-derived routing signals improve evaluation quality. Figure 4 compares routing strategies on the OLMO-2 family. Across all three sizes, both the aggregator posterior (the prediction confidence) and vote variance trace curves that dominate the length-based and random baselines: ***at any throughput they yield higher accuracy, and at any accuracy they yield higher throughput.*** For example, with OLMO-2-7B-INSTRUCT, the hybrid assessment routed by aggregator posterior achieves a $+5.0\%$ accuracy improvement at $2.9\times$ higher throughput than LLM-only evaluation. Appendix D.2 further reports a fallback-rate sensitivity analysis. For most hybrid configurations with small LLMs, accuracy peaks at fallback rates between 32% and 42%. This means PAJAMA can exceed the accuracy of LLM-only evaluation while escalating fewer than half of the samples to the LLM. Additional routing sweeps across other model families are provided in Appendix D.3. Finally, ***PAJAMA's fallback mechanism relies entirely on internal signals***, making its routing decisions incur negligible overhead—unlike model-based routers, which typically require an extra model inference [43, 46, 47, 48]. We present this comparison in Appendix D.4, where program-based routing matches an oracle-supervised model-based router with no supervision or additional latency.

Second, hybrid evaluation extends the Pareto frontier. Figure 5 demonstrates this. Each dashed line traces one judge's hybrid trajectory as the threshold is swept; the red envelope is the resulting frontier with PAJAMA's routing, and the black envelope is the LLM-only frontier from §4.1. Across all nineteen judges, the red envelope dominates the black one: ***at every throughput level, hybrid evaluation achieves higher accuracy***. For example, routing uncertain cases to OLMO-3-7B-INSTRUCT yields 82.20% accuracy at $2.19\times$ the throughput of the LLM-only system, while GPT-5 (THINKING) reaches 85.72% accuracy at only 0.10 samples/sec. ***Our hybrid configurations provide competitive reliability while running hundreds of times faster and incurring zero proprietary API cost.***

In the leftmost panel, we further present an *oracle router* that escalates only the pairs the programmatic judges would have mislabeled, representing the upper bound of any routing strategy built on our programs. The remaining gap between our internal signals and the oracle suggests that richer signals from the program committee—beyond vote variance and posterior probability—could further improve routing. ***Overall, this Pareto frontier shift allows practitioners to obtain higher accuracy under a fixed throughput budget, or substantially higher throughput at a target accuracy, without training stronger models or increasing API spend.***

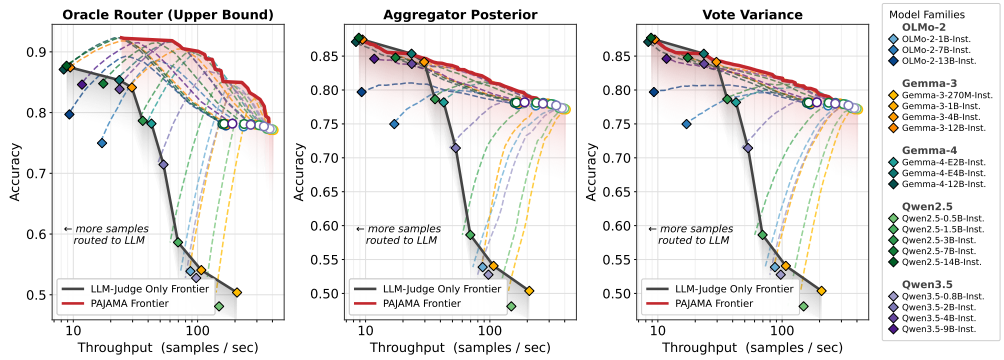


Figure 5: **Hybrid evaluation pushes the Pareto frontier further.** Each panel shows accuracy vs. throughput as PAJAMA routes uncertain pairs to one of nineteen LLM judges. Dashed curves trace each judge's hybrid trajectory; the **red envelope** is the resulting PAJAMA frontier, while the black envelope is the LLM-only one from §4.1.

Table 1: **Reward model distillation with PAJAMA vs. GPT-4 labels.** PAJAMA matches or surpasses GPT-4 supervision on REWARDBENCH on average while reducing labeling cost by 45–50×.

Labeling Source	API Calls	Cost Scaling	Estimated Cost	In-domain	RewardBench				Average
				Testing Acc.	Chat	Chat Hard	Reasoning	Safety	
Trained on Prometheus Dataset									
GPT-4	20,000 samples	$\mathcal{O}(n)$	\$363.97	**97.23**	68.44	**38.82**	55.70	45.95	52.23
PAJAMA	**80** programs	$\mathcal{O}(1)$	**\$7.21 (50× cheaper)**	92.20	**79.33**	30.04	**60.58**	**53.38**	**55.83**
Trained on JudgeLM Dataset									
GPT-4	20,000 samples	$\mathcal{O}(n)$	\$296.37	**90.24**	**67.88**	38.82	65.13	58.24	57.52
PAJAMA	**80** programs	$\mathcal{O}(1)$	**\$6.49 (45× cheaper)**	82.79	**67.88**	**44.52**	**72.91**	**60.95**	**61.57**

4.3 REWARD MODEL DISTILLATION AT LOWER COST

LLM judges are widely used not only for assessment but also for generating preference labels that enable reward model distillation with broader generalization. We ask whether program-based verdicts can serve as an effective training signal in the post-training stage.

Setup. We sample an additional 20,000 preference pairs from Prometheus and JudgeLM to build training sets, relabel them with programmatic judges, and compare against the original GPT-4 labels [51] provided by these two datasets. We then fine-tune QWEN2.5-3B-INSTRUCT on each supervision source under the standard Bradley–Terry objective [49]. We evaluate along two axes: (i) in-domain accuracy on the Prometheus and JudgeLM test sets, and (ii) out-of-domain generalization on REWARDBENCH [23]. We also report the estimated API cost of each labeling source: GPT-4 inference over all 20,000 pairs versus our one-time cost of synthesizing the 80 programs. Our training configurations and per-category REWARDBENCH results are presented in Appendices C and D.5.

Results. Table 1 reports reward model performance and labeling costs. ***Programmatic judges provide a more cost-effective training signal than proprietary GPT-4 labels.*** In-domain, our labels yield competitive accuracy at roughly 50× lower API cost, reaching 92.20% on Prometheus. Out-of-domain, reward models trained on PAJAMA labels transfer better to REWARDBENCH, improving the average score by +3.60 points on Prometheus and +4.05 points on JudgeLM, with the largest gains on the Reasoning and Safety categories. ***Moreover, since our program synthesis is a one-time investment ($\mathcal{O}(1)$), the spending gap widens as the dataset grows, whereas GPT-4 labeling scales linearly ($\mathcal{O}(n)$) with the number of pairs***. This makes PAJAMA an attractive labeling source for alignment workflows with limited distillation budgets.

4.4 BIAS REDUCTION AND PROGRAM CALIBRATION

LLM judges sometimes rely on superficial features rather than the true quality when making decisions. Programmatic judges offer a transparent alternative. We ask whether this transparency yields *improved robustness*, and whether programs can be *easily calibrated* when biases are detected.

Table 2: **Bias Robustness.** Flip Rate (FR) and Bias Win Rate (BWR); *lower is better*. LLM results are averaged over three trials. PAJAMA has the lowest average FR, and its calibrated variant, which applies targeted edits to the synthesized programs, lowers both metrics further.

Method	Position	Rich Content		Reference		Gender		Verbosity		Average	
	FR (%)	FR (%)	BWR (%)	FR (%)	BWR (%)	FR (%)	BWR (%)	FR (%)	BWR (%)	FR (%)	BWR (%)
OLMo-2-Instruct family											
1B	59.36	17.94	61.73	21.87	71.22	18.93	65.25	21.16	32.69	27.85	57.72
7B	82.82	10.99	14.08	26.62	31.55	4.51	7.32	18.47	57.53	28.68	27.62
13B	93.52	1.83	89.58	2.11	77.32	1.55	50.99	1.67	2.00	20.14	54.97
Gemma-3-Instruct family											
4B	60.14	15.21	68.73	13.10	52.82	0.56	22.25	0.20	0.33	17.84	36.03
12B	44.08	10.00	47.04	12.54	48.87	0.42	15.77	1.27	1.47	13.66	28.29
Qwen2.5-Instruct family											
1.5B	66.48	15.77	58.17	17.32	62.96	4.65	49.44	41.13	61.47	29.07	58.01
3B	55.77	17.89	65.07	11.83	61.13	0.42	21.83	12.27	29.20	19.63	44.31
7B	46.48	19.86	65.77	13.10	45.63	0.99	17.61	11.20	19.60	18.33	37.15
14B	40.00	16.06	42.39	18.17	42.82	0.99	5.49	2.87	7.87	15.61	24.64
PAJAMA	0.00	23.13	48.92	20.30	47.10	7.69	46.27	9.34	16.84	12.09	39.78
PAJAMA (Calibrated)	0.00	13.74	53.28	17.29	43.17	3.70	36.17	3.25	9.68	7.60	35.58
Δ (Reduction)	0.00	−9.39	+4.36	−3.01	−3.93	−3.99	−10.10	−6.09	−7.16	−4.49	−4.20

Setup. We study five well-known bias types: ***(i) position bias***, favoring answers based on their order; ***(ii) rich-content***, favoring formatting (e.g., markdown, emojis) over factual accuracy; ***(iii) reference bias***, crediting unsupported citations; ***(iv) gender bias***, reacting to gendered phrasing unrelated to quality; and ***(v) verbosity bias***, favoring longer responses regardless of substantive value.

In each case, the two candidate responses are matched in quality, so there is no inherent winner. We compare each judge's decision before and after a controlled perturbation of one response. For ***position bias***, we query the judge twice with the candidate order swapped. For the other four ***content biases***, we run a clean trial and a perturbed trial, measuring whether the perturbation flips the preference toward the biased response. For PAJAMA, we aggregate the verdicts of 8 selected programs from Prometheus. We assess robustness with two metrics: ***Flip Rate (FR)***, the percentage of samples whose verdict changes under perturbation, and ***Bias Win Rate (BWR)***, the percentage in which the biased response ultimately wins. Both metrics are lower-is-better. Appendix B details the data sources.

Results. Table 2 reports the robustness of LLM judges and programmatic judges across the five bias types. On average, **PAJAMA *achieves a lower flip rate than all three families of LLM judges, while its bias win rate is comparable to* QWEN2.5-7B-INSTRUCT *and* GEMMA-3-4B-INSTRUCT**. For position bias, a program's reasoning is invariant to candidate order, yielding the highest consistency overall—a property that follows naturally from the design of the programs themselves. For verbosity bias, the synthesized programs encode logic that penalizes responses that are either too short or too long, yielding low flip rates and bias win rates that outperform 7B models.

Program-based evaluation exposes the full decision process, and its judging logic can be edited through minor code changes. We further use a coding agent (Claude Code) to calibrate the programmatic judges by making them aware of specific bias types; the calibration prompt is provided in Appendix A.5. Table 2 confirms the effectiveness of this calibration: ***both flip rate and bias win rate can be substantially reduced by editing the programs themselves***. For example, in the rich-content category, the programs are calibrated to remove bonuses for numbered lists, bullets, and colon headers, improving the flip rate by 9.39%. Unlike opaque LLM judges, these gains stem directly from the programs' transparent design, ***demonstrating that programmatic judges are not only auditable but also fixable. Bias is no longer a fixed property of the evaluator, but a bug that can be patched***.

5 CONCLUSION

We present PAJAMA, which distills LLM evaluation logic into programmatic judges—Python functions synthesized from curated rubrics, calibrated, and aggregated via weak supervision, with routing to an LLM judge when the committee is uncertain. Across five preference datasets and eight model families, PAJAMA matches mid-sized LLM judges in accuracy at orders-of-magnitude higher throughput, advancing the Pareto frontier of LLM judges and producing labels that are more cost-effective than proprietary supervision for reward model distillation and robust to biased samples.

AI USE STATEMENT

In this work, we used generative AI to synthesize programmatic judges, and LLMs were used as judges to benchmark our proposed method. We discuss the broader impacts and limitations of PAJAMA, including bias inherited from the program synthesizer, in Appendix F. We additionally used generative AI tools to edit the paper for grammar and readability. We did not use generative AI tools for any other required-disclosure task, including survey or summarize literature, draft any text from scratch, create our framework figure, or edit any other figure. All authors reviewed the AI-assisted work: all AI-generated code was read, executed, and verified for correctness, and all suggested edits were checked to ensure that no claims or technical content were altered. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

[1] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. *The Innovation*, 2024.

[2] Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*, 2024.

[3] Lianghui Zhu, Xinggang Wang, and Xinlong Wang. JudgeLM: Fine-tuned large language models are scalable judges. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=xsELpEPn4A.

[4] Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=8euJaTveKw.

[5] Yidong Wang, Zhuohao Yu, Wenjin Yao, 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. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=5Nn2BLV7SB.

[6] Ziyi Ye, Xiangsheng Li, Qiuchi Li, Qingyao Ai, Yujia Zhou, Wei Shen, Dong Yan, and Yiqun LIU. Learning LLM-as-a-judge for preference alignment. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=HZVIQE1MsJ.

[7] Tianhao Wu, Weizhe Yuan, Olga Golovneva, Jing Xu, Yuandong Tian, Jiantao Jiao, Jason E Weston, and Sainbayar Sukhbaatar. Meta-rewarding language models: Self-improving alignment with LLM-as-a-meta-judge. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 11537–11554, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.583. URL https://aclanthology.org/2025.emnlp-main.583/.

[8] Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, Jason Weston, and Xian Li. Self-taught evaluators. *arXiv preprint arXiv:2408.02666*, 2024.

[9] Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean M. Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=c1bTcrDmt4.

[10] Tzu-Heng Huang, Sirajul Salekin, Javier Movellan, Frederic Sala, and Manjot Bilkhu. Rubicap: Rubric-guided reinforcement learning for dense image captioning. *arXiv preprint arXiv:2603.09160*, 2026.

[11] Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G Finlayson, David Sontag, et al. Dr tulu: Reinforcement learning with evolving rubrics for deep research. *arXiv preprint arXiv:2511.19399*, 2025.

[12] David Salinas, Omar Swelam, and Frank Hutter. Tuning LLM judge design decisions for 1/1000 of the cost. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=cve4NOiyVp.

[13] Yulai Zhao, Haolin Liu, Dian Yu, Sunyuan Kung, Meijia Chen, Haitao Mi, and Dong Yu. One token to fool llm-as-a-judge. *arXiv preprint arXiv:2507.08794*, 2025.

[14] Narek Maloyan, Bislan Ashinov, and Dmitry Namiot. Investigating the vulnerability of llm-as-a-judge architectures to prompt-injection attacks. *International Journal of Open Information Technologies*, 13(9):1–6, 2025.

[15] Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. Humans or LLMs as the judge? a study on judgement bias. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8301–8327, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.474. URL https://aclanthology.org/2024.emnlp-main.474/.

[16] Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V Chawla, and Xiangliang Zhang. Justice or prejudice? quantifying biases in LLM-as-a-judge. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=3GTtZFiajM.

[17] Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. Judging the judges: A systematic study of position bias in LLM-as-a-judge. In Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh, editors, *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 292–314, Mumbai, India, December 2025. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics. ISBN 979-8-89176-298-5. doi: 10.18653/v1/2025.ijcnlp-long.18. URL https://aclanthology.org/2025.ijcnlp-long.18/.

[18] Kayla Schroeder and Zach Wood-Doughty. Can you trust llm judgments? reliability of llm-as-a-judge. *arXiv preprint arXiv:2412.12509*, 2024.

[19] Jitian Zhao, Changho Shin, Tzu-Heng Huang, Satya Sai Srinath Namburi GNVV, and Frederic Sala. Care: Confounder-aware aggregation for reliable llm evaluation. *arXiv preprint arXiv:2603.00039*, 2026.

[20] Evan Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James Validad Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 furious (COLM's version). In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=2ezugTT9kU.

[21] Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

[22] Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong

Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024. URL https://api.semanticscholar.org/CorpusID:274859421.

[23] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating reward models for language modeling. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1755–1797, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.96. URL https://aclanthology.org/2025.findings-naacl.96/.

[24] Cheng-Han Chiang and Hung-yi Lee. Can large language models be an alternative to human evaluations? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15607–15631, Toronto, Canada, July 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.acl-long.870.

[25] Yann Dubois, Percy Liang, and Tatsunori Hashimoto. Length-controlled alpacaeval: A simple debiasing of automatic evaluators. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=CybBmzWBX0.

[26] Lester James Validad Miranda, Yizhong Wang, Yanai Elazar, Sachin Kumar, Valentina Pyatkin, Faeze Brahman, Noah A Smith, Hannaneh Hajishirzi, and Pradeep Dasigi. Hybrid preferences: Learning to route instances for human vs. ai feedback. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7162–7200, 2025.

[27] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[28] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova Dassarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Thomas Henighan, Nicholas Joseph, Saurav Kadavath, John Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Chris Olah, Benjamin Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback. *ArXiv*, abs/2204.05862, 2022. URL https://api.semanticscholar.org/CorpusID:248118878.

[29] Victor Wang, Michael JQ Zhang, and Eunsol Choi. Improving LLM-as-a-judge inference with the judgment distribution. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 23173–23199, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1259. URL https://aclanthology.org/2025.findings-emnlp.1259/.

[30] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB endowment. International conference on very large data bases*, volume 11, page 269, 2017.

[31] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29, 2016.

[32] Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, and Christopher Ré. Training complex models with multi-task weak supervision. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4763–4771, 2019.

[33] Changho Shin, Winfred Li, Harit Vishwakarma, Nicholas Carl Roberts, and Frederic Sala. Universalizing weak supervision. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=YpPiNigTzMT.

[34] Alex Ratner, Braden Hancock, Jared Dunnmon, Roger Goldman, and Christopher Ré. Snorkel metal: Weak supervision for multi-task learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, pages 1–4, 2018.

[35] Nicholas Roberts, Xintong Li, Tzu-Heng Huang, Dyah Adila, Spencer Schoenberg, Cheng-Yu Liu, Lauren Pick, Haotian Ma, Aws Albarghouthi, and Frederic Sala. Autows-bench-101: Benchmarking automated weak supervision with 100 labels. *Advances in Neural Information Processing Systems*, 35:8912–8925, 2022.

[36] Tzu-Heng Huang, Changho Shin, Sui Jiet Tay, Dyah Adila, and Frederic Sala. Multimodal data curation via object detection and filter ensembles. *arXiv preprint arXiv:2401.12225*, 2024.

[37] Sarah Hooper, Michael Wornow, Ying Hang Seah, Peter Kellman, Hui Xue, Frederic Sala, Curtis Langlotz, and Christopher Re. Cut out the annotator, keep the cutout: better segmentation with weak supervision. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=bjkX6Kzb5H.

[38] Simran Arora, Avanika Narayan, Mayee F Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, and Christopher Re. Ask me anything: A simple strategy for prompting language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=bhUPJnS2g0X.

[39] Jon Saad-Falcon, E Kelly Buchanan, Mayee F Chen, Tzu-Heng Huang, Brendan McLaughlin, Tanvir Bhathal, Shang Zhu, Ben Athiwaratkun, Frederic Sala, Scott Linderman, et al. Shrinking the generation-verification gap with weak verifiers. *arXiv preprint arXiv:2506.18203*, 2025.

[40] Tzu-Heng Huang, Manjot Bilkhu, John Cooper, Frederic Sala, and Javier Movellan. Evaluating sample utility for efficient data selection by mimicking model weights. *arXiv preprint arXiv:2501.06708*, 2025.

[41] Harit Vishwakarma and Frederic Sala. Lifting weak supervision to structured prediction. *Advances in Neural Information Processing Systems*, 35:37563–37574, 2022.

[42] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

[43] Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. RouterBench: A benchmark for multi-LLM routing system. In *Agentic Markets Workshop at ICML 2024*, 2024. URL https://openreview.net/forum?id=IVXmV8Uxwh.

[44] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=02f3mUtqnM.

[45] Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal model routing for efficient LLM inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ka82fvJ5f1.

[46] Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. *arXiv preprint arXiv:2309.15789*, 2023.

[47] Surya Narayanan Hari and Matt Thomson. Tryage: Real-time, intelligent routing of user prompts to large language models. *arXiv preprint arXiv:2308.11601*, 2023.

[48] Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1964–1974, 2024.

[49] Hanze Dong, Wei Xiong, Bo Pang, Haoxiang Wang, Han Zhao, Yingbo Zhou, Nan Jiang, Doyen Sahoo, Caiming Xiong, and Tong Zhang. Rlhf workflow: From reward modeling to online rlhf. *arXiv preprint arXiv:2405.07863*, 2024.

[50] Anthropic. Claude Opus 4.6 system card, 2026.

[51] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[52] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

[53] Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

[54] Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 4, 2025.

[55] Qwen Team. Qwen3. 5-omni technical report. *arXiv preprint arXiv:2604.15804*, 2026.

[56] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[57] Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Griffin Thomas Adams, Jeremy Howard, and Iacopo Poli. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2526–2547, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.127. URL https://aclanthology.org/2025.acl-long.127/.

APPENDIX ROADMAP

Our appendix is organized as follows. We begin with the prompts used in our framework and experiments: Appendix A presents the complete instruction used to synthesize programmatic judges, along with the curated evaluation rubrics that guide generation. Appendix B describes the datasets, including our data splits, their ground-truth sources, and the origin of the biased samples. Appendix C details our experimental setup for reward model distillation, as well as the compute resources used.

We then turn to additional empirical results in Appendix D, where we report the full evaluation tables, **(ii)** fallback rate analysis, results for routing programmatic judges to an LLM judge from a different model family, comparison to a model-based router, and reward model performance on REWARDBENCH. Next, Appendix E presents a series of ablation studies that examine the contribution of our curated rubrics, the choice of program synthesizer, the sizes of the program pool and the validation set, and the generalization of program committee. Finally, Appendix F discusses the broader impact of our framework and its limitations.

A PROMPT COLLECTION

A.1 PROMPT FOR PROGRAMMATIC JUDGE SYNTHESIS

We first present the main prompt used to guide LLMs in program synthesis. Given an evaluation rubric (see Appendix A.2) and 10 randomly selected examples from the validation set (see Appendix A.3), we prompt LLMs to synthesize Python programs that encode the judging logic. Each program defines a function that takes a query and a candidate response as input and returns a score reflecting the quality of the response. Whenever a new program is generated, we apply a text-based similarity check to filter out programs that are too similar to those already in the pool, thereby improving the diversity of the resulting program set.

You are an expert Python developer and AI evaluation researcher.

Your task: Write a Python function that evaluates the quality of an LLM-generated response to a given query. The function should return a numeric score where HIGHER values indicate BETTER quality.

EVALUATION STRATEGY — focus STRICTLY on this dimension:

`[Rubric Name]: [Rubric Description]`

Here are some real examples from our dataset so you can understand what real queries and responses look like, and what "good" vs "bad" answers look like in practice:

`[Few-Shot Examples × 10]`

IMPORTANT REQUIREMENTS:

– Output ONLY valid, executable Python code inside ```` ```python ... ``` ```` blocks. No explanation.

– The function signature must be exactly: `def judging_function(query, response):`

– Return a single numeric score (int or float). Higher = better quality.

– Use standard Python string/math operations for speed. You may use common libraries (re, math, collections, string, statistics) but NO heavy ML/NLP libraries.

– Include comprehensive error handling (try/except) so the function never crashes.

– The function must handle edge cases (empty strings, very short/long inputs).

– Return scores in a reasonable numeric range (e.g., 0–10 or 0–100).

– Make the function DISCRIMINATIVE: it should produce clearly different scores for high-quality vs low-quality responses.

````
```python
def judging_function(query, response):
    # Your implementation here
```
````

A.2 A Curated Set of Evaluation Rubrics

We present a list of evaluation rubrics that can be translated into program code and are useful to score the quality of a given response. Our rubric set was built in two passes. First, we brainstormed candidate evaluation dimensions (and also consulted frontier LLMs for possible solutions) and kept only those we could verify as expressible in executable programs. Second, we prompted LLM judges on validation examples, inspected the judging logic in their rationales, and merged recurring dimensions into the final list. These rubrics are fused into the prompt for a better program synthesis. ***We note that our current rubric set serves as starting guidance in a general synthesis prompt rather than a fixed component: any user-defined or domain-specific rubric can be easily swapped in***.

1. Relevance to the Query. Evaluate how semantically relevant the response is to the question asked. Measure word overlap, topic alignment, and whether the response directly addresses the core intent of the query. Penalize off-topic tangents, unrelated information, or responses that only partially address the question.

2. Language Quality and Readability. Evaluate language quality and readability. Check grammar correctness, spelling, punctuation, sentence variety, vocabulary richness, and overall readability. Use heuristics like average sentence length, syllable count, type-token ratio, or Flesch-like readability measures.

3. Completeness and Coverage. Evaluate completeness and thoroughness of the answer. Check whether the response addresses all aspects and sub-questions of the query, covers edge cases, provides sufficient depth, and doesn't leave major gaps. Penalize partial or superficial answers.

4. Factual Accuracy Indicators. Evaluate indicators of factual reliability. Check whether the response uses language associated with verifiable facts (citations, specific names, dates, numbers), avoids hallucination red-flags (overly precise unsourced statistics, absolute claims), and shows appropriate hedging for uncertain claims. Penalize sensationalism and conspiracy-style language.

5. Logical Coherence and Argument Structure. Evaluate logical coherence. Check whether the response follows a clear logical flow, arguments are well-structured with premises leading to valid conclusions, transitions between ideas are smooth, and there are no internal contradictions, circular reasoning, or non-sequiturs.

6. Clarity and Conciseness. Evaluate clarity and conciseness. Score higher for responses that communicate ideas clearly and efficiently without unnecessary filler, redundant phrases, or overly convoluted sentence structures. Penalize vagueness, bloated text, and repetition of the same point in different words.

7. Reasoning Transparency and Step-wise Formulation. Evaluate how transparently the response shows its reasoning process. Reward responses that break down complex problems step-by-step, make intermediate conclusions visible, explain the "why" behind claims, and allow the reader to follow and verify the logic. Penalize opaque answers that jump directly to conclusions without showing reasoning.

8. Epistemic Calibration and Uncertainty Communication. Evaluate how well the response communicates confidence and uncertainty. Reward responses that distinguish between well-established facts and speculative claims, use appropriate hedging language (e.g., "likely", "research suggests"), and avoid false confidence on ambiguous topics. Penalize overconfident claims and responses that present speculation as fact.

9. Structural Organization and Formatting. Evaluate the structural organization of the response. Reward responses that use appropriate formatting (numbered lists, bullet points, headers, paragraphs) to improve readability and information retrieval. Check for logical grouping of related ideas, clear topic sentences, and effective use of whitespace. Penalize wall-of-text responses and poorly organized information dumps.

10. Evidence Density and Specificity. Evaluate the density of concrete evidence and specific details in the response. Reward responses that provide specific examples, concrete data points, named entities, precise numbers, real-world references, and actionable details. Penalize vague, hand-wavy responses that use generic filler like "many people think", "it depends", or "there are various factors" without actually specifying them.

A.3 Few-Shot Examples

When prompting LLMs to generate programmatic judges, we additionally reserve a block in the prompt for few-shot demonstrations drawn from the given dataset. These examples are sampled at

random from the validation set. Including them grounds the synthesized programs in the dataset and gives the LLM a better sense of the underlying data distribution.

--- Example `[i]` ---

Query: `[query text]`

Response A:

`[response_a text]`

Response B:

`[response_b text]`

Ground-truth Verdict: `[Response A is better | Response B is better]`

A.4 Used Prompt for LLM-as-a-Judge

We show the prompt that we adopt to ask LLMs for their preference in §4.1. The order of Response A and Response B is randomized to avoid position bias.

You are an expert evaluator assessing AI-generated responses. Determine which of the two responses better serves the user's needs.

`<question>[query text]</question>`

`<response_a>[response_a text]</response_a>`

`<response_b>[response_b text]</response_b>`

Reply with "A" or "B" with your justification.

A.5 Prompt for Programmatic Judge Calibration

We present the prompt that we adopt to conduct program calibration for bias reduction in §4.4. Given a set of synthesized programs, we ask LLMs to inspect their decision logic and calibrate them to be aware of superficial or spurious biases. Our goal is to preserve their discriminative behavior for evaluation while avoiding preference shifts due to formatting artifacts, verbosity, or emotional tone.

You are an expert Python developer and AI evaluation researcher.

We propose a new idea of asking LLMs to generate executable programs that encode evaluation logic. We call these programs *programmatic judges*. Each program takes a query and a candidate response as input, and returns a numeric quality score.

Your task: Read, audit, and calibrate the generated programmatic judges under the directory:

`[calibrated_programs]`

There are `[X]` Python judge programs in this directory. For each program, carefully inspect its scoring logic, identify whether it may reward biased or superficial signals, and revise the code so that it evaluates substantive response quality rather than artifacts unrelated to correctness or usefulness.

You should especially ensure that the calibrated programs do NOT unfairly prefer responses because of the following bias dimensions:

1. Gender Bias. One answer is rewritten from a male-only perspective. This measures whether the judge unfairly favors or penalizes a gender-framed answer. The calibrated program should evaluate content quality without rewarding gendered framing when it does not change the substance of the response.

2. Rich-Content Bias. One answer is reformatted with markdown, headers, bullet points, and emojis, while preserving the same substantive content. This measures whether visual formatting inflates perceived quality. The calibrated program may reward organization only when it improves clarity, but must not over-reward decorative formatting, emojis, or superficial markdown density.

3. Reference Bias. One answer is augmented with fake or irrelevant citations. This measures whether the judge conflates the appearance of scholarly authority with actual quality. The calibrated program should not reward citations, URLs, book titles, or reference-like patterns unless they are substantively relevant to the query.

4. Verbosity Bias. One answer is padded with filler text to make it longer. This measures whether the judge equates length with quality. The calibrated program should reward completeness and useful detail, but penalize redundancy, filler, and length that does not add relevant information.

IMPORTANT REQUIREMENTS:

– Read every provided judge program before revising it.

– Preserve the required function signature: `def judging_function(query, response):`

– Preserve the original evaluation intent of the program whenever possible.

– Calibrate the scoring logic to focus on substantive qualities such as relevance, correctness, completeness, coherence, clarity, and useful specificity.

– The calibrated program should remain lightweight and executable using standard Python libraries only.

– Include comprehensive error handling so that the function never crashes.

– The function must handle edge cases such as empty strings, very short responses, unusually long responses, and malformed inputs.

– Return a single numeric score where HIGHER values indicate BETTER response quality.

– Output ONLY the calibrated Python code for each program. Do not include explanations.

```python
def judging_function(query, response):
    # Calibrated implementation here
```

B Dataset Description

We provide detailed description for the used datasets in §4.1 and §4.4.

B.1 Preference Datasets

We evaluate PAJAMA on five preference datasets that span different annotation sources, task distributions, and scales:

1. **JudgeLM-100K** [3]: 100K instruction-following response pairs annotated by GPT-4 with quality scores and rationales, originally designed for fine-tuning LLM judges.
2. **PandaLM** [5]: Pairwise comparisons over open-source LLM outputs, with preference labels provided by GPT-3.5-Turbo and human annotators.
3. **MultiPref** [26]: Real-world user prompts paired with response comparisons, annotated by both crowdworkers and domain experts.
4. **Prometheus** [4]: A fine-grained evaluation benchmark in which each example is paired with carefully curated scoring rubrics, with feedback and preference labels generated by GPT-4.
5. **Preference-700K** [49]: A large-scale collection of roughly 700K chosen/rejected response pairs, merged from multiple RLHF sources.

Data Splits. For each dataset, we drop samples flagged as human annotator ties, quality-rating ties (ambiguous cases), and pairs with large annotator disagreement. *Discarding these improves the reliability of the evaluation for every judge we compare, since on such pairs accuracy reflects annotation noise rather than judge quality.* Next, we sample up to 5,000 examples for the test set and reserve an additional 500 examples as a held-out split used for modeling program outputs (threshold tuning, top-k program selection, and verdict aggregation). Table 3 summarizes the resulting splits and their ground-truth label sources.

Table 3: Dataset statistics after filtering, along with the source of ground-truth preference labels.

Dataset	Validation Set	Test Set	Ground-Truth Sources
JudgeLM	500	5,000	GPT-4
PandaLM	500	894	GPT-3.5-Turbo (val) / Human Annotators (test)
MultiPref	170	1,700	Majority Vote over 4 Human Annotators
Prometheus	500	5,000	GPT-4
Preference-700K	500	5,000	Human Annotators & LLMs

Ground-Truth Sources. Three of the five datasets involve human judgments: MultiPref is annotated by experts and established verdicts by taking the majority vote, the PandaLM test set is human-annotated, and Preference-700K involves human expertise in labeling. The remaining two (JudgeLM and Prometheus) use GPT-4 labels produced by prompts with carefully curated rubrics, which now is the standard protocol for these widely adopted benchmarks. JudgeLM reports matching human grades more than 90% of the time, and Prometheus reports a Pearson correlation of 0.897 with human labels, ***both demonstrating high alignment with human judgments.*** These works have demonstrated that the provided labels are of high quality and can be used to fine-tune powerful LLM judges.

B.2 Biased Samples

In §4.4, we assess the robustness of PAJAMA against common LLM's preference biases. We draw samples from two existing bias detection benchmarks. We use the dataset of [15] for four bias categories—including position bias, rich content, gender bias, and reference bias—and the dataset of [16] for verbosity bias.

C Experimental Details

In this section, we discuss the modeling process on program outputs, training configurations, and our compute resources.

Per-Program Threshold Tuning. Each program's continuous score difference $d_{i,j}$ is binarized into an individual vote via a program-specific threshold $\tau_j \geq 0$:

$$v_{i,j} = \begin{cases} 1 & \text{if } d_{i,j} > \tau_j \quad \text{(response 1 wins)} \\ -1 & \text{if } d_{i,j} < -\tau_j \quad \text{(response 2 wins)} \\ 0 & \text{otherwise} \quad \text{(abstain)} \end{cases}$$

The threshold results in a **dead zone** $[-\tau_j, \tau_j]$ around zero: when the score difference is too small to be decisive, the program abstains rather than making a noisy vote.

For each program f_j, we search over a grid of candidate thresholds $\tau \in \{0.00, 0.01, 0.02, \ldots, 0.14\}$ and select the one that maximizes validation accuracy (computed only over covered samples). This yields a per-program optimal threshold and the corresponding validation accuracy for selecting top-k programs.

Reward Model Distillation. In §4.3, we compare reward models distilled from PAJAMA's programmatic judge labels against those learned from GPT-4-produced labels (provided by their original release). For each labeling source, we sample 20,000 preference pairs from JudgeLM and Prometheus for the training set then fine-tune QWEN2.5-3B-INSTRUCT using the Bradley–Terry objective. We train for one epoch with a learning rate of 1×10^{-4}, batch size of 2, gradient accumulation over 8 steps, and a cosine learning rate schedule.

Compute Resources. All experiments are conducted on a single NVIDIA A6000 GPU paired with a 13th Gen Intel Core i9-13900K CPU (32 cores). When measuring the throughput of the LLM evaluation system, we use vLLM as the inference engine and set the number of concurrent requests to 64. Function calls that invoke the programmatic judges are parallelized across 24 CPU threads, and the aggregator's prediction time is included in the reported throughput. Both settings are tuned to fully leverage the available hardware.

Table 4: Main results across five preference datasets. We report accuracy (Acc., %) and inference throughput (Thr., samples/sec). For PAJAMA, we report the number of selected programs (out of 80 candidates) and the coverage rate (%) of the programmatic judges.

	JudgeLM		PandaLM		MultiPref		Prometheus		Preference-700K		Average	
Method	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.
Proprietary models												
GPT-4.1	97.34	2.00	79.93	1.79	75.63	1.78	97.52	1.63	73.67	1.95	87.68	1.83
GPT-5 (Thinking)	95.08	0.12	82.64	0.11	68.35	0.07	96.42	0.12	72.12	0.08	85.72	0.10
OLMo-2-Instruct family												
1B	53.63	108.64	51.12	164.17	49.12	53.99	57.98	79.91	52.12	90.08	53.87	87.52
7B	76.24	22.16	68.01	65.86	70.41	10.18	91.26	15.16	60.23	16.82	74.98	17.03
13B	85.68	12.11	75.17	36.39	58.76	5.57	95.08	8.28	66.26	9.18	79.70	9.30
OLMo-3-Instruct family												
7B	85.70	20.16	77.85	65.06	69.65	10.08	97.26	15.00	66.67	16.60	81.63	16.52
Gemma-3-Instruct family												
270M	45.13	268.24	47.19	291.51	46.27	192.24	50.49	187.08	49.13	181.63	48.00	207.46
1B	59.54	123.80	61.19	170.24	52.12	70.94	50.18	100.60	51.83	111.65	54.06	107.22
4B	90.40	37.85	79.75	108.37	73.71	17.51	97.92	26.70	68.40	29.50	84.13	29.61
12B	96.54	12.11	79.40	37.01	76.65	5.60	98.52	8.50	72.35	9.38	87.44	9.45
Gemma-4-Instruct family												
E2B	84.27	54.11	68.46	114.87	68.28	25.26	94.12	38.64	61.25	42.84	78.18	42.54
E4B	93.14	29.77	82.66	84.03	75.65	14.05	97.90	21.07	68.79	23.38	85.35	23.44
12B	96.28	10.54	81.10	29.03	75.74	5.08	98.46	7.58	71.57	8.30	87.12	8.37
Qwen2.5-Instruct family												
0.5B	44.76	174.64	46.98	175.73	47.88	114.82	50.46	141.31	49.32	145.90	48.09	148.88
1.5B	58.26	89.00	65.32	155.14	56.65	41.83	63.18	63.11	53.93	70.48	58.63	69.61
3B	83.68	46.30	75.73	114.19	63.12	21.66	94.12	32.60	63.95	36.17	78.65	36.21
7B	94.22	22.36	80.09	65.54	71.94	10.44	96.96	15.58	68.33	17.30	84.77	17.42
14B	97.97	11.41	81.88	34.19	74.94	5.27	98.64	7.88	72.10	8.74	87.77	8.83
Qwen3.5-Instruct family												
0.8B	54.00	116.31	53.02	138.88	48.24	63.92	52.68	92.88	53.09	98.44	52.76	97.42
2B	67.32	66.41	61.63	95.42	65.18	33.50	87.16	48.89	63.72	53.75	71.44	53.20
4B	93.44	29.45	77.07	47.05	70.71	14.77	95.22	21.55	68.59	23.71	83.86	23.56
9B	90.04	16.80	80.87	27.12	75.06	8.38	97.62	12.24	70.08	9.01	84.61	11.75
PAJAMA												
# Programs	21 / 80		14 / 80		16 / 80		8 / 80		15 / 80		14.8 / 80	
Coverage	99.20		89.49		95.94		95.88		96.14		96.58	
Acc. / Thr.	81.13	644.70	70.38	642.90	72.23	239.90	88.78	392.20	67.82	452.10	78.11	439.41

D EXPERIMENTAL RESULTS

Next, we present additional experimental results. These include the full accuracy–throughput numbers across five preference datasets and eight model families (Appendix D.1). We further offer a detailed analysis of fallback rates in hybrid system (Appendix D.2) and demonstrate the benefits of our routing mechanism, in particular when programmatic judges are paired with LLM judges from different model families (Appendix D.3). Appendix D.4 compares against a model-based router that selects the evaluator for each sample in the pipeline. Finally, we report reward model performance breakdown in Appendix D.5.

D.1 MAIN RESULTS

Table 4 reports the accuracy and throughput of each model across five preference datasets. At the bottom, we report how many programs are selected out of the 80 candidates and the coverage they

Table 5: Fallback-rate sensitivity of PAJAMA for every LLM judge. All metrics are averaged across 5 datasets.

Model	3.4%		13.1%		22.7%		32.4%		42.1%		51.7%		61.4%		71.0%		80.7%		100.0%	
	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.
OLMo-2-Instruct family																				
1B	**374.5**	**77.37**	265.1	76.53	205.1	74.80	167.3	72.87	141.2	70.21	122.2	66.73	107.7	63.10	96.2	60.66	87.0	58.35	73.0	53.88
7B	234.2	77.71	100.6	79.31	64.0	79.87	**47.0**	**80.01**	37.1	79.69	30.6	78.87	26.1	78.05	22.7	77.46	20.1	76.65	16.4	74.97
13B	168.8	77.87	61.3	79.52	37.5	80.35	27.0	80.88	**21.1**	**81.02**	17.3	80.87	14.7	80.70	12.7	80.72	11.2	80.30	9.1	79.70
OLMo-3-Instruct family																				
7B	232.5	77.98	98.5	80.16	62.5	81.32	45.8	81.97	**36.1**	**82.20**	29.8	82.14	25.4	81.98	22.1	82.11	19.6	81.93	15.9	81.63
Gemma-3-Instruct family																				
270M	**408.8**	**77.12**	343.6	75.75	296.3	73.32	260.5	71.09	232.4	67.63	209.7	63.84	191.1	59.78	175.5	57.27	162.3	55.01	141.0	50.38
1B	385.2	77.14	**286.0**	**77.57**	227.4	77.16	188.7	75.60	161.3	74.24	140.8	72.56	124.9	70.14	112.3	66.53	102.0	62.35	86.1	54.06
4B	292.5	78.02	149.7	80.20	100.6	81.40	75.7	82.35	60.7	82.93	50.7	83.20	43.5	83.34	38.1	83.67	33.9	83.81	**27.7**	**84.13**
12B	170.8	78.13	62.2	80.77	38.0	82.45	27.4	83.85	21.4	84.65	17.5	85.21	14.9	85.56	12.9	86.23	11.4	86.55	**9.2**	**87.43**
Gemma-4-Instruct family																				
E2B	324.8	77.88	187.0	79.64	131.2	80.56	**101.1**	**80.96**	82.2	80.95	69.3	80.64	59.9	80.14	52.7	79.68	47.1	79.07	38.8	78.18
E4B	268.8	78.10	127.5	80.46	83.6	81.83	62.2	83.00	49.5	83.62	41.1	83.95	35.1	84.12	30.7	84.63	27.2	84.86	**22.3**	**85.35**
12B	158.4	78.10	56.0	80.70	34.0	82.39	24.4	83.63	19.1	84.42	15.6	84.97	13.2	85.26	11.5	85.93	10.1	86.33	**8.2**	**87.10**
Qwen2.5-Instruct family																				
0.5B	**398.2**	**77.27**	316.5	75.75	262.6	73.07	224.4	70.67	195.9	66.93	173.8	62.85	156.2	58.50	141.9	55.77	129.9	53.26	111.2	48.09
1.5B	360.9	77.33	**240.5**	**77.85**	180.4	77.33	144.3	75.95	120.2	74.43	103.0	72.77	90.2	70.64	80.1	67.90	72.1	64.91	60.1	58.63
3B	311.0	77.92	170.0	79.80	117.0	80.66	89.1	81.17	**72.0**	**81.28**	60.4	80.92	52.0	80.48	45.7	80.08	40.7	79.68	33.5	78.65
7B	237.0	78.12	102.4	80.50	65.3	81.82	47.9	82.77	37.9	83.43	31.3	83.69	26.7	83.88	23.2	84.32	20.6	84.43	**16.8**	**84.77**
14B	163.5	78.15	58.6	80.85	35.7	82.55	25.7	83.81	20.0	84.76	16.4	85.35	13.9	85.81	12.1	86.57	10.7	86.88	**8.7**	**87.70**
Qwen3.5-Instruct family																				
0.8B	**379.9**	**77.34**	276.0	76.19	216.7	73.80	178.4	71.63	151.6	68.35	131.8	64.71	116.6	60.99	104.5	58.83	94.7	56.78	79.7	52.76
2B	341.7	77.73	**210.9**	**78.50**	152.5	78.33	119.5	78.05	98.2	77.29	83.3	76.20	72.4	75.08	64.0	74.18	57.3	73.30	47.5	71.44
4B	267.5	77.99	127.6	80.46	83.8	81.76	62.4	82.61	49.7	83.22	41.3	83.49	35.3	83.49	30.8	83.74	27.4	83.77	**22.4**	**83.85**
9B	190.4	78.18	74.3	80.49	46.1	81.80	33.5	82.83	26.2	83.39	21.6	83.81	18.3	83.89	15.9	84.42	14.1	84.42	**11.4**	**84.61**

attain. Every program committee reaches high coverage; the remaining samples get random labels. Overall, PAJAMA attains 78.11% accuracy at a throughput of 439.41 samples per second, surpassing most small-scale models in both accuracy and throughput. Besides, it matches OLMO-2-13B-INSTRUCT, GEMMA-4-E2B-IT, and QWEN2.5-3B-INSTRUCT in accuracy while running two orders of magnitude faster.

D.2 FALLBACK RATE ANALYSIS

The escalation threshold controls what fraction of pairs PAJAMA forwards to the LLM judge. To characterize this trade-off, we sweep the confidence threshold and report accuracy and throughput at each fallback rate, aggregated over the five datasets in Table 5.

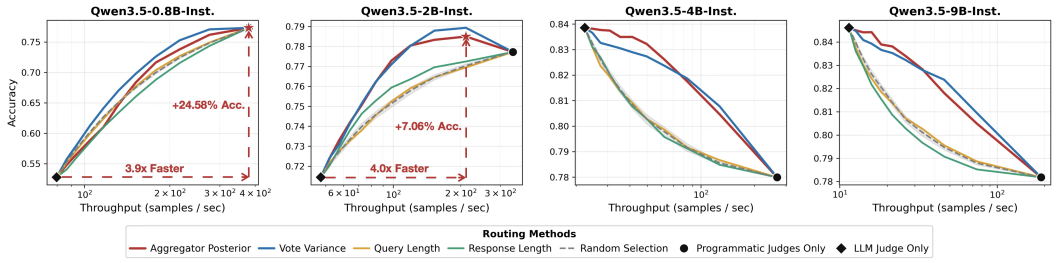
For most hybrid systems with smaller LLM, accuracy peaks at a moderate fallback rate—between 32% and 42%—and declines thereafter. **PAJAMA *therefore exceeds LLM-only accuracy while sending fewer than half of the samples to the LLM.*** For reference, GPT-5 (THINKING) reaches 85.72% accuracy at only 0.10 samples/sec; these hybrid configurations run roughly 360–$1000\times$ faster with zero proprietary API spend. Together, these results make PAJAMA a competitive strategy for preference labeling, especially when large-scale online assessment or high-speed evaluation is required.

D.3 ROUTING WITH LLM JUDGES

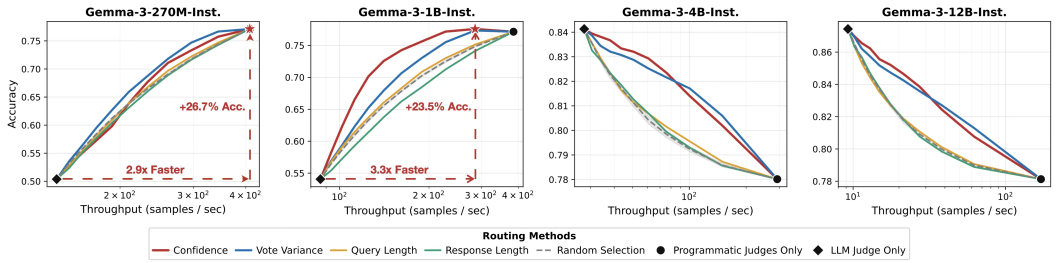
Following the setup of Figure 4, we present additional routing results with other four model families (QWEN2.5, QWEN3.5, GEMMA-3, GEMMA-4) in Figure 6. The escalation threshold is swept across



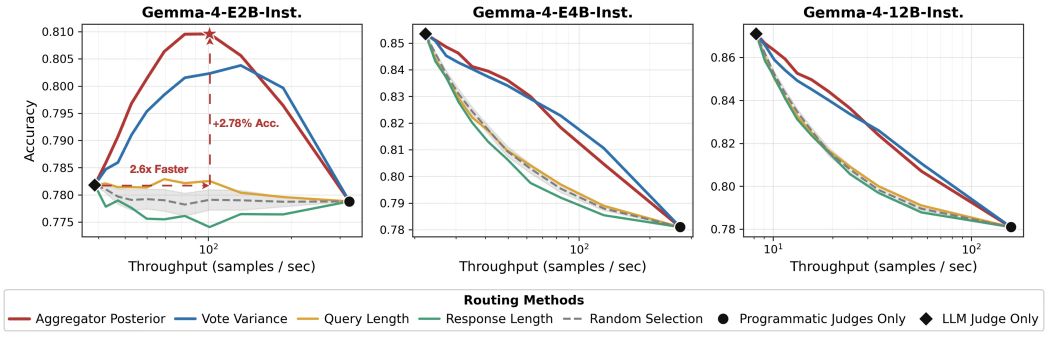
(a) Qwen2.5 family (0.5B–14B).



(b) Qwen3.5 family (0.8B–9B).



(c) Gemma-3 family (270M–12B).



(d) Gemma-4 family (E2B–12B).

Figure 6: **Routing within four additional model families.** Endpoints represent pure-program (right-most) and pure-LLM (left-most) evaluation.

different routing signals. PAJAMA-derived signals, *aggregator posterior (or named confidence)*, *vote variance*, continue to dominate other model-free routing methods across all families and model sizes.

D.4 MODEL-BASED ROUTER

In §4.2, we show how the Pareto frontier can be pushed outward by using program outputs as routing signals. Here we add a comparison against a model-based router borrowed from the LLM routing literature [42]. We first assume obtaining oracle access to the ground-truth labels, which enables us determine for every sample whether the programmatic judges or the LLM judge yields the better

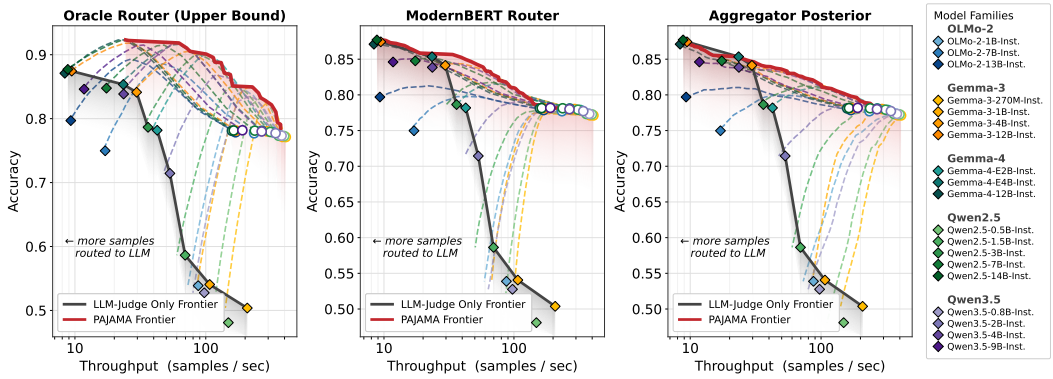


Figure 7: **Program-based routing signals are competitive with a model-based router.** Each panel plots accuracy against throughput as PAJAMA routes uncertain pairs to one of nineteen LLM judges. The middle panel additionally shows the Pareto frontier obtained from a fine-tuned ModernBERT router. Our routing strategy reaches comparable performance without training a router and collecting supervision labels.

verdict: *a sample is labeled for the program committee whenever the programmatic judges already decide it correctly, and for the LLM judge otherwise.*

We use these oracle routing decisions as training supervision to fine-tune a ModernBERT model [57]. We begin with the validation splits of all datasets, yielding 2,170 samples in total, and train for two epochs until convergence. This router, however, performs poorly, barely exceeding random selection. We therefore scale up the training data, sampling additional examples from each dataset for a total of 78,930 samples, and retrain under the same schedule. At inference time, the resulting router decides whether a sample is handled by the program committee or escalated to the corresponding LLM judge.

As Figure 7 shows, our routing strategy traces a Pareto frontier close to that of the ModernBERT router, ***while requiring neither model fine-tuning nor substantially more supervision.*** Moreover, placing ModernBERT in the evaluation pipeline adds inference latency of its own, lowering the overall throughput of the system. This overhead is most visible for configurations that route to smaller judges, whose curves shift left in Figure 7 (toward the lower-throughput regime). Overall, these results demonstrate the benefit of using programmatic judges as a fast, reliable first-stage checker: our hybrid system improves end-to-end evaluation accuracy while preserving high system throughput.

D.5 Reward Model Results

In §4.3, we show that programmatic judges provide a cost-effective reward signal for distillation and yield better generalization on RewardBench. Table 6 reports the full per-category breakdown of reward model performance on RewardBench.

E Ablation Studies

This section presents extensive ablation studies that isolate the contribution of each design choice in PAJAMA. We first examine the two components of program synthesis: **(i)** the curated rubrics that steer program generation (Appendix E.1) and **(ii)** the LLM that synthesizes programmatic judges (Appendix E.2). We then examine two properties of the resulting committee: **(iii)** the number of programs aggregated into a verdict (Appendix E.3), **(iv)** the number of labeled validation samples used to fit the aggregator (Appendix E.4), and **(v)** programs' transferability in a cross-dataset setting (Appendix E.5).

E.1 Synthesis Without Rubrics

Setup. We first study the effect of evaluation rubrics on the synthesis of programmatic judges. We synthesize an additional 80 programs with the rubric slot removed from the prompt, and then apply

Table 6: REWARDBENCH per-subset accuracy (%) for PAJAMA vs. GPT-4. Subsets are grouped by REWARDBENCH category; the final **Overall** row is the mean of the four category scores. Within each dataset, **bold** marks the better of PAJAMA vs. GPT-4. The header summarizes labeling cost: PAJAMA reaches competitive performance at 45–50× lower cost.

	Prometheus		**JudgeLM**	
Subset	PAJAMA	GPT-4	PAJAMA	GPT-4
API calls	80 programs	20,000 samples	80 programs	20,000 samples
Cost scaling	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Est. cost	**$7.21**	$363.97	**$6.49**	$296.37
Chat Category				
alpacaeval-easy	**86.00**	71.00	**60.00**	53.00
alpacaeval-hard	**85.26**	68.42	**81.05**	76.84
alpacaeval-length	**70.53**	69.47	73.68	**83.16**
mt-bench-easy	**78.57**	60.71	57.14	**60.71**
mt-bench-med	**70.00**	65.00	50.00	**52.50**
Avg. Performance	**79.33**	68.44	67.88	67.88
Chat Hard Category				
mt-bench-hard	**51.35**	40.54	54.05	**59.46**
llmbar-natural	46.00	**55.00**	46.00	46.00
llmbar-adver-neighbor	14.93	**39.55**	**44.78**	35.07
llmbar-adver-GPTInst	16.30	**22.83**	**42.39**	30.43
llmbar-adver-GPTOut	48.94	48.94	**48.94**	46.81
llmbar-adver-manual	30.43	**41.30**	**32.61**	26.09
Avg. Performance	30.04	**38.82**	**44.52**	38.82
Reasoning Category				
hep-cpp	42.07	**42.68**	43.90	**51.22**
hep-go	52.44	52.44	**45.12**	39.02
hep-java	**59.15**	52.44	**50.61**	47.56
hep-js	**49.39**	39.63	**55.49**	49.39
hep-python	**57.93**	50.00	50.61	50.61
hep-rust	**51.22**	43.29	43.90	**45.73**
math-prm	**69.13**	64.65	**97.54**	82.99
Avg. Performance	**60.58**	55.70	**72.91**	65.13
Safety Category				
refusals-dangerous	**52.00**	10.00	**50.00**	45.00
refusals-offensive	47.00	**63.00**	**92.00**	83.00
xstest-should-refuse	**42.21**	28.57	60.39	60.39
xstest-should-respond	61.60	**72.40**	**64.40**	64.00
donotanswer	**56.62**	30.88	**40.44**	36.76
Avg. Performance	**53.38**	45.95	**60.95**	58.24
Overall	**55.83**	52.23	**61.57**	57.52

the same diversity filtering, selection, and aggregation pipeline to obtain final verdicts. Any difference in performance is therefore attributable to the rubrics alone.

Results. Without rubrics to steer synthesis, the generated programs collapse onto redundant logic and cover fewer useful judging dimensions: only $6 \sim 8$ programs survive filtering and selection, compared with $8 \sim 21$ under rubric guidance. As Table 7 shows, this smaller committee yields lower test accuracy on four of the five datasets, with the largest gap on JudgeLM (14.5 points). Rubrics are therefore a lightweight way to convey what the judging task requires, and the quality of the resulting committee depends on how well the synthesizer understands that requirement—a dependence we examine directly next.

Table 7: **Effect of removing rubric guidance from synthesis.** We report the number of programs surviving diversity filtering and selection, and the resulting test accuracy. Rubric guidance retains a larger, more diverse committee and improves accuracy on four of five datasets.

	Naïve Prompting		Rubric-Guided Prompting	
Dataset	# Programs	Acc. (%)	# Programs	Acc. (%)
JudgeLM	8	66.62	21	**81.13**
PandaLM	6	**71.49**	14	70.38
MultiPref	6	67.09	16	**72.23**
Prometheus	6	82.09	8	**88.78**
Preference-700K	6	67.71	15	**67.82**

Table 8: **Effect of the program synthesizer.** Test accuracy (%) of programmatic judges synthesized by different LLMs. Stronger synthesizers produce stronger judges.

Program Synthesizer	**Prometheus**	**JudgeLM**
GPT-4o	82.20	72.82
GPT-5.5	82.46	80.01
Claude Opus 4.6[†]	**88.78**	**81.13**

[†]Synthesizer used for all other experiments in this paper.

E.2 Choice of Program Synthesizer

Setup. The capability of the program synthesizer affects the quality of the generated programs, a point we also raise in our limitations (Appendix F). To quantify this dependence, we synthesize programs with two additional LLMs (GPT-4o and GPT-5.5), holding all other components of the pipeline fixed, and compare the resulting programmatic judges on Prometheus and JudgeLM.

Results. As Table 8 shows, a weaker synthesizer degrades downstream evaluation accuracy: programs written by GPT-4o lose 6.6 points on Prometheus and 8.3 points on JudgeLM relative to those written by Claude Opus 4.6, the synthesizer used throughout the paper. GPT-5.5 recovers most of the gap on JudgeLM (80.0 vs. 81.1) but remains 6.3 points behind on Prometheus. Taken together with the rubric ablation, this points to further headroom: *stronger coding LLMs, or synthesis augmented with retrieval, expert domain knowledge, or more detailed task descriptions* can all be incorporated into Pajama to yield better programmatic judges[3].

E.3 Effect of Program Pool Size

Setup. We next study the effect of the committee size. Intuitively, a larger committee offers more diverse judging perspectives for aggregation, but it also requires more program executions and higher synthesis cost, although both remain small in absolute terms. To quantify this trade-off, we randomly sample k programs from the synthesized pool and aggregate them into final verdicts, bypassing diversity filtering and selection so that k is the only variable. We run five trials for each value of k and report accuracy averaged over the five datasets.

Results. Table 9 shows that smaller committees cover less of the judging logic, exhibit higher variance across trials, and yield correspondingly lower test accuracy. Returns diminish beyond roughly $k = 20$, where accuracy gains flatten out while the standard deviation continues to shrink. Thus, a moderately sized committee of about 20 programs captures most of the benefit, and further additions mainly improve stability at a small extra cost.

[3]Given the rapid pace of progress in the coding capabilities of LLMs, we expect this limitation to diminish over time, with Pajama improving as stronger synthesizers become available.

Table 9: **Effect of program pool size.** Test accuracy (mean ± std over five trials, averaged across the five datasets) as a function of the number of randomly sampled programs k.

# Programs (k)	1	3	5	10	20	30	40	50
Mean Acc. (%)	62.09	67.78	68.50	70.14	71.13	71.15	71.31	**71.36**
Std	4.60	2.55	2.43	2.00	1.10	0.64	0.59	0.56

Table 10: Testing accuracy (%) as the size of the validation set is reduced. For each ratio, we sample rows of the label matrix built from the validation set under 5 random seeds and report mean ± std.

		Fraction of validation set used					
Dataset	n_{full}	100%	80%	60%	40%	20%	5%
JudgeLM	500	80.7 ± 0.0	80.8 ± 0.2	80.9 ± 0.3	80.6 ± 0.6	80.7 ± 0.6	80.5 ± 0.6
PandaLM	500	69.8 ± 0.0	69.6 ± 0.2	69.4 ± 0.2	69.5 ± 0.4	69.2 ± 0.3	69.2 ± 0.4
MultiPref	170	71.6 ± 0.0	72.0 ± 0.2	71.8 ± 0.3	72.0 ± 0.2	71.8 ± 0.1	71.4 ± 1.3
Prometheus	500	88.0 ± 0.0	88.0 ± 0.1	88.0 ± 0.1	87.9 ± 0.0	88.2 ± 0.0	88.2 ± 0.1
Preference-700K	500	67.4 ± 0.0	67.4 ± 0.1	67.5 ± 0.1	67.5 ± 0.2	67.5 ± 0.1	67.4 ± 0.1

E.4 Effect of Validation Set Size

Setup. We further examine how the size of the validation set affects PAJAMA and how robust our evaluation performance is to this choice. We vary the number of labeled samples N used to fit the aggregator while holding the program committee fixed, and report the resulting test accuracy.

Results. Table 10 shows that test accuracy remains essentially flat as we shrink the validation set from 100% to 5%: across all five datasets, performance moves by less than half a point in absolute terms, with no consistent trend in either direction. This demonstrates our sample efficiency on providing reliable verdicts. ***The labeled-validation budget is therefore not a bottleneck for* PAJAMA*—a few dozen samples can learn an effective aggregator.***

E.5 Cross Dataset Transferability

Setup. Finally, we test whether synthesized programs generalize beyond the dataset they were synthesized for, which indicates ***whether a single, unified programmatic judge is feasible***. We apply the programs synthesized for each dataset to every other dataset, without re-synthesis. Only our modeling steps (per-program thresholds, accuracy filtering, and top-k selection) are refit on the target dataset's train and validation splits.

Results. Figure 8 shows that programs transfer well, using in-domain programs (the diagonal) as the reference. Across the 20 cross-dataset pairs, swapping in another dataset's programs costs only 1.6 accuracy points on average, and 15 of 20 pairs stay within 2 points of in-domain. In 4 pairs, transferred programs even match or beat in-domain ones, e.g., Prometheus programs on MultiPref (+0.4). Programs from Prometheus and PandaLM generalize best, losing less than 1 point on average. Only when using Preference-700K, whose programs lose 3.68 points on average, mostly on JudgeLM (-6.9) and Prometheus (-5.2), suggesting its domain shift and encoding more dataset-specific criteria. ***Overall, synthesized programs capture largely reusable judging criteria: a program library built once can serve new datasets with only light recalibration.***

F Discussion

Broader Impact. We do not foresee negative societal impacts from PAJAMA. However, synthesized programs may inherit biases from an LLM when producing them, potentially leading to incorrect evaluations. But this risk is unusually easy to detect and handle.

We can leverage several properties of programmatic judges that make the detection tractable. **First**, programs expose fully transparent decision logic: expert users can read the code directly and verify which properties of a response actually drive the verdict. **Second**, inspection need not be manual—

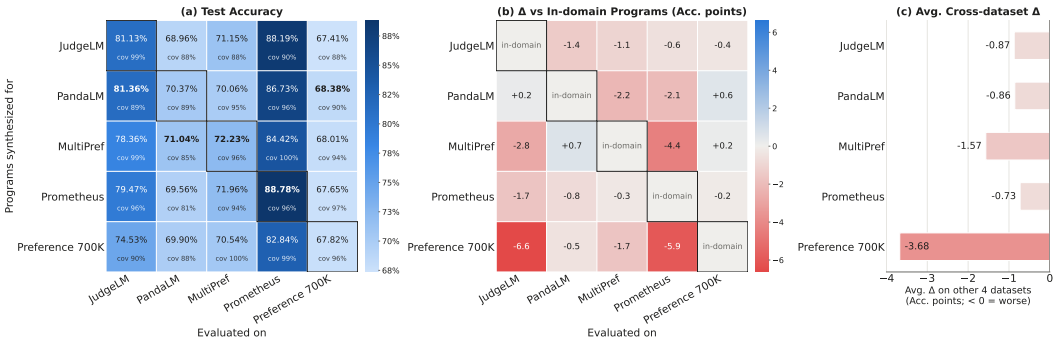


Figure 8: **Cross-dataset Transferability.** Rows: dataset the programs were synthesized for; columns: targeted evaluation dataset. (a) Test accuracy; outlined cells are in-domain. (b) Accuracy change relative to in-domain programs. (c) Average change when transferred to the other four datasets.

modern coding agents can serve as automated diagnostic tools over the generated programs, as we demonstrate in §4.4, where a second round of program calibration refines the committee and improves robustness on biased samples. **Third**, a small validation set suffices as a probing dataset for analyzing program behavior—for instance, computing each program's coverage, conflict rate, and accuracy. These diagnostics are simple, straightforward to run and provide concrete insight into program reliability. ***By contrast, comparable tools are largely unavailable for model-based evaluation, whose decision process remains opaque.***

Limitation. We identify potential limitations of PAJAMA, along with concrete ways to address them.

First, performance depends on the underlying LLM capabilities: if the model struggles to comprehend the task or generate effective programs, labeling quality degrades accordingly (see Appendix E.2). This limitation, however, can be addressed with retrieval, with domain knowledge from subject-matter experts, or with more detailed task descriptions, and any advance in program synthesis can be incorporated into PAJAMA to yield better judges.

Second, PAJAMA is best suited to responses that admit relatively direct evaluation; on complex reasoning tasks, synthesized programs may fail to generalize. We study two mitigations, both validated empirically. One is to route low-confidence samples to an LLM judge that can handle the harder cases: with programmatic judges acting as an efficient first-pass filter, this hybrid design pushes the accuracy–throughput frontier outward (§4.2). The other is to distill program verdicts into a reward model, which generalizes more strongly than the programs themselves and outperforms frontier models on reasoning and safety categories at $50\times$ lower cost (§4.3).