Shrinking the Generation-Verification Gap with Weak Verifiers

Jon Saad-Falcon [*1] **E. Kelly Buchanan** [*1] **Mayee F. Chen** [*1] **Tzu-Heng Huang** [2] **Brendan McLaughlin** [1]
Tanvir Bhathal [1] **Shang Zhu** [3] **Ben Athiwaratkun** [3] **Frederic Sala** [2] **Scott Linderman** [1] **Azalia Mirhoseini** [1]
Christopher Ré [1]

Abstract

Verifiers can improve language model (LM) capabilities by scoring and ranking responses from a pool of generated candidates. Currently, high-quality verifiers are either unscalable (e.g., humans) or limited in utility (e.g., tools like Lean for formal proofs). While LM judges and reward models have become broadly useful as general-purpose verifiers, a significant performance gap remains between them and oracle verifiers (i.e. verifiers with perfect accuracy). To help close this gap, we introduce WEAVER, a framework for designing a strong verifier by combining multiple weak, imperfect verifiers. First we find that weighted ensembles of verifiers, which typically require learning from labeled data, significantly outperform unweighted combinations due to differences in verifier accuracies. To reduce the dependency on labeled data, WEAVER leverages weak supervision to estimate each verifier's accuracy and combines their outputs into a unified score that better reflects true response quality. However, directly applying weak supervision algorithms poses several challenges, including inconsistent verifier output formats and handling low-quality verifiers. WEAVER addresses these challenges by using dataset statistics to normalize outputs and filter specific verifiers. We study the effectiveness of WEAVER in test-time repeated sampling settings, where a model generates multiple candidate responses and selects one from among them. Our evaluations demonstrate that WEAVER significantly improves over $Pass@1$—the performance when simply selecting the first candidate response—across several reasoning and math tasks, achieving o3-mini-level accuracy with Llama 3.3 70B Instruct (a much cheaper non-reasoning model) as the generator, and an ensemble of 70B or smaller judge and reward models as the verifiers (87.7% average). This gain mirrors the jump achieved between GPT-4o and o3-mini (69.0% vs. 86.7%), which required extensive fine-tuning and post-training interventions. To reduce the computational costs of running verifier ensembles for WEAVER, we train a compact 400M cross-encoder using WEAVER's combined output scores. This distilled model retains 98.7% of WEAVER's full accuracy while reducing verification compute by up to 99.97%.

1. Introduction

A core challenge in deploying language models (LMs) is *verification*: determining the quality or correctness of a model's response. This problem arises across various components of the LM pipeline, including dataset curation, model alignment, and inference-time decision-making. Verification relies on *verifiers*—functions that score responses. When combined with repeated sampling—generating multiple candidate responses from a LM—a perfect verifier can be used to select a correct candidate response, significantly enhancing model capability on tasks such as math, code, and reasoning (Snell et al., 2024; Brown et al., 2024; Puri et al., 2025). For example, Llama 3.1 8B Instruct can match Llama 3.1 70B Instruct and even GPT-4o performances on MATH500 (Hendrycks et al., 2021) and MiniF2F (Zheng et al., 2022) when paired with perfect verifiers for these mathematics tasks. However, without a perfect verifier, a *generation-verification gap* emerges (Song et al., 2025b): a LM can generate a correct response, but we fail to identify it.

The generation-verification gap is prevalent across many tasks across mathematics, coding, scientific reasoning, instruction-following, and more. For some of these set-

[*]Equal contribution [1]Stanford University, Stanford, CA, USA [2]University of Wisconsin-Madison, Madison, WI, USA [3]Together AI, San Francisco, CA, USA. Correspondence to: Jon Saad-Falcon <jonsaadfalcon@stanford.edu>, E. Kelly Buchanan <kelly.buchanan@stanford.edu>, Mayee F. Chen <mfchen@stanford.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

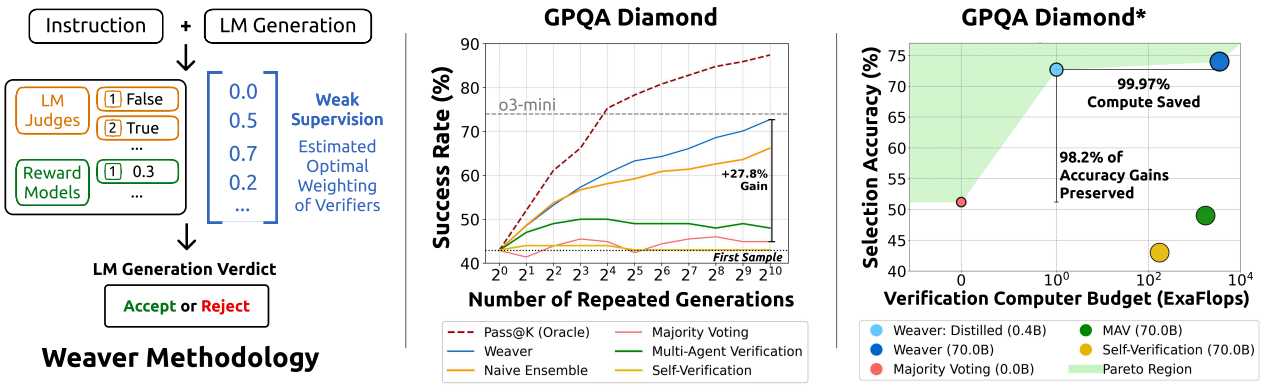

Figure 1: **WEAVER Framework**: We propose WEAVER, a framework combining multiple weak verifiers to effectively scale repeated sampling without parameter finetuning on ground truth labels **(left)**. WEAVER significantly outperforms majority voting and shrinks a model's *generation-verification gap* by 14.5%, on average, for GPQA Diamond and other datasets (Table 2) **(middle)**. By distilling WEAVER from an ensemble of 70B verifiers to a single 400M cross-encoder, we can preserve 98.2% of the accuracy gains of WEAVER while reducing inference compute cost by 99.97% **(right)**.

tings, we have access to *oracle verifiers* that can perfectly identify correct responses. A prominent example is Lean, a formal theorem prover that can be used for problems such MiniF2F (Zheng et al., 2022). However, this is often a limited setup, as not all mathematical proofs can be processed by Lean. Alternatively, humans could judge LM responses but manual evaluation is often expensive, noisy, and difficult to scale (Hosking et al., 2024; Clark et al., 2021; Karpinska et al., 2021). In contrast, LMs prompted as judges (Chiang et al., 2024) and reward models (Lambert et al., 2024; Singhi et al., 2025a; Liu et al., 2025a) can be applied off-the-shelf to tasks like mathematics, coding, scientific reasoning, instruction-following (Hendrycks et al., 2021; Rein et al., 2024; Jain et al., 2024; Li et al., 2023). However, these *weak verifiers* produce noisy, inconsistent scores, often exhibit poor calibration, and suffer from high false positive rates (Stroebl et al., 2024). We ask: *to what extent can we leverage weak verifiers to improve accuracy in the repeated sampling regime?*

We explore *scaling verification*, specifically how to combine *multiple weak verifiers* to improve response selection for repeated sampling. As new pre-trained models become available, the pool of weak verifiers continues to expand and offer diverse, complementary sources of signal that could improve response selection if they can be aggregated effectively. Recent work has explored scaling verification through techniques such as self-verification or averaging LM judge scores (Lifshitz et al., 2025; Zhao et al., 2025; Chen et al., 2025) although other work has found limitations to scaling test-time compute when utilizing weak verifiers for response selection (Stroebl et al., 2024). We observe three key challenges towards ensembling weak verifiers:

1. **Naively aggregating weak verifiers is insufficient for reliable verification.** Weak verifiers such as LM-based judges or reward models produce noisy, biased, and poorly calibrated scores, leading to inconsistent performance. (Stroebl et al., 2024; Lambert et al., 2024; Chiang et al., 2024). While using a naive unweighted average of verifier scores is straightforward, it implicitly assumes uniform verifier quality, causing low-quality verifiers to dominate and degrade the overall accuracy (Verga et al., 2024; Xu et al., 2024; Eisenstein et al., 2023b). Moreover, while previous work has hypothesized that more sophisticated weighted ensembles should perform better, this claim has not been studied (Lifshitz et al., 2025).

2. **Effective ensembling with limited labeled data is challenging.** More sophisticated ensembling techniques typically learn verifier weights from labeled data, but such data is expensive and difficult to obtain. *Weak Supervision* (WS), a family of statistical techniques developed for data labeling, offers a potential solution through algorithms that aggregate multiple weak signals—such as crowd-worker annotations and expert-defined heuristics—while only requiring a small amount of labeled data (Ratner et al., 2016; 2019; Fu et al., 2020). In traditional WS, practitioners can design and shape each weak signal to ensure sufficient quality (i.e., iteratively tweaking program-based heuristics), and guarantees of WS hinge on a baseline level of quality. Our weak signals, however, are fixed pre-trained language model verifiers, which have wildly varying accuracy—especially when applied to out-of-distribution tasks—and can emit incompatible outputs (logits, binary scores, Likert scores) (Lambert et al., 2024) that we cannot easily tweak. Due to these conditions, WS algorithms may not perform well when

directly applied to verification.

3. **Verification is expensive to deploy at inference.** Verification can dominate inference-time costs (Singhi et al., 2025a; Liu et al., 2025a), since each verifier must process both the problem and its candidate response(s) (Lightman et al., 2023), often evaluating intermediate steps (Lightman et al., 2023) and multiple solution paths (Snell et al., 2024). In fact, achieving gains over unverified generation (i.e. majority voting) can require $10\times$ to $128\times$ the inference compute per query (Singhi et al., 2025b; Lifshitz et al., 2025; Zhao et al., 2025; Chen et al., 2025).

In this work, we introduce WEAVER, a framework for aggregating weak verifiers without supervised finetuning on ground truth labels (Figure 1). First, we demonstrate that if we have access to a large corpus of labeled training data (e.g., 50,000 query-response pairs), we can learn weighted ensembles that can outperform naive averaging by up to 11.2% points. This is because weighted ensembles take advantage of wide variability in verifier accuracy. However, in many real-world scenarios, we do not have access to such quantities of labeled data. Second, to reduce the dependency on labeled data, we adapt Weak Supervision to the verification setting by addressing challenges around inconsistent outputs and low-accuracy verifiers. WEAVER filters out uninformative verifiers, normalizes verifier scores, and builds a latent variable model over these scores and the unknown true labels to estimate the verifier accuracies to be used as weights for the ensemble (Ratner et al., 2016; Hall, 2003).

Empirically, given a repeated sampling budget and a set of verifiers, WEAVER improves over repeated sampling with unweighted averaging of verifier scores by 17.1% and with majority voting by 13.5% (Table 2; Figure 3). Compared to an LM's $Pass@1$, WEAVER allows us to improve performance by 17.9% for 8B models and 14.5% for 70B models across reasoning and mathematics tasks (Tables 2 and 20). *This mirrors the performance jump from GPT-4o to o3-mini* (73.9% vs. 88.2%)—but only via increased sampling at test time rather than parameter tuning or post-training procedures. We also study how WEAVER scales along different axes of test-time compute: generation, verifiers, model size, and inference budget (Section 5.2). We find that even as we increase the number of generations, many standard verification baselines (e.g. majority voting) quickly plateau (Figure 3). Naive ensembling saturates more slowly, but its gains are limited by sensitivity to the model choice and the number of verifiers.

Finally, to mitigate the compute costs of calling multiple weak verifiers for each response, we extend WEAVER by training a 400M-parameter cross-encoder verifier using WEAVER's selected responses. We demonstrate that using a distilled WEAVER cross-encoder as a verifier *retains 98.7% of the accuracy gains* from the learned verifier ensemble while reducing compute costs by three orders of magnitude – *saving 99.97% inference FLOPS* while still capturing an effective verification strategy (Section 6). Overall, our findings highlight that more reliable, scalable verification is possible even in the absence of ground-truth labels—paving the way for improved data filtering, model alignment, and inference-time decision-making.

2. Related Work

LM Judges and Reward Models: Both LM judges and reward models are promising approaches for evaluating language model outputs, but their high false positive rates limit their reliability (Stroebl et al., 2024). LM judges can evaluate outputs without additional training (Liu et al., 2023; Wang et al., 2023a; Fu et al., 2023), using approaches from simple prompting to chain-of-thought reasoning (Liu et al., 2023) to specialized fine-tuning (Saad-Falcon et al., 2023; Tang et al., 2024) to multi-LM inference architectures (Saad-Falcon et al., 2024a; Kalra & Tang, 2025). However, they face poor generalization across contexts (Es et al., 2023; Saad-Falcon et al., 2023; Ravi et al., 2024) and systematic biases in position and self-preference (Chen et al., 2024a; Pan et al., 2024; Zheng et al., 2023b; Koo et al., 2023). Similarly, while reward models have become central to model alignment (Bradley & Terry, 1952; Christiano et al., 2017; Liu & Zeng, 2024), they struggle with noisy training signals from low inter-annotator agreement (Askell et al., 2021; Ouyang et al., 2022; Wang et al., 2024a; Dubois et al., 2024b) and learned biases favoring attributes like response length (Lambert & Calandra, 2023; Singhal et al., 2023; Dubois et al., 2024a). Recent work has improved individual verifier reliability through better data collection, chain-of-thought reasoning, and natural language unit tests (Wang et al., 2023b; Zhang et al., 2024; Saad-Falcon et al., 2024b), yet fundamental challenges persist (Eisenstein et al., 2023a; Chaudhari et al., 2024). WEAVER advances beyond these approaches by combining multiple verification signals with adaptive weighting, thus leveraging the complementary strengths of weak verifiers while suppressing noise and reducing false positives.

Weak Supervision: WEAVER builds upon statistical techniques from weak supervision, which emerged as a framework for programmatically generating training labels by aggregating multiple weak sources (Ratner et al., 2016; 2020). While a majority of the work focuses on classification tasks (Ratner et al., 2019; Fu et al., 2020; Chen et al., 2022), recent advances have expanded to handle multi-task settings (Shin et al., 2021) and structured prediction (Vishwakarma & Sala, 2022). Weak Supervision has also been applied to LM prompting (Arora et al., 2022) and routing (Guha et al., 2024). WEAVER applies Weak Supervision to answer veri-

fication, treating binary imperfect verification signals (e.g. reward models and LM judges) as weak supervision voters that classify candidate solutions as correct or incorrect. This novel application combines predictions by converting these diverse signals into binary verdicts, enabling WEAVER to learn better verification strategies from weak but complementary verifiers.

Verification as another compute axis and aggregation: Recent work has explored verification as a new scaling axis (Lifshitz et al., 2025; Liu et al., 2025b; Zhao et al., 2025; Singhi et al., 2025b; Stroebl et al., 2024; Chen et al., 2025). However this work limits their analysis to one verifier, and instead scale how many times to verify (Zhao et al., 2025). Approaches that do leverage multiple verifiers often rely on substantial amounts of labeled data for aggregation or creating specialized verifiers (Kirchner et al., 2024; Lifshitz et al., 2025). With WEAVER, we show that it is possible to combine verifiers without ground truth labels, even when they are not specialized. Other work has focused on combining multiple verifiers for post-training the base model using RLHF (Wang et al., 2024d; Eisenstein et al., 2023b; Wang et al., 2025).

3. Preliminaries

First, we define the problem of how to select among repeated samples. We then define verifiers and key evaluation metrics, including the generation-verification gap.

Problem Definition Let $q \in \mathcal{Q}$ be a input text query, and let $r \in \mathcal{R} \sim \mathcal{M}(q)$ be a corresponding response sampled from language model $\mathcal{M}$ with non-zero temperature. For a given query-response pair (q, r), we define $y : \mathcal{Q} \times \mathcal{R} \to \{0, 1\}$ such that $y(q, r)$ is the correctness label of r for q.

We are given an unlabeled test dataset $= \{(q_i, r_i)\}_{i=1}^{n}$, where $r_i = \{r_{ij}\}_{j=1}^{K}$ consists of K repeatedly sampled responses from $\mathcal{M}$ for each q_i. We also assume access to a small labeled development dataset $\subset$, comprising 1% of the test set (e.g. 5 to 10 query-answer pairs), which is used to estimate global statistics such as the task difficulty probability, $\Pr(y_{ij} = 1)$. We do not have access to true labels $yij := y(q_i, r_{ij})$ for any i, j in $\setminus$.

For each $(q_i, r_i) \in$, our goal is to select a correct response $j^\star \in [K]$ that satisfies $y_{ij^\star} = 1$. We can broadly describe this selection rule using a scoring function $f : \mathcal{Q} \times \mathcal{R} \to \mathbb{R}$, namely $j^\star := \arg\max_j f^\star(q_i, r_{ij})$.

Using verifiers A verifier, either a reward model or an LM prompted as a judge, can be expressed as a scoring function on query-response pairs $v : \mathcal{Q} \times \mathcal{R} \to \mathbb{R}$. For reward models, the verifier score is continuous, while for LM judges, the verifier score is typically discrete (for our setup, we use $[0, 1]$ and $\{0, 1\}$, respectively). We assume that we have access to multiple verifiers $= \{v_1, \ldots, v_m\}$. We apply each of the m verifiers to each (q_i, r_{ij}), for a total of nmK scores on , with $s_{ijk} := v_k(q_i, r_{ij})$. We aim to use $\mathcal{V}$ to construct a verification strategy f.

Evaluation metrics The $Pass@1$ metric is the probability that an LM's first response is correct. $Pass@K$ generalizes this metric and is defined as the probability that there exists a correct response among K generated responses: $Pass@K = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(\exists j \in [K] : y_{ij} = 1)$. This metric is independent of the verification strategy, and depends on the choice of $\mathcal{M}$, K, and the task dataset. The success rate of a verification strategy $\hat{f}$ is $\frac{1}{n}\sum_{i=1}^{n} y_{i\hat{j}}$, where $\hat{j} = \arg\max_{j\in[k]} \hat{f}(q_i, r_{ij})$. Success rate is dependent on the verification strategy and bounded by Pass@K, and equality is obtained with oracle verification (i.e., $\hat{f} = f^\star$ can always select a correct j as long as it exists).

We define the *generation-verification gap* as Pass@K - Success Rate. A large positive gap indicates that although correct answers are generated, the verification strategy fails to select them consistently. We aim to close this gap and will use it to evaluate verification strategies.

4. WEAVER: A Framework for Weak Verifier Aggregation

In Section 4.1, we demonstrate that naively averaging multiple verifier scores to select responses significantly underperforms weighted ensembles; however, common methods for computing weights require labeled data (Schapire, 2013; Ying et al., 2015). We introduce WEAVER (Section 4.2), a method for weighted aggregation of verifier scores with minimal data that draws inspiration from Weak Supervision. Unlike prior work, WEAVER adapts weak supervision to verification by addressing challenges unique to verifier aggregation, such as inconsistent score formats and the presence of low-quality or adversarial verifiers. To our knowledge, this is the first framework to successfully apply weak supervision to ensemble verifier scores for response selection.

4.1. How to aggregate multiple verifiers: weighted vs unweighted ensembles

A straightforward approach for using multiple verifiers is a naive ensemble—selecting the response with the highest average verifier score: $f(q_i, r_{ij}) = \frac{1}{m}\sum_{k=1}^{m} s_{ijk}$. This approach (Lifshitz et al., 2025) does not consider the relative accuracy of verifiers. However, we observed that there is significant variation in the success rates of individual verifiers—spanning a range of up to 37.5%—suggesting that naive ensembles could be suboptimal (Table 16).

An alternative is to use a weighted ensemble. One approach is to use a labeled dataset to identify and use the top-

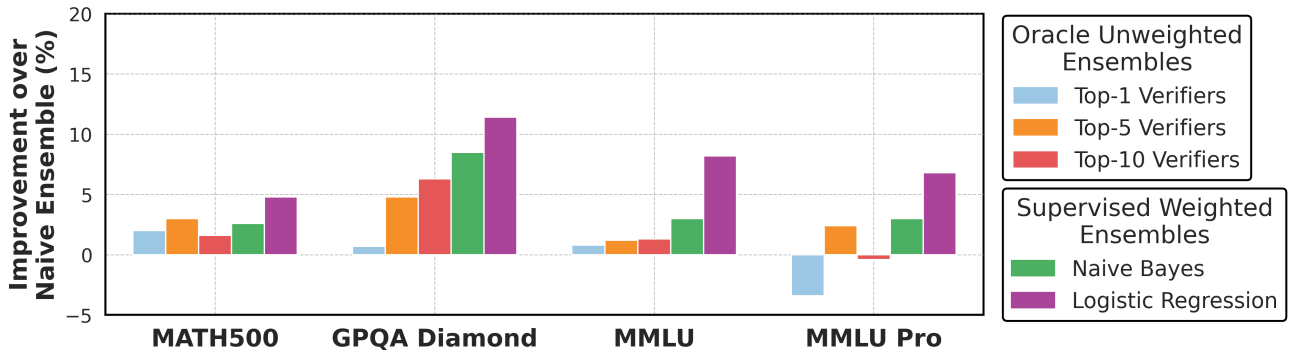

Figure 2: **Weighted Verifier Ensembles Outperform Naive Verifier Ensembles**: By using oracle data to keep the best verifiers (i.e. *top-K verifier ensembles*) or learn aggregation weights for verifiers (i.e. *supervised weighted ensembles*), we can improve beyond naive combinations of the verifiers available by 3.6% and 7.8%, on average, respectively.

performing verifier, effectively assigning a weight of 0 to discarded verifiers. Other strategies include using Logistic Regression or a Naive Bayes classifier, where the scoring function $f(q_i, r_{ij})$ is the probability $\Pr(y_{ij} = 1|s_{ij1}, \ldots, s_{ijm})$. These classifiers are fit using labeled data and can be either modeled as a logistic function or factorized using Bayes' rule and independence assumptions, respectively.

In Figure 2, we compare a naive ensemble with weighted ensembles for several tasks, using Llama 3.3 70B Instruct to generate responses and using a collection of 33 7B-72B reward models and LM judges as verifiers (Appendix C.1). We see that using a weighted ensemble can achieve up to 11.2 points higher success rate than the naive ensemble. However, all weighted ensembles shown are "oracle" methods: they are computed using y_{ij} for all $i \in [n], j \in [K]$, although in practice these labels are unknown for . In fact, when we instead use $0.01n$ labeled samples, accuracy drops by 20.1% on average (Table 17). This raises the question of how to best construct weighted ensembles with limited labeled data.

4.2. WEAVER: weighted ensembling of verifier scores with minimal labeled data

We first describe the WS method we use in WEAVER to construct a weighted ensemble over binary verifier scores. Because verifiers often produce scores in inconsistent formats and exhibit low accuracies—challenges not typically encountered in traditional WS—we introduce a binarization and verifier discarding strategy in Appendices B.2 and B.3 to discard low-quality verifiers and ensure that only sufficiently reliable binary scores are used as input to the WS method.

4.2.1. WEAK SUPERVISION ALGORITHM

In Weak Supervision, the input is an unlabeled dataset, where each entry has multiple binary "votes" on the true label. Applied to our setting, each entry is a query-response pair, forming a dataset of size nK, and verifier scores s_{ijk} are binarized into votes $\bar{s}_{ijk} \in \{0, 1\}$ for all i, j, k. Our goal is to predict the probability that a response is correct, $\Pr(y_{ij} = 1|s_{ij1}, \ldots, s_{ijm})$ for all i, j.

WS model We can view all y_{ij} across query-response pairs as samples of an unknown random variable Y and each $\bar{s}_{ijk}$ across i, j as samples of a random variable S_k. WS then defines a latent variable graphical model over the random binary vector $\{Y, S_1, \ldots, S_m\}$, where Y is latent while $S_1, \ldots S_m$ are observable. While existing WS methods assume various models, one common assumption is that $S_i \perp S_j|Y$ for each S_i, S_j. That is, S_i and S_j are conditionally independent given Y; intuitively, each verifier is assumed to capture independent aspects of the correctness of the response (Figure 22 in Appendix C.4). Under this assumption, we can write the posterior probability of a correct generation as the following, for some given binary verifier scores $\{\bar{s}_1, \ldots, \bar{s}_m\}$:

$$\Pr(Y = 1|S_1 = \bar{s}_1, \ldots, S_m = \bar{s}_m) = \frac{\prod_{i=1}^{m} \Pr(S_i = \bar{s}_i|Y = 1) \Pr(Y = 1)}{\Pr(S_1 = \bar{s}_1, \ldots, S_m = \bar{s}_m)}. \quad (1)$$

The weighted ensemble score for each query-response pair can thus be written in terms of: 1) $\Pr(S_1 = \bar{s}_1, \ldots, S_m = \bar{s}_m)$, which can be computed from the data; 2) $\Pr(Y = 1)$, which can be estimated from ; and 3) $\Pr(S_i = \bar{s}_i|Y = 1)$, or equivalently $\Pr(S_i = 1|Y = 1)$, which is the verifier's "accuracy parameter"—this cannot be computed directly since we do not have access to Y. Next, we discuss how to estimate these accuracy parameters, $\Pr(S_i = 1|Y = 1)$, without labels.

WS parameter estimation We outline a parameter estimation technique first introduced in (Ratner et al., 2020). Due

to the assumption that $S_i \perp S_j | Y$, the following equation holds:

$$\begin{aligned}\Pr(S_i, S_j) &= \Pr(S_i, S_j | Y=1)\Pr(Y=1) \\ &\quad + \Pr(S_i, S_j | Y=0)\Pr(Y=0) \\ &= \Pr(S_i|Y=1)\Pr(S_j|Y=1)\Pr(Y=1) \\ &\quad + \Pr(S_i|Y=0)\Pr(S_j|Y=0)\Pr(Y=0). \quad (2)\end{aligned}$$

Note that $\Pr(S_i, S_j)$ can be computed from the known verifier scores, and $\Pr(Y=1)$ is estimated from . Then, (2) is a quadratic equation over the accuracy parameters. We can write this equation for every pair S_i, S_j, and for every pair of values $\{0,1\}^2$ they can take. Furthermore, we can write another type of equation over the accuracy parameters:

$$\begin{aligned}\Pr(S_i=1) &= \Pr(S_i=1|Y=1)\Pr(Y=1) \quad (3) \\ &\quad + \Pr(S_i=1|Y=0)\Pr(Y=0). \quad (4)\end{aligned}$$

This is a consistency property that holds regardless of the conditional independence assumption, and we can write this equation for each of the m S_i's. Because we know that the accuracy parameters should follow equations 2 and 4, we can construct an objective function that aims to minimize the difference between the left and right hand sides of these equations. We write this efficiently in matrix notation. Let $P \in \mathbb{R}^{2\times 2}$ be a diagonal matrix with diagonal $[\Pr(Y=0)\ \ \Pr(Y=1)]$. Define $\mu \in \mathbb{R}^{m\times 2}$ to be the matrix of accuracy parameters, and define $O \in \mathbb{R}^{2m\times 2m}$ to be a matrix over the joint probabilities of pairs of S_i, S_j; more formally: Let $p_{i,y} = \Pr(S_i = 1 | Y = y)$ and $p_i = \Pr(S_i = 1)$.

$$\begin{aligned}\mu_{2i-1:2i,1:2} &= \begin{pmatrix} 1-p_{i,0} & 1-p_{i,1} \\ p_{i,0} & p_{i,1}\end{pmatrix}, \\ O_{2i-1:2i,2i-1:2i} &= \begin{pmatrix} 1-p_i & 0 \\ 0 & p_i\end{pmatrix} \ \forall i \in [m], \\ O_{2i-1:2i,2j-1:2j} &= \begin{pmatrix} \Pr(S_i=0,S_j=0) & \Pr(S_i=0,S_j=1) \\ \Pr(S_i=1,S_j=0) & \Pr(S_i=1,S_j=1)\end{pmatrix} \\ &\quad \forall i \neq j \in [m]. \quad (5)\end{aligned}$$

Let off-diag denote the elements of a matrix that lie outside its 2×2 block diagonal. Then, to estimate μ that satisfies both equations 2 and 4, we have the following objective:

$$\text{minimize}_\mu \| O_{\text{off-diag}} - (\mu P \mu^T)_{\text{off-diag}} \|^2 + \| \text{diag}(O) - \mu P \mathbf{1}^T \|^2 \quad (6)$$

We optimize 6 using gradient descent to estimate the verifier accuracy parameters. These estimates are then used in Equation (1) to select the response with the highest estimated posterior. To further improve modeling of verifier accuracies, we explore whether partitioning the query distribution by empirical difficulty can yield better weak supervision estimates. As detailed in Appendix B.4, we cluster queries based on the observed ratio of correct to incorrect generations, and fit a separate WEAVER model within each difficulty bucket. We provide more details in Appendix B.1.

5. Results

In section 5.1, we provide empirical results on WEAVER's performance compared to other approaches for selecting responses in repeated sampling. In section 5.2, we study how WEAVER's performance scales along several axes: the number of responses, model size, verifier counts, and inference compute.

Datasets, Verifiers, and Baselines Our reward models range in size from 8B to 72B, are all open-source, and are obtained from RewardBench (Lambert et al., 2024), a popular evaluation tool for reward models. We prompt open-source language models from Chatbot Arena (Chiang et al., 2024) to serve as judges. Unless specified, we use Llama 3.3 70B Instruct to generate responses and use all 33 reward models and judges. We evaluate on MATH500, GPQA Diamond, MMLU College, and MMLU Pro. See Appendix C.1 for more details.

We compare WEAVER against verifier-free baselines as well as standard verification strategies. First Sample, also known as Pass@1, only uses the first response and does not scale test-time compute or verification. Majority Voting involves repeated sampling but not verification, picking the most common final answer from the responses (Brown et al., 2024; Snell et al., 2024; Chen et al., 2024c). We compare against the highest scoring reward model and a naive ensemble of the top-10 reward models on RewardBench. We also evaluate two recently proposed methods that scale verification but do not use different verifier models or weighted ensembles: Self-Verification (Zhao et al., 2025) and Multi-Agent Verification (Lifshitz et al., 2025). Lastly, we report the oracle Pass@K rate, which establishes an upper bound for the success rate of these verification strategies.

5.1. WEAVER Shrinks the Gap with Frontier LMs

In Table 2, we evaluate WEAVER along with baseline verification methods, the first sample performance of frontier LMs, and the Pass@100 metric. We use LlaMA 3.3 70B Instruct to generate $K = 100$ responses per query. We find that WEAVER's weighted ensembling of multiple verifiers allows us to outperform majority vote by 15.5% and come within 4.2% of the Pass@100 oracle metric. Furthermore, WEAVER rivals the performance of frontier reasoning models—coming within 0.5% of OpenAI's o3-mini (OpenAI, 2025)—even though we use a non-reasoning model for generation.

5.2. WEAVER Improves Compute-Accuracy Trade-Off for Scaling

By proposing to combine multiple weak verifiers instead of one, we introduce yet another axis for test-time scaling. In this section, we study how well scaling verification with

Table 1: **Scaling Dimensions for Generation and Verification Models**

Scaling Dimension	Base Model	Verifier Type	Visuals
Sample Count: More Generations	Temperature-based sampling	Majority Vote, Weak Verifier, Top-K, WEAVER	Figure 3
Model Size: Larger Models	Llama 8B → 70B	RM-8B → RM-70B	Table 3
Verifier Count: More Models	Llama 8B/70B	RMs and LM Judges	Figure 21
Inference Compute: More FLOPs for Gen./Ver.	Temp-based sampling	Weak Verifiers + WEAVER	Figure 4

Table 2: **WEAVER Outperforms Baseline Verification Methods and Shrinks Gap with Frontier LMs.**

	Methodology	Generations (K)	Datasets				Average
			MATH 500	GPQA Diamond	MMLU College	MMLU Pro	
Baselines	First Sample	1	78.0%	42.9%	82.6%	69.9%	68.4%
	Majority Voting	100	83.0%	47.4%	84.1%	74.4%	72.2%
	Highest Scoring RM on RewardBench (Minghao Yang, 2024; Lambert et al., 2024)	100	78.2%	49.7%	86.0%	77.0%	**72.7%**
	Naive Ensemble of Top-10 RMs on RewardBench (Lambert et al., 2024)	100	75.4%	41.3%	88.1%	71.4%	69.1%
	Self-Verification (Zhao et al., 2025)	100	78.1%	43.1%	82.0%	69.5%	66.9%
	Multi-Agent Verification (Lifshitz et al., 2025)	100	81.3%	47.8%	84.1%	72.6%	71.6%
	WEAVER	100	93.4%	72.1%	94.9%	90.2%	**87.7%**
Frontier Approaches	GPT-4o (OpenAI, 2023)	1	77.4%	35.9%	87.1%	75.4%	69.0%
	Claude 3.7 Sonnet (Anthropic, 2025)	1	69.2%	48.0%	86.1%	78.1%	70.4%
	Llama 4 Maverick (Meta, 2025)	1	87.6%	68.9%	91.1%	81.0%	82.2%
	o3-mini (OpenAI, 2025)	1	94.4%	74.0%	92.2%	86.0%	86.7%
	Oracle Verification (Pass@100)	100	98.6%	81.0%	96.0%	92.0%	**91.9%**

WEAVER interacts with common previously studied axes for verification, summarized in Table 1.

(1) Scaling Candidate Generations: we study the performance of verification methods as we increase the number of repeated samples in Figure 3. Based on prior work (Bradley & Terry, 1952; Chen et al., 2021), as the number of responses increases, we are more likely to see a correct response (i.e. Pass@K increases), and hence more likely to select a correct response given a good verification strategy. However, differences in verification translate into different scaling rates. We evaluate the performance of WEAVER and baselines for $K = 2^0$ to 2^{10}, comparing to o3-mini and Pass@K as well. Across all tasks, WEAVER yields the most substantial gains when scaling the number of generations. WEAVER consistently narrows the generation-verification gap with the oracle upper bound (Pass@K) while alternative verification strategies plateau after a few generations. The effect is particularly pronounced on difficult tasks like GPQA. We detail the scaling trends observed in Figure 3 in Appendix C.3.

(2) Scaling Model Sizes: In Table 3, we study how WEAVER applied on smaller models (both verifiers and for generating responses) can allow us to match the performance of larger models, enabling weak-to-strong verification. We consider an 8B setting—using LlaMA 3.1 8B to generate responses along with 8B verifiers—and compare this to a 70B setting (LlaMA 3.3 70B Instruct, 8B-72B verifiers) as well as o3-mini. We see that WEAVER applied at the 8B scale comes within 1.6% of the majority vote baseline at the 70B scale, and WEAVER at 70B surpasses o3-mini by 1.0%, demonstrating a weak-to-strong verification phenomenon. Verifier calibration details are available in Appendix C.5.

(3) Scaling Verifier Count: Two axes for scaling verification are **(1)** the number of verifiers used and **(2)** the number of scores sampled from each verifier. Figure 21 shows how performance changes as we ensemble 1 to 15 verifiers using both naive averaging and WEAVER. Verifiers are greedily added in order of individual accuracy, from highest to lowest. Aggregating more verifiers improves performance by up to 8.5% over the top-1 verifier. As shown in Figure 21, WEAVER consistently outperforms naive ensemble averaging across both *Oracle Top-5 Verifiers* and *Total Verifiers* configurations for verifier ensembling, with improvements ranging from +2.4% to +10.1% across all datasets. The performance gains are particularly pronounced on GPQA Diamond (+10.1%) and MMLU Pro (+5.1%), demonstrating WEAVER's effectiveness in aggregating verifier signals through learned weights rather than simple averaging. However, gains diminish as more models are added—reflecting the classic ensemble bias-variance tradeoff: initial improvements stem from variance reduction, while additional verifiers contribute redundant signal due to correlated biases on hard examples (Abe et al., 2024). We compare alternative score calibration strategies beyond WEAVER's binary

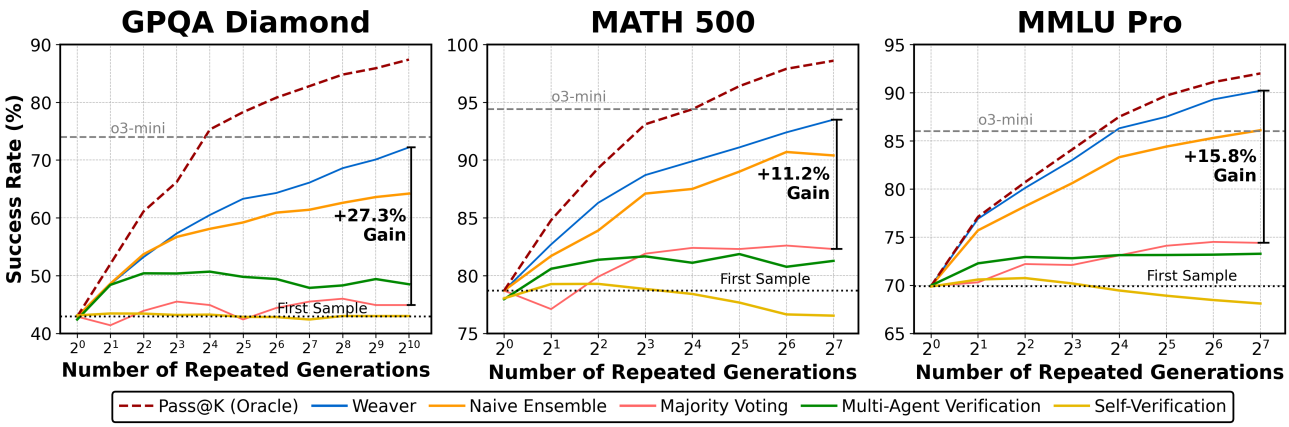

Figure 3: **Scaling Generations Boosts Performance with WEAVER**: The generation-verification gap shrinks when increasing K and leveraging WEAVER, outperforming alternative verification methods by an average 18.3%.

Table 3: **WEAVER Reduces Gap between Model Classes: 8B and 70B, 70B and Frontier LM**

Generator Model	Verifier Model	Aggregation Strategy	Datasets				Average
			MATH	GPQA Diamond	MMLU College	MMLU Pro	
Llama 3.1 8B Instruct	N/A	Majority Vote	69.0%	30.5%	72.7%	56.4%	57.2%
	8B and below	WEAVER	80.0%	47.1%	85.7%	67.2%	**70.0%**
		Δ w. WEAVER	+11.0%	+16.6%	+13.0%	+10.2%	+12.8%
Llama 3.3 70B Instruct	N/A	Majority Vote	83.0%	44.9%	84.1%	74.4%	71.6%
	72B and below	WEAVER	93.4%	72.2%	94.9%	90.2%	**87.6%**
		Δ w. WEAVER	+10.4%	+27.3%	+10.8%	+15.8%	+16.0%
o3-mini	N/A	First Sample	94.4%	74.0%	92.2%	86.0%	**86.7%**

transformation in Appendix C.7, and find that the default binarization yields the strongest downstream selection performance. We also explore scaling the number of scores per verifier—via prompt tuning or temperature variation—in Appendix C.5. While this yields modest improvements, increasing verifier count remains the more effective strategy. That said, both methods are complementary and can be combined for further gains.

(4) Scaling Test-Time Compute: We study how performance scales in the total compute used for both verification and repeated generations. Figure 4 shows the relationship between inference-time compute and success rate for different generation-verification systems. For each method, we scale the number of generations exponentially from 1 to 100 and plot the required inference compute for generation and verification together versus the success rate. Note that Figure 4 differs from Figure 3, since Majority Voting requires 0 verification inference calls while WEAVER requires 30+ calls for the weak verifiers. We find that WEAVER achieves the highest maximum success rate; notably, majority voting plateaus at around 2^2 to 2^3 ExaFLOPs per query while WEAVER continues scaling until 512 ExaFLOPs. However, the additional compute required for WEAVER can be prohibitive. We explore how to reduce this computational burden while retaining WEAVER's performance in the next section.

6. WEAVER Distillation: Improving Verification Efficiency at Inference

We explore distillation strategies for fine-tuning a smaller LM as a task-specific verifier. In particular, we train *cross-encoders*; the input is a concatenated query-response pair, while the output is WEAVER's pseudolabel generated from Weak Supervision, namely $\Pr(y_{ij} = 1 | s_{ij1}, \ldots, s_{ijm})$ (see Section 4). For the model, we selected ModernBERT-Large (396M) (Warner et al., 2024). For more details, please see Appendix C.6.

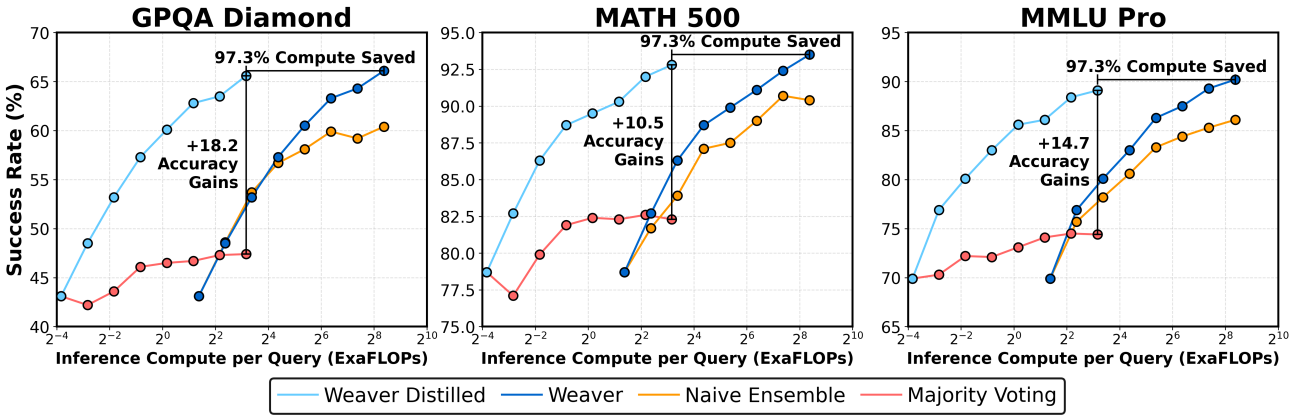

Figure 4: **WEAVER Improves the Accuracy-Compute Performance Trade-Offs.** Success rate (%) as a function of total inference compute per query (generation and verification compute, log scaled) for different verification strategies. Each point represents a different number of candidate generations (from 2^0 to 2^7). WEAVER achieves the highest accuracy while requiring more compute than Majority Voting but demonstrates continued scaling benefits, while WEAVER Distilled maintains most of WEAVER's performance gains with 97.3% compute savings and substantial accuracy improvements over baseline methods.

Figure 5 shows the performance of WEAVER on the Llama-70B generations against the cross-encoder on GPQA Diamond. Across tasks, we find that the distilled cross-encoder is able to capture 98.2% of the performance of WEAVER. When running WEAVER with all the verifiers, it costs 35.35 exaFLOPs for each query's set of 100 samples. Running a 400M cross-encoder costs 1.01 exaFLOPs for evaluating 100 samples and *reduces compute cost by more than three orders of magnitude, saving 99.97% of the FLOPs originally required for running the 70B verifiers*. We also outperform majority voting by 23.2% while only incurring a 0.57% increased inference cost over only generating the responses. We see similar results for additional datasets in Figure 22 (Appendix C.6).

These results suggest that, through distillation, we can capture the combined strengths of the weak verifiers used for WEAVER, and deploy generalizable and lightweight cross-encoders that use only a fraction of the parameters used for generation. This reduces our hardware constraints considerably; *rather than utilizing an 8-GPU node per 70B verifier (i.e. Nvidia H200s with 80B memory), we only require a single A100 GPU with 32GB of memory for our cross-encoder.*

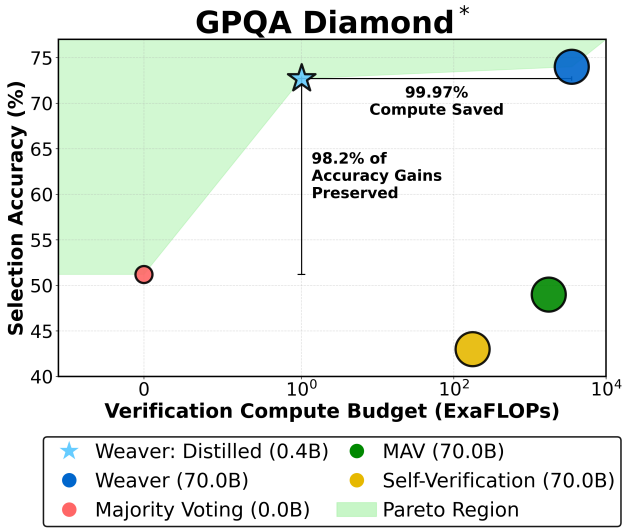

Figure 5: **Distilling WEAVER into a 400M Cross-Encoder Almost Entirely Captures the Performance of WEAVER, Yielding 99.97% Compute Savings.** *We train/evaluate on an 80:20 split.

References

Abe, T., Buchanan, E. K., Pleiss, G., and Cunningham, J. P. Pathologies of predictive diversity in deep ensembles. Transactions on Machine Learning Research, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=TQfQUksaC8. Featured Certification.

Anthropic. Claude 3.7 sonnet and claude code, February 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet. Announcement blog post, 5 min read.

Arora, S., Narayan, A., Chen, M. F., Orr, L., Guha, N., Bhatia, K., Chami, I., Sala, F., and Ré, C. Ask me anything: A simple strategy for prompting language models, 2022. URL https://arxiv.org/abs/2210.02441.

Askell, A., Bai, Y., Chen, A., Drain, D., Ganguli, D., Henighan, T., Jones, A., Joseph, N., Mann, B., DasSarma, N., et al. A General Language Assistant as a Laboratory for Alignment. ArXiv Preprint arXiv:2112.00861, 2021.

Bradley, R. A. and Terry, M. E. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. Biometrika, 39(3/4):324–345, 1952.

Brown, B., Juravsky, J., Ehrlich, R., Clark, R., Le, Q. V., Ré, C., and Mirhoseini, A. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. Learning to rank: from pairwise approach to listwise approach. In Proceedings of the 24th international conference on Machine learning, pp. 129–136, 2007.

Chaudhari, S., Aggarwal, P., Murahari, V., Rajpurohit, T., Kalyan, A., Narasimhan, K., Deshpande, A., and da Silva, B. C. RLHF Deciphered: A Critical Analysis of Reinforcement Learning from Human Feedback for LLMs, 2024.

Chen, G. H., Chen, S., Liu, Z., Jiang, F., and Wang, B. Humans or LLMs as the Judge? A Study on Judgement Bias. ArXiv Preprint arXiv:2402.10669, 2024a.

Chen, J., Ren, J., Chen, X., Yang, C., Sun, R., and Arık, S. Ö. Sets: Leveraging self-verification and self-correction for improved test-time scaling. arXiv preprint arXiv:2501.19306, 2025.

Chen, L., Davis, J. Q., Hanin, B., Bailis, P., Stoica, I., Zaharia, M., and Zou, J. Are more llm calls all you need? towards scaling laws of compound inference systems, 2024b. URL https://arxiv.org/abs/2403.02419.

Chen, L., Davis, J. Q., Hanin, B., Bailis, P., Stoica, I., Zaharia, M., and Zou, J. Are more llm calls all you need? towards scaling laws of compound inference systems. arXiv preprint arXiv:2403.02419, 2024c.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021.

Chen, M. F., Fu, D. Y., Adila, D., Zhang, M., Sala, F., Fatahalian, K., and Ré, C. Shoring up the foundations: fusing model embeddings and weak supervision. In Cussens, J. and Zhang, K. (eds.), Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, volume 180 of Proceedings of Machine Learning Research, pp. 357–367. PMLR, 01–05 Aug 2022. URL https://proceedings.mlr.press/v180/chen22e.html.

Chiang, W.-L., Zheng, L., Sheng, Y., Angelopoulos, A. N., Li, T., Li, D., Zhang, H., Zhu, B., Jordan, M., Gonzalez, J. E., and Stoica, I. Chatbot arena: An open platform for evaluating llms by human preference, 2024.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep Reinforcement Learning from Human Preferences. Advances in Neural Information Processing Systems, 30, 2017.

Clark, E., August, T., Serrano, S., Haduong, N., Gururangan, S., and Smith, N. A. All that's 'human' is not gold: Evaluating human evaluation of generated text, 2021. URL https://arxiv.org/abs/2107.00061.

Cui, G., Yuan, L., Wang, Z., Wang, H., Li, W., He, B., Fan, Y., Yu, T., Xu, Q., Chen, W., et al. Process reinforcement through implicit rewards. arXiv preprint arXiv:2502.01456, 2025.

Dorka, N. Quantile regression for distributional reward models in rlhf. arXiv preprint arXiv:2409.10164, 2024.

Dubois, Y., Galambosi, B., Liang, P., and Hashimoto, T. B. Length-Controlled AlpacaEval: A Simple Way to Debias

Automatic Evaluators. ArXiv Preprint arXiv:2404.04475, 2024a.

Dubois, Y., Li, C. X., Taori, R., Zhang, T., Gulrajani, I., Ba, J., Guestrin, C., Liang, P. S., and Hashimoto, T. B. AlpacaFarm: A Simulation Framework for Methods that Learn from Human Feedback. Advances in Neural Information Processing Systems, 36, 2024b.

Eisenstein, J., Berant, J., Nagpal, C., Agarwal, A., Beirami, A., D'Amour, A. N., Dvijotham, K. D., Heller, K. A., Pfohl, S. R., and Ramachandran, D. Reward Model Underspecification in Language Model Alignment. In NeurIPS 2023 Workshop on Distribution Shifts: New Frontiers with Foundation Models, 2023a.

Eisenstein, J., Nagpal, C., Agarwal, A., Beirami, A., D'Amour, A., Dvijotham, D., Fisch, A., Heller, K., Pfohl, S., Ramachandran, D., et al. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. arXiv preprint arXiv:2312.09244, 2023b.

Es, S., James, J., Espinosa-Anke, L., and Schockaert, S. RAGAs: Automated Evaluation of Retrieval Augmented Generation. ArXiv Preprint arXiv:2309.15217, 2023.

Fu, D., Chen, M., Sala, F., Hooper, S., Fatahalian, K., and Ré, C. Fast and three-rious: Speeding up weak supervision with triplet methods. In International conference on machine learning, pp. 3280–3291. PMLR, 2020.

Fu, J., Ng, S.-K., Jiang, Z., and Liu, P. GPTScore: Evaluate as You Desire. Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), 2023.

Guha, N., Chen, M. F., Chow, T., Khare, I. S., and Re, C. Smoothie: Label free language model routing. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024.

Hall, A. R. Generalized method of moments. A companion to theoretical econometrics, pp. 230–255, 2003.

HazyResearch. metal. https://https://github.com/HazyResearch/metal, 2018.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. arXiv preprint arXiv:2103.03874, 2021.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. arXiv preprint arXiv:2203.15556, 2022a.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022b. URL https://arxiv.org/abs/2203.15556.

Hosking, T., Blunsom, P., and Bartolo, M. Human feedback is not gold standard, 2024. URL https://arxiv.org/abs/2309.16349.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Kalra, N. and Tang, L. Verdict: A library for scaling judge-time compute, 2025. URL https://arxiv.org/abs/2502.18018.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Karpinska, M., Akoury, N., and Iyyer, M. The perils of using mechanical turk to evaluate open-ended text generation, 2021. URL https://arxiv.org/abs/2109.06835.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., and Potts, C. Dspy: Compiling declarative language model calls into self-improving pipelines. arXiv preprint arXiv:2310.03714, 2023.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Kirchner, J. H., Chen, Y., Edwards, H., Leike, J., McAleese, N., and Burda, Y. Prover-verifier games improve legibility of llm outputs. arXiv preprint arXiv:2407.13692, 2024.

Koo, D., Choi, Y., and Choi, E. Cognitive Biases in Large Language Models as Evaluators. ArXiv Preprint arXiv:2312.05441, 2023.

Lambert, N. and Calandra, R. The alignment ceiling: Objective mismatch in reinforcement learning from human feedback. ArXiv Preprint arXiv:2311.00168, 2023.

Lambert, N., Pyatkin, V., Morrison, J., Miranda, L., Lin, B. Y., Chandu, K., Dziri, N., Kumar, S., Zick, T., Choi, Y., Smith, N. A., and Hajishirzi, H. RewardBench: Evaluating Reward Models for Language Modeling, 2024.

Li, X., Zhang, T., Dubois, Y., Taori, R., Gulrajani, I., Guestrin, C., Liang, P., and Hashimoto, T. B. Alpacaeval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 2023.

Lifshitz, S., McIlraith, S. A., and Du, Y. Multi-agent verification: Scaling test-time compute with multiple verifiers, 2025. URL https://arxiv.org/abs/2502.20379.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Liu, C. Y. and Zeng, L. Skywork Reward Model Series. https://huggingface.co/Skywork, September 2024.

Liu, C. Y., Zeng, L., Liu, J., Yan, R., He, J., Wang, C., Yan, S., Liu, Y., and Zhou, Y. Skywork-reward: Bag of tricks for reward modeling in llms. arXiv preprint arXiv:2410.18451, 2024.

Liu, R., Gao, J., Zhao, J., Zhang, K., Li, X., Qi, B., Ouyang, W., and Zhou, B. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling, 2025a. URL https://arxiv.org/abs/2502.06703.

Liu, R., Gao, J., Zhao, J., Zhang, K., Li, X., Qi, B., Ouyang, W., and Zhou, B. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. arXiv preprint arXiv:2502.06703, 2025b.

Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., and Zhu, C. G-eval: Nlg evaluation using gpt-4 with better human alignment. ArXiv Preprint arXiv:2303.16634, 2023.

Meta. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation, 4 2025. URL https://ai.meta.com/blog/llama-4/. Accessed: 2025-05-12.

Minghao Yang, Chao Qu, X. T. Inf-orm-llama3.1-70b, 2024. URL https://huggingface.co/infly/INF-ORM-Llama3.1-70B.

OpenAI. GPT-4 Technical Report. ArXiv Preprint arXiv:2303.08774, 2023.

OpenAI. Openai o3-mini system card. Technical report, OpenAI, January 2025. URL https://cdn.openai.com/o3-mini-system-card-feb10.pdf. Publication.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35:27730–27744, 2022.

Pan, Q., Ashktorab, Z., Desmond, M., Cooper, M. S., Johnson, J., Nair, R., Daly, E., and Geyer, W. Human-Centered Design Recommendations for LLM-as-a-judge. In Proceedings of the 1st Human-Centered Large Language Modeling Workshop, pp. 16–29, 2024.

Puri, I., Sudalairaj, S., Xu, G., Xu, K., and Srivastava, A. A probabilistic inference approach to inference-time scaling of llms using particle-based monte carlo methods, 2025. URL https://arxiv.org/abs/2502.01618.

Ratner, A., Hancock, B., Dunnmon, J., Sala, F., Pandey, S., and Ré, C. Training complex models with multi-task weak supervision. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pp. 4763–4771, 2019.

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., and Ré, C. Snorkel: rapid training data creation with weak supervision. The VLDB Journal, 29(2):709–730, 2020.

Ratner, A. J., De Sa, C. M., Wu, S., Selsam, D., and Ré, C. Data programming: Creating large training sets, quickly. Advances in neural information processing systems, 29, 2016.

Ravi, S. S., Mielczarek, B., Kannappan, A., Kiela, D., and Qian, R. Lynx: An Open Source Hallucination Evaluation Model, 2024.

Reimers, N. and Gurevych, I. Making monolingual sentence embeddings multilingual using knowledge distillation. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 11 2020. URL https://arxiv.org/abs/2004.09813.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. GPQA: A graduate-level google-proof q&a benchmark. In First Conference on Language Modeling, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Rosset, S., Zhu, J., and Hastie, T. Margin maximizing loss functions. Advances in neural information processing systems, 16, 2003.

Saad-Falcon, J., Khattab, O., Potts, C., and Zaharia, M. ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems. ArXiv Preprint arXiv:2311.09476, 2023.

Saad-Falcon, J., Lafuente, A. G., Natarajan, S., Maru, N., Todorov, H., Guha, E., Buchanan, E. K., Chen, M., Guha, N., Ré, C., and Mirhoseini, A. Archon: An architecture search framework for inference-time techniques. arXiv preprint arXiv:2409.15254, 2024a.

Saad-Falcon, J., Vivek, R., Berrios, W., Naik, N. S., Franklin, M., Vidgen, B., Singh, A., Kiela, D., and Mehri, S. Lmunit: Fine-grained evaluation with natural language unit tests, 2024b. URL https://arxiv.org/abs/2412.13091.

Schapire, R. E. Explaining adaboost. In Empirical inference: festschrift in honor of vladimir N. Vapnik, pp. 37–52. Springer, 2013.

Shin, C., Li, W., Vishwakarma, H., Roberts, N., and Sala, F. Universalizing weak supervision. arXiv preprint arXiv:2112.03865, 2021.

Singhal, P., Goyal, T., Xu, J., and Durrett, G. A long way to go: Investigating length correlations in rlhf. ArXiv Preprint arXiv:2310.03716, 2023.

Singhi, N., Bansal, H., Hosseini, A., Grover, A., Chang, K.-W., Rohrbach, M., and Rohrbach, A. When to solve, when to verify: Compute-optimal problem solving and generative verification for llm reasoning, 2025a. URL https://arxiv.org/abs/2504.01005.

Singhi, N., Bansal, H., Hosseini, A., Grover, A., Chang, K.-W., Rohrbach, M., and Rohrbach, A. When to solve, when to verify: Compute-optimal problem solving and generative verification for llm reasoning, 2025b. URL https://arxiv.org/abs/2504.01005.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Song, M., Su, Z., Qu, X., Zhou, J., and Cheng, Y. Prmbench: A fine-grained and challenging benchmark for process-level reward models, 2025a. URL https://arxiv.org/abs/2501.03124.

Song, Y., Zhang, H., Eisenach, C., Kakade, S., Foster, D., and Ghai, U. Mind the gap: Examining the self-improvement capabilities of large language models, 2025b. URL https://arxiv.org/abs/2412.02674.

Stroebl, B., Kapoor, S., and Narayanan, A. Inference scaling flaws: The limits of llm resampling with imperfect verifiers, 2024. URL https://arxiv.org/abs/2411.17501.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., and Wei, J. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022. URL https://arxiv.org/abs/2210.09261.

Tang, L., Laban, P., and Durrett, G. MiniCheck: Efficient Fact-Checking of LLMs on Grounding Documents, 2024.

Verga, P., Hofstatter, S., Althammer, S., Su, Y., Piktus, A., Arkhangorodsky, A., Xu, M., White, N., and Lewis, P. Replacing judges with juries: Evaluating llm generations with a panel of diverse models, 2024. URL https://arxiv.org/abs/2404.18796.

Vishwakarma, H. and Sala, F. Lifting weak supervision to structured prediction. Advances in Neural Information Processing Systems, 35:37563–37574, 2022.

Wang, B., Zheng, R., Chen, L., Liu, Y., Dou, S., Huang, C., Shen, W., Jin, S., Zhou, E., Shi, C., Gao, S., Xu, N., Zhou, Y., Fan, X., Xi, Z., Zhao, J., Wang, X., Ji, T., Yan, H., Shen, L., Chen, Z., Gui, T., Zhang, Q., Qiu, X., Huang, X., Wu, Z., and Jiang, Y.-G. Secrets of RLHF in Large Language Models Part II: Reward Modeling, 2024a.

Wang, H., Xiong, W., Xie, T., Zhao, H., and Zhang, T. Interpretable preferences via multi-objective reward modeling and mixture-of-experts. ArXiv Preprint arXiv:2406.12845, 2024b.

Wang, J., Liang, Y., Meng, F., Sun, Z., Shi, H., Li, Z., Xu, J., Qu, J., and Zhou, J. Is ChatGPT a good NLG Evaluator? A Preliminary Study. ArXiv Preprint arXiv:2303.04048, 2023a.

Wang, J., Xie, R., Zhu, S., Wang, J., Athiwaratkun, B., Dhingra, B., Song, S. L., Zhang, C., and Zou, J. Improving model alignment through collective intelligence of open-source llms, 2025. URL https://arxiv.org/abs/2505.03059.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. arXiv preprint arXiv:2406.01574, 2024c.

Wang, Z., Dong, Y., Zeng, J., Adams, V., Sreedhar, M. N., Egert, D., Delalleau, O., Scowcroft, J. P., Kant, N., Swope, A., and Kuchaiev, O. HelpSteer: Multi-Attribute Helpfulness Dataset for SteerLM, 2023b.

Wang, Z., Nagpal, C., Berant, J., Eisenstein, J., D'Amour, A., Koyejo, S., and Veitch, V. Transforming and combining rewards for aligning large language models. arXiv preprint arXiv:2402.00742, 2024d.

Warner, B., Chaffin, A., Clavié, B., Weller, O., Hallström, O., Taghadouini, S., Gallagher, A., Biswas, R., Ladhak, F., Aarsen, T., Cooper, N., Adams, G., Howard, J., and Poli, I. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference, 2024. URL https://arxiv.org/abs/2412.13663.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Xu, T., Helenowski, E., Sankararaman, K. A., Jin, D., Peng, K., Han, E., Nie, S., Zhu, C., Zhang, H., Zhou, W., Zeng, Z., He, Y., Mandyam, K., Talabzadeh, A., Khabsa, M., Cohen, G., Tian, Y., Ma, H., Wang, S., and Fang, H. The perfect blend: Redefining rlhf with mixture of judges, 2024. URL https://arxiv.org/abs/2409.20370.

Ying, L. et al. Decision tree methods: applications for classification and prediction. Shanghai archives of psychiatry, 27(2):130, 2015.

Yuan, L., Li, W., Chen, H., Cui, G., Ding, N., Zhang, K., Zhou, B., Liu, Z., and Peng, H. Free process rewards without process labels. arXiv preprint arXiv:2412.01981, 2024.

Zhang, L., Hosseini, A., Bansal, H., Kazemi, M., Kumar, A., and Agarwal, R. Generative Verifiers: Reward Modeling as Next-Token Prediction, 2024.

Zhao, E., Awasthi, P., and Gollapudi, S. Sample, scrutinize and scale: Effective inference-time search by scaling verification. arXiv preprint arXiv:2502.01839, 2025.

Zheng, K., Han, J. M., and Polu, S. Minif2f: a cross-system benchmark for formal olympiad-level mathematics, 2022. URL https://arxiv.org/abs/2109.00110.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023a.

Zheng, L., Xu, D., Dong, J., Zeng, A., Xie, S., Xing, E. P., and Liang, P. Evaluation Biases for Large Language Models. ArXiv Preprint arXiv:2305.17926, 2023b.

A. Table of Contents

B. WEAVER Methodology

B.1. Weak Supervision Model

We can construct a data generating model over response correctness y and the binary verifier outputs $\bar{s}$. The model is defined as:

$$\begin{aligned} \text{Response Correctness:} \quad & y_{ij} \sim \text{Bernoulli}(\pi)\ \forall i \in [n], j \in [K], \\ \text{Verifier Score:} \quad & \bar{s}_{ijk} \mid y_{ij} \sim \begin{cases} \text{Bernoulli}(w_{k,1}), & \text{if } y_{ij} = 1, \\ \text{Bernoulli}(1 - w_{k,0}), & \text{if } y_{ij} = 0, \end{cases} \forall i \in [n], j \in [K], k \in [m] \end{aligned}$$

where:

- π is the probability that a response is correct.
- $w_{k,1}$ is the true positive rate (TPR) of verifier k, and $w_{k,0}$ is the true negative rate (TNR), which we refer to as the verifier's *accuracy parameters.*

Here, each verifier k emits a binary score $\bar{s}_{ijk} \in \{0, 1\}$, which is assumed to be a noisy indicator of whether response y_{ij} is correct. The likelihood of the verifier's binary output $X_{ijk} \in \{0, 1\}$ is:

$$\Pr(S_k = \bar{s}_{ijk} \mid Y = y_{ij}) = \begin{cases} w_{k,1}, & \text{if } y_{ij} = 1 \text{ and } \bar{s}_{ijk} = 1, \\ 1 - w_{k,1}, & \text{if } y_{ij} = 1 \text{ and } \bar{s}_{ijk} = 0, \\ w_{k,0}, & \text{if } y_{ij} = 0 \text{ and } \bar{s}_{ijk} = 0, \\ 1 - w_{k,0}, & \text{if } y_{ij} = 0 \text{ and } \bar{s}_{ijk} = 1. \end{cases}$$

We are interested in estimating the correctness of a response $y_{ij} \in \{0, 1\}$ based on assessments from multiple verifiers $\bar{s}_{ij} = \{\bar{s}_{ij1}, \bar{s}_{ij2}, \ldots, \bar{s}_{ijm}\}$. Applying Bayes' Rule, we get:

$$\Pr(y_{ij} = 1 \mid S = \bar{s}_{ij}) = \frac{\Pr(S = \bar{s}_{ij} \mid y_{ij} = 1) \Pr(y_{ij} = 1)}{\Pr(\bar{s}_{ij})} \tag{7}$$

where $\Pr(\bar{s}_{ij}) = \sum_{y' \in \{0,1\}} \Pr(\bar{s}_{ij} \mid y_{ij} = y') \Pr(y_{ij} = y')$.

Equation (7) requires evaluating the full conditional likelihood:

$$\Pr(S = \bar{s}_{ij} \mid y_{ij}) = \Pr(S_1 = \bar{s}_{ij1}, S_2 = \bar{s}_{ij2}, \ldots, S_m = \bar{s}_{ijm} \mid y_{ij}),$$

which is a joint distribution over m binary random variables. Since each verifier $S_k \in \{0, 1\}$ is binary, then there are 2^m possible verifier output configurations for $S \in \{0, 1\}^m$. This results in $2^m - 1$ free parameters per class label to construct the distribution $\Pr(S = \bar{s}_{ij} \mid y_{ij})$.

Conditional Independence Assumption To avoid this exponential blowup, we can assume that the verifiers provide *conditionally independent* outputs:

$$P(S \mid y) = \prod_{k=1}^{m} P(S_k \mid y),$$

which reduces the number of parameters from $O(2^m)$ to $O(m)$ and enables efficient inference, under the assumption that each verifier provides unique information about the correctness of a response.

Then, Equation (7) simplifies to a Naive Bayes-style estimator:

$$\begin{aligned} \Pr\big(y_{ij} = 1 \mid S = \bar{s}_{ij}\big) &= \frac{\Pr(S_1 = \bar{s}_{ij1}, \ldots, S_m = \bar{s}_{ijm} | y_{ij} = 1) \Pr(y_{ij} = 1)}{\Pr(S = \bar{s}_{ij})} \\ &= \frac{\Pr(y_{ij} = 1) \prod_{k=1}^{m} \Pr\big(S_k = \bar{s}_{ijk} \mid y_{ij} = 1\big)}{\sum_{y' \in \{0,1\}} \Pr(y_{ij} = y') \prod_{k=1}^{m} \Pr\big(S_k = \bar{s}_{ijk} \mid y_{ij} = y'\big)} \end{aligned} \tag{8}$$

The parameters in Equation (8) include:

- The prior probability of correctness $\pi = \Pr(y_{ij} = 1)$.
- The verifier-specific conditional likelihoods $P(S_k \mid y_{ij})$.

B.1.1. Parameter Estimation

Supervised Setting When ground-truth labels y_{ij} are available, parameter estimation reduces to computing empirical frequencies. We can estimate the prior as:

$$\hat{\pi} = \frac{1}{N} \sum_{i,j} \mathbf{1}\{y_{ij} = 1\}$$

For each verifier k, we could estimate:

$$\hat{w}_{k,1} = \frac{\sum_{i,j} \mathbf{1}\{y_{ij} = 1\} \cdot \mathbf{1}\{S_k = 1\}}{\sum_{i,j} \mathbf{1}\{y_{ij} = 1\}},$$
$$\hat{w}_{k,0} = \frac{\sum_{i,j} \mathbf{1}\{y_{ij} = 0\} \cdot \mathbf{1}\{S_k = 0\}}{\sum_{i,j} \mathbf{1}\{y_{ij} = 0\}}.$$

Weak Supervised Setting When a few labeled y_{ij} are available, we can use it to estimate π, but we still need to estimate the verifier accuracy parameters $w_{k,1}, w_{k,0}$ to compute $\prod_{k=1}^{m} \Pr(S_k = \bar{s}_{ijk} | y_{ij} = 1)$. Instead of using labeled data, we estimate accuracy parameters using moment matching. In particular, we match observable second moments of verifier outputs to the model-implied moments under conditional independence assumptions, based on an approach from (HazyResearch, 2018).

Pairwise Statistics. For each pair of verifiers k_1, k_2 and binary outputs $a, b \in \{0, 1\}$, we can express the joint probability of their outputs using the marginalization rule and the conditional independence assumption:

$$\begin{aligned} &\Pr(S_{k_1} = a, S_{k_2} = b) \\ &= \Pr(S_{k_1} = a | Y = 1) \Pr(S_{k_2} = a | Y = 1) \Pr(Y = 1) + \Pr(S_{k_1} = b | Y = 0) \Pr(S_{k_2} = b | Y = 0) \Pr(Y = 0) \end{aligned} \tag{9}$$

where the conditional distributions for verifier k are:

$$\Pr(S_k = a \mid y = 1) \;=\; \begin{cases} w_{k,1}, & a = 1, \\ 1 - w_{k,1}, & a = 0, \end{cases} \quad \Pr(S_k = a \mid y = 0) \;=\; \begin{cases} 1 - w_{k,0}, & a = 1, \\ w_{k,0}, & a = 0. \end{cases}$$

Marginal Statistics. Similarly, each verifier's marginal distribution can be written as:

$$\Pr(S_k = 1) = \Pr(S_k = 1 | Y = 1) \Pr(Y = 1) + \Pr(S_k = 1 | Y = 0) \Pr(Y = 0) \tag{10}$$

Note that this equation holds true regardless of the conditional independence assumption.

Estimation method

- Construct the second order moment matrix $O \in \mathbb{R}^{(2m) \times (2m)}$, where:

$$O_{2i-1:2i, 2i-1:2i} = \Pr(S_i = 0) \qquad 0 \tag{11}$$
$$0 \qquad \Pr(S_i = 1) \; \forall i \in [m]$$
$$O_{2i-1:2i, 2j-1:2j} = \Pr(S_i = 0, S_j = 0) \qquad \Pr(S_i = 0, S_j = 1) \tag{12}$$
$$\Pr(S_i = 1, S_j = 0) \qquad \Pr(S_i = 1, S_j = 1) \; \forall i \neq j \in [m] \tag{13}$$

- Construct the conditional probability matrix $\mu \in \mathbb{R}^{(2V) \times 2}$, where each row encodes:

$$\mu_{2k+a,\, b} \;=\; \Pr\big(S_k = a \mid y = b\big)$$

$$\mu = \begin{bmatrix} w_{k_1,0} & 1-w_{k_1,1} \\ 1-w_{k_1,0} & w_{k_1,1} \\ \cdots & \cdots \\ w_{k_m,0} & 1-w_{k_m,1} \\ 1-w_{k_m,0} & w_{k_m,1} \end{bmatrix}$$

- Label prior matrix $P \in \mathbb{R}^{2\times 2}$ is a diagonal matrix:

$$P = \begin{bmatrix} \Pr(y_{ij}=0) & 0 \\ 0 & \Pr(y_{ij}=1) \end{bmatrix}.$$

Then, equation 9 is equivalent to $O = \mu P \mu^\top$ on the entries off of the 2×2 block diagonal, and equation 10 is equivalent to $\text{diag}(0) = \mu P \not{\mu}^\top$. Therefore, we optimize the following loss to compute μ.

$$\text{minimize}_\mu \left\| O_{\text{off-diag}} - (\mu\, P\, \mu^T)_{\text{off-diag}} \right\|^2 + \left\| \text{diag}(O) - \mu\, P\, \mathbf{1}^T \right\|^2,$$

By solving via gradient-descent, we obtain estimates of the verifier accuracy parameters $\{w_{k,1}, w_{k,0}\}$.

B.1.2. Inference: computing response correctness probabilities

Once the accuracy parameters $\{w_{k,1}, w_{k,0}\}$ are estimated and $P(y_{ij}=1)$ is computed from a small labeled development dataset, we can compute posterior correctness probabilities for each response:

$$\Pr(y_{ij}=1|S=\bar{s}_{ij}) \propto \Pr(y_{ij}=1)\prod_{k=1}^{m}\Pr(S_k=\bar{s}_{ijk}|y_{ij}=1)$$
$$\Pr(y_{ij}=0|S=\bar{s}_{ij}) \propto \Pr(y_{ij}=0)\prod_{k=1}^{m}\Pr(S_k=\bar{s}_{ijk}|y_{ij}=0)$$

Normalizing these, we have a full posterior $P(y_{ij}=1|S=\bar{s}_{ij})$, which provides a score with which we can select a response for each query.

B.2. Adapting Weak Supervision to the Verification Setting

Given a set of verifiers, we elaborate on the design choices behind the weak supervision model described in Section 4. In particular, we describe challenges around normalization, binarization and filtering out low-quality verifiers. In Section B.3, we describe WEAVER's approach to normalizing, binarizing, and filtering verifiers.

B.2.1. Normalization

Verifier outputs often differ substantially in scale, range, and distribution. For instance, some verifiers output unbounded real-valued scores (e.g., log-likelihoods), while others output normalized probabilities or learned regression values. Some standard losses under this framework include ranking losses, binary classification losses and regression losses. Some examples include:

Ranking losses

- **Pairwise logistic loss:** $\mathcal{L}(s_1, s_0) = \log(1+\exp(-(s_1-s_0)))$
- **Bradley–Terry loss:** $\mathcal{L}(s_1, s_0) = \log(1+\exp(s_0-s_1))$
- **Triplet margin loss:** $\mathcal{L}(s_1, s_0) = \max(0, 1-(s_1-s_0))$

Binary classification losses

- **Logistic loss:** $\mathcal{L}(s, y) = \log(1+\exp(-y's)), \quad y' = 2y-1$
- **Hinge loss:** $\mathcal{L}(s, y) = \max(0, 1-y's)$

Regression losses

- **Squared loss:** $\mathcal{L}(s, y) = (s - y)^2$

- **Huber loss:** $\mathcal{L}_\delta(s, y) = \begin{cases} \frac{1}{2}(s-y)^2 & \text{if } |s-y| \leq \delta \\ \delta(|s-y| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$

For regression losses, it is essential that the model output s lies in the same range as the label $y \in [0, 1]$.

To combine out of the box verifiers, which may be trained under different constraints, verifier scores must be comparable. We note that standard losses imposes different but related invariance assumptions:

- **Ranking and binary classification losses** are invariant to positive affine transformations of the form $s \mapsto \alpha s + \beta$, where $\alpha > 0$. This means that the relative ordering or decision boundary between scores is preserved even if scores differ in scale or offset.

- **Regression losses**, in contrast, directly penalize deviations between scores and targets, so the absolute scale of scores matters. Because many verifier outputs are meant to approximate probabilities or correctness labels $y \in [0, 1]$, it is essential to constrain scores to the same interval.

B.2.2. BINARIZATION

The weak supervision algorithm described in the prior section requires binary verifier outputs. This is naturally suited for judge-style verifiers—such as language models prompted to answer yes/no questions—which output discrete $0, 1$ labels. However, many verifiers, especially reward models, emit continuous scores and often vary in scale and calibration. This raises a key design question: should we input these scores to the weak supervision model as-is, or should we binarize them?

Using continuous scores retains fine-grained information about the confidence of each verifier. This can improve ranking-based performance metrics such as AUC and may allow the weak supervision model to better resolve disagreements among verifiers. However, it introduces challenges when combining signals across verifiers with inconsistent calibration or scale: a score of 0.8 may have different meanings for different verifiers. As seen in Figure Figure 6 different verifiers exhibit different score distributions even when evaluating the same set of responses. Some verifiers are sharply bimodal, others skew heavily toward low or high scores, and some produce nearly flat or noisy distributions.

To address this, we evaluate several binarization strategies that convert continuous verifier scores into discrete labels. Figure 7 compares the AUC performance of a logistic regression model trained on verifier outputs across four binarization methods: no binarization (continuous scores), a fixed threshold at 0.5, class balance-based thresholds, and quantile binarization. We observe that while continuous scores can achieve strong AUC when sufficient training data is available, simple binarization strategies—especially those that account for score distribution skew—perform comparably and are more robust under limited supervision.

Figure 8 shows only modest differences across binarization strategies for selection accuracy. We note in highly imbalanced datasets, as in the case GPQA, simple quantile-based binarization performs particularly well, likely because it adjusts for the skewed distribution of scores, i.e. it discards ambiguous mid-range scores and retains only the most confident signals.

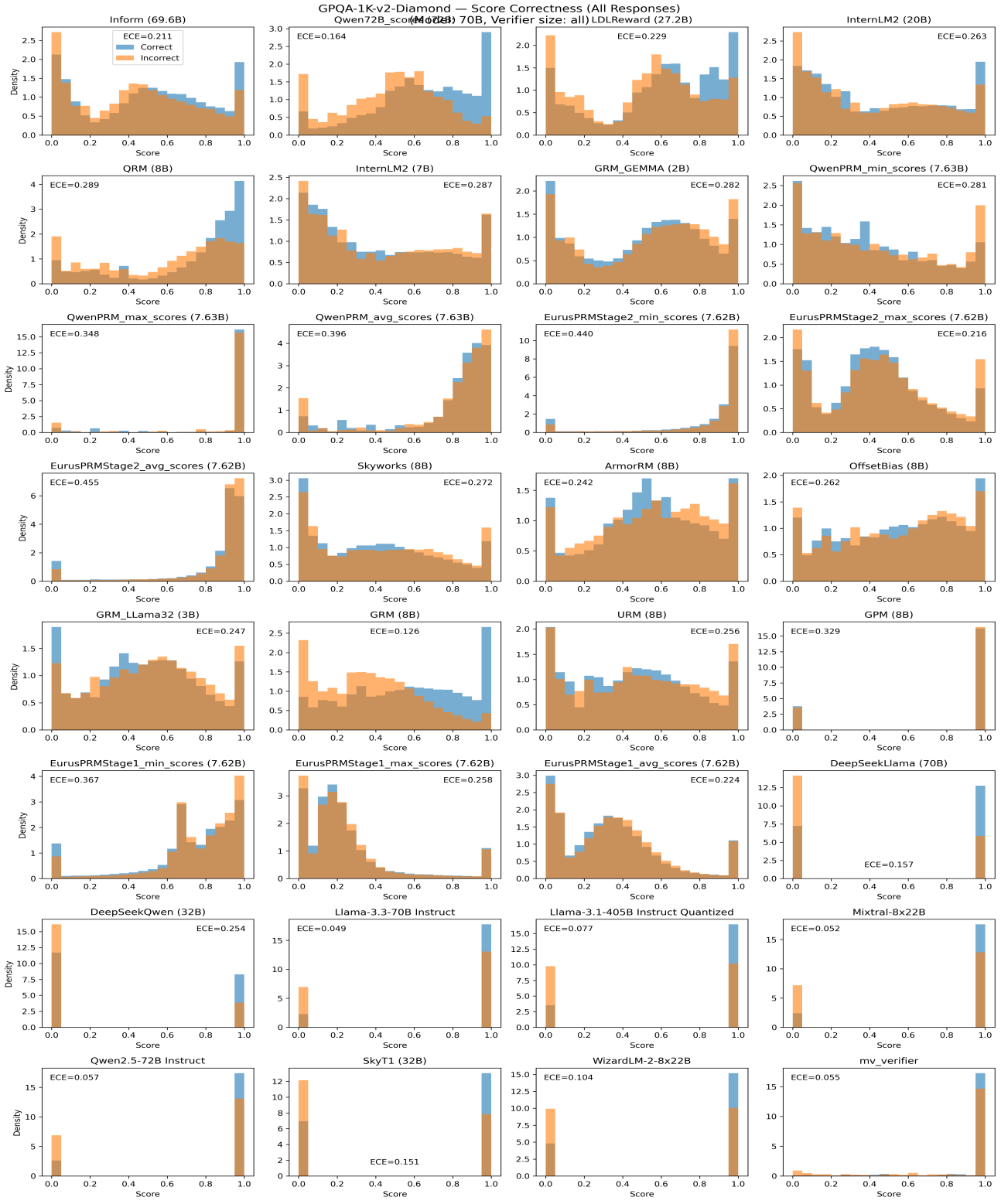

Figure 6: Accuracy of verifiers on the GPQA dataset. For each verifier, we compute the fraction of problems for which the top-ranked response (according to its score) is correct. While some verifiers consistently select high-quality answers, others perform near chance or worse, motivating the need to filter out low-quality verifiers before applying weak supervision.

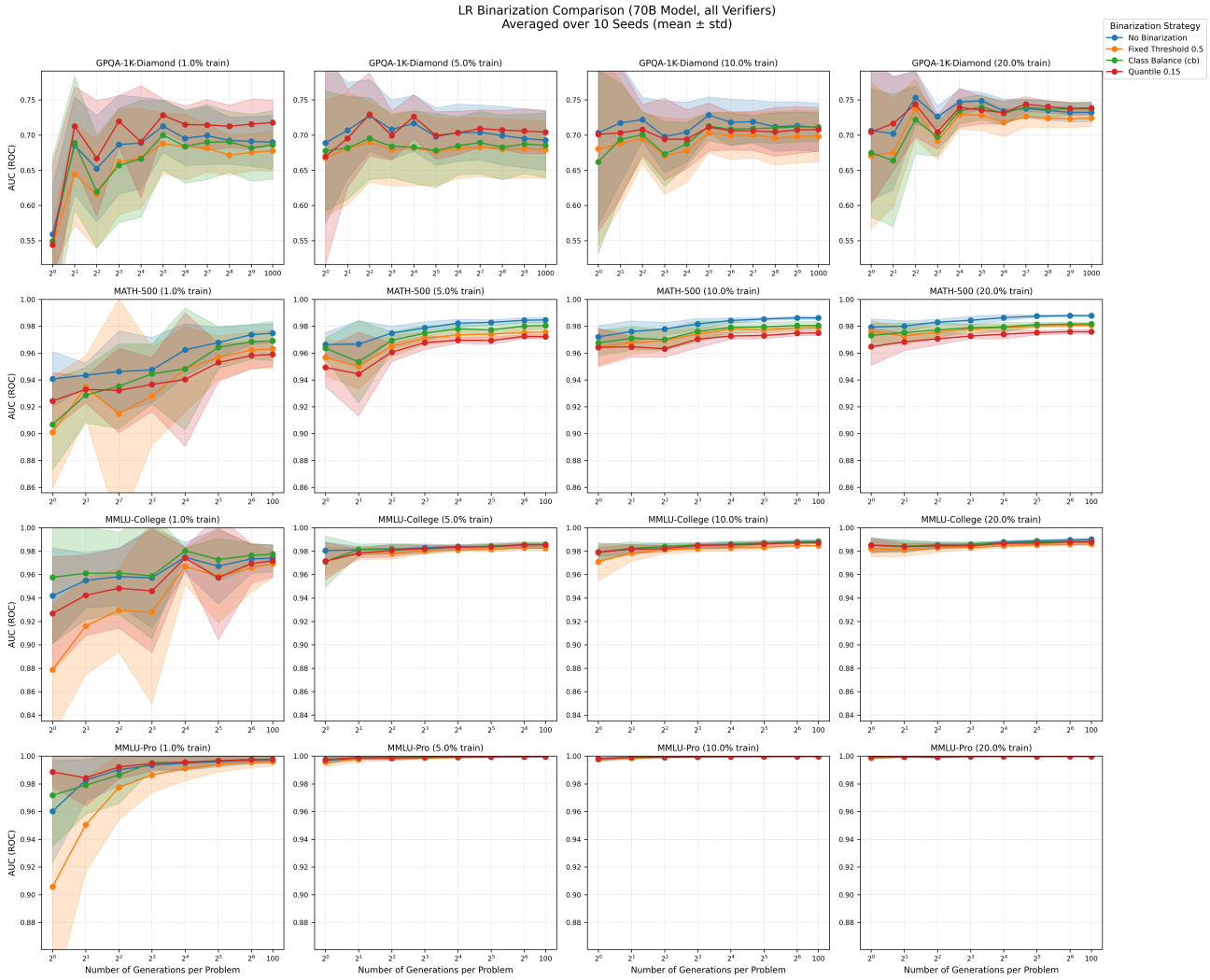

Figure 7: Same conventions as Figure 8 where the performance reported is the area under the curve of the ROC.

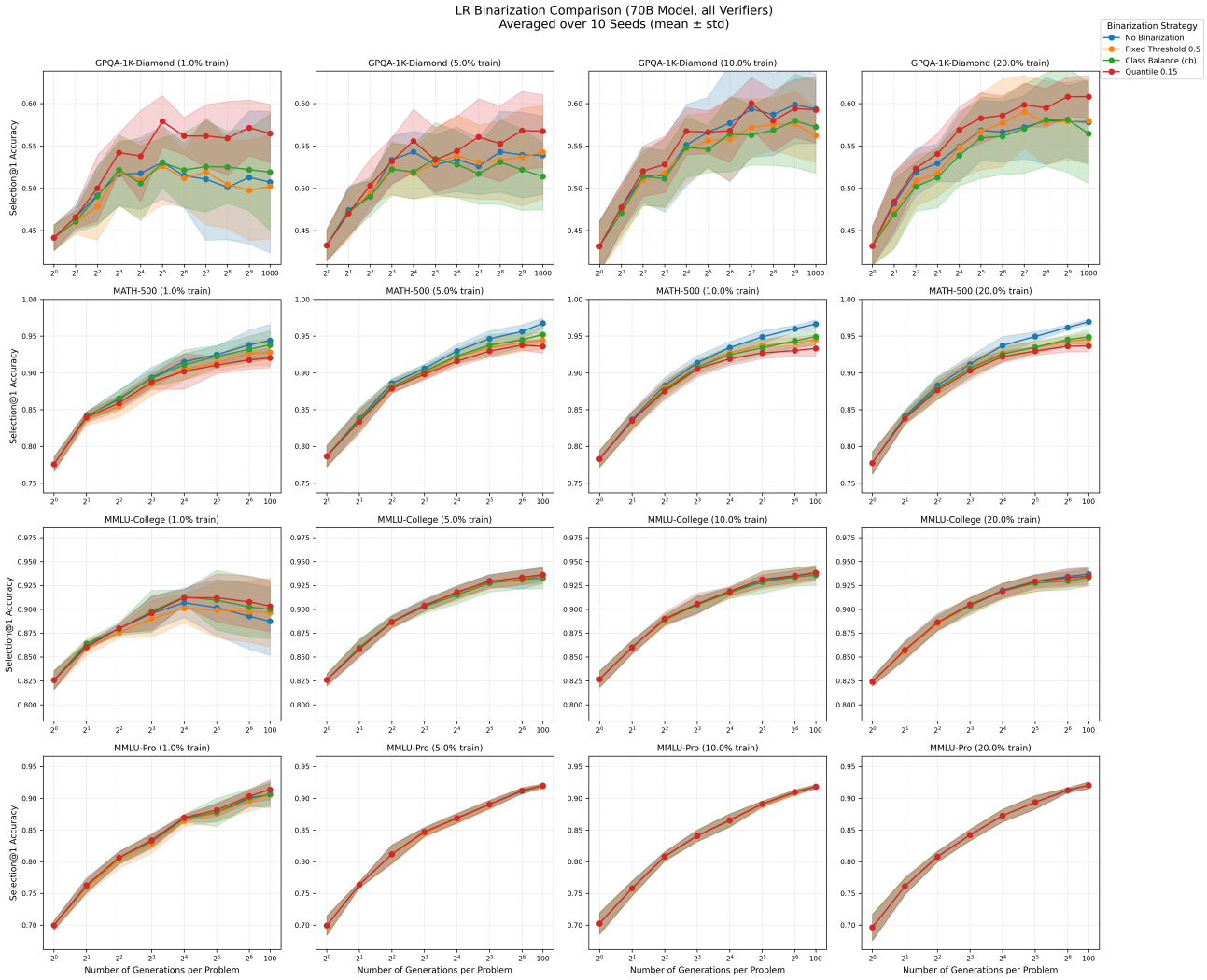

Figure 8: Performance for logistic regression models trained on the verifier outputs under different binarization strategies: (1) None: uses raw continuous scores without binarization, (2) Fixed Threshold: applies a uniform threshold across all data, (3) Class Balance: chooses a threshold per verifier so that the proportion of positive labels matches the true class distribution (4) Quantile: assigns positive labels to only the top 15% of scores, focusing on high-confidence predictions. Results are shown across four datasets (GPQA-Diamond, MATH-500, MMLU-College, MMLU-Pro) and four training fractions $(1\%, 5\%, 10\%, 20\%)$. For each seed, we use a random subset of problems as the training set and report the performance on the remaining problems. Curves report mean and standard deviation over multiple seeds. Performance is shown as a function of the number of generations per problem (x-axis, log scale).

.

B.2.3. Filtering out Low-Quality Verifiers

Verifiers with low accuracy or extreme marginals (e.g., near-constant outputs) not only degrade ensemble performance but also undermine the stability and identifiability of Weak Supervision algorithms, worsening the estimation error. *How do we discard verifiers that have low signal?*

- Skewed marginals: Consider a dataset where $\Pr(y = 1) \approx 0.5$ and we have a verifier with $\Pr(S_k = 1) \approx 0.99$. A skewed verifier with an extreme marginal (e.g., from naive thresholding for binarization) and near-constant outputs adds little information to the ensemble. It primarily increases noise in the objective in Eq. 6 and should thus be discarded. Yet, not at all verifiers that have extreme marginals add little signal; for instance, if instead $\Pr(y = 1) \approx 0.99$, a skewed

verifier could be highly accurate. Therefore, the definition of a low-quality verifier depends on the distribution of correct responses.

- Breaking symmetry in the WS objective: a common assumption of Weak Supervision is that a majority of the verifiers have better-than-random accuracy (Fu et al., 2020). Otherwise, there is a possibility that the WS algorithm can yield non-unique solutions; the terms in Eq. 6 are the joint probabilities over pairs of verifiers as well as their marginals, which do not uniquely determine if a verifier satisfies $w_{k,1}, w_{k,0} > 0.5$ or not. Therefore, it is critical to remove as many low-accuracy verifiers as possible to ensure that the estimation procedure converges to a unique solution.

B.3. Adaptation Method

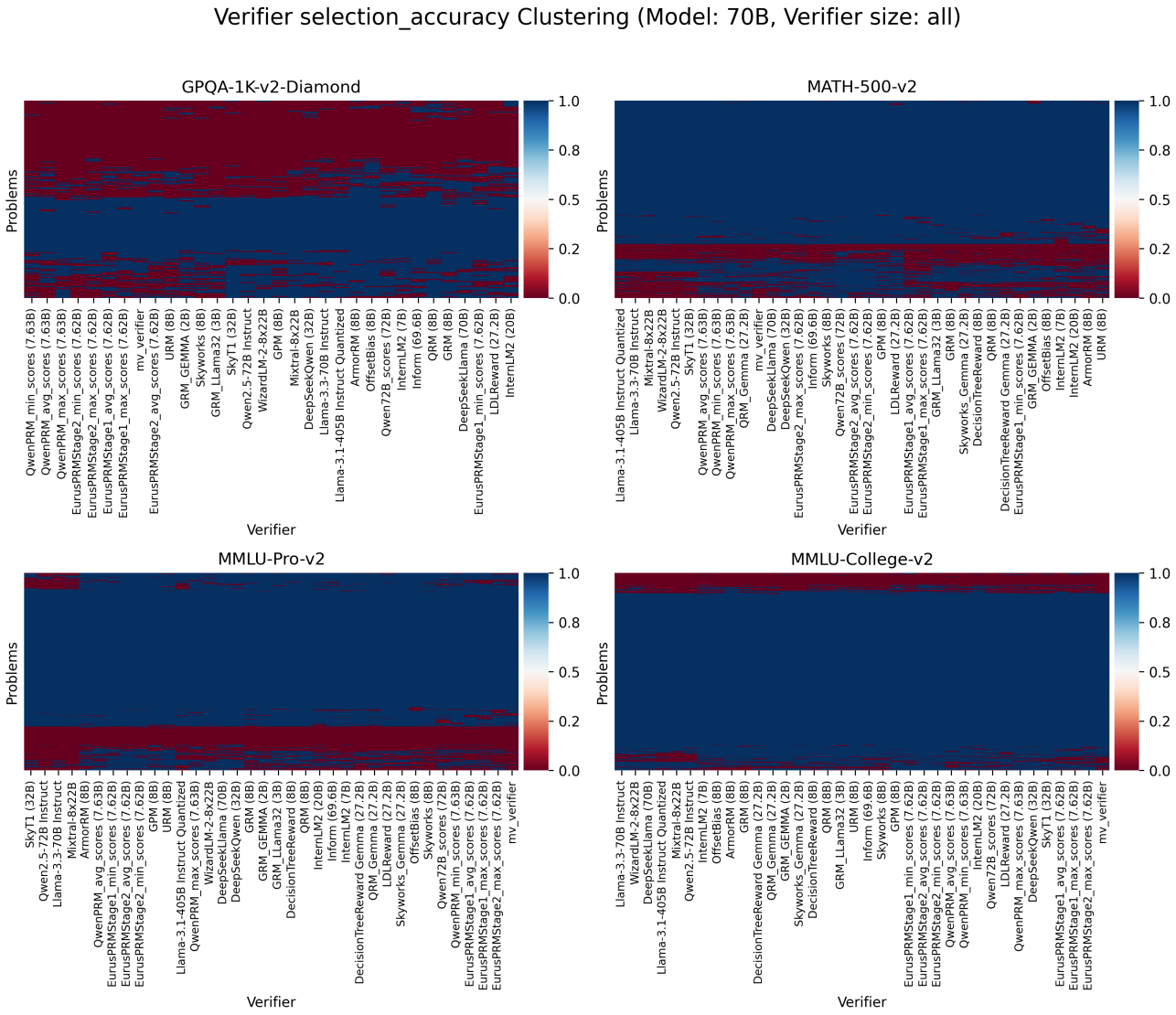

Figure 9: Selection accuracy for each verifier across problems and datasets.

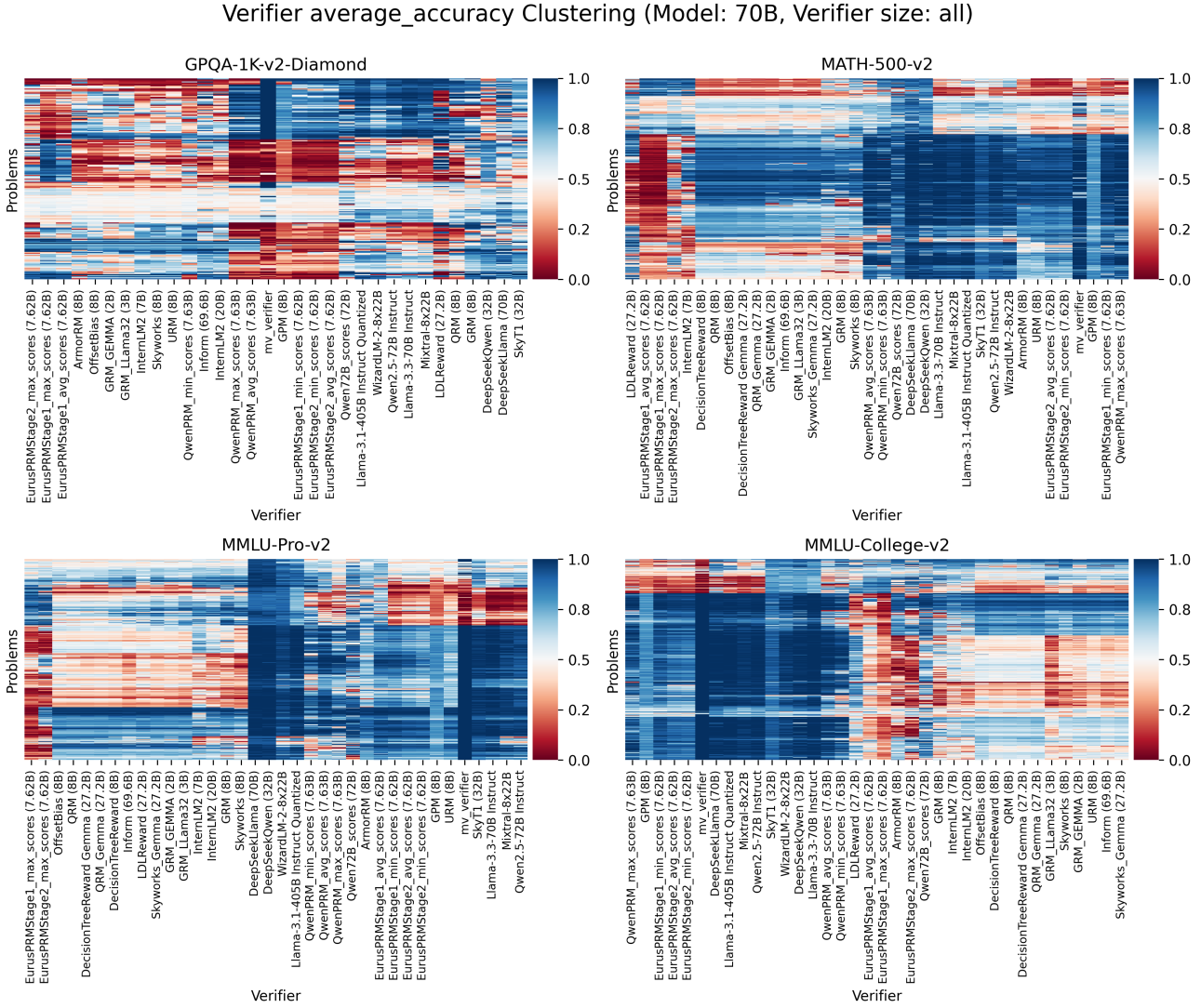

Figure 10: Average accuracy for each verifier across problems and datasets given Llama-70B responses scores.

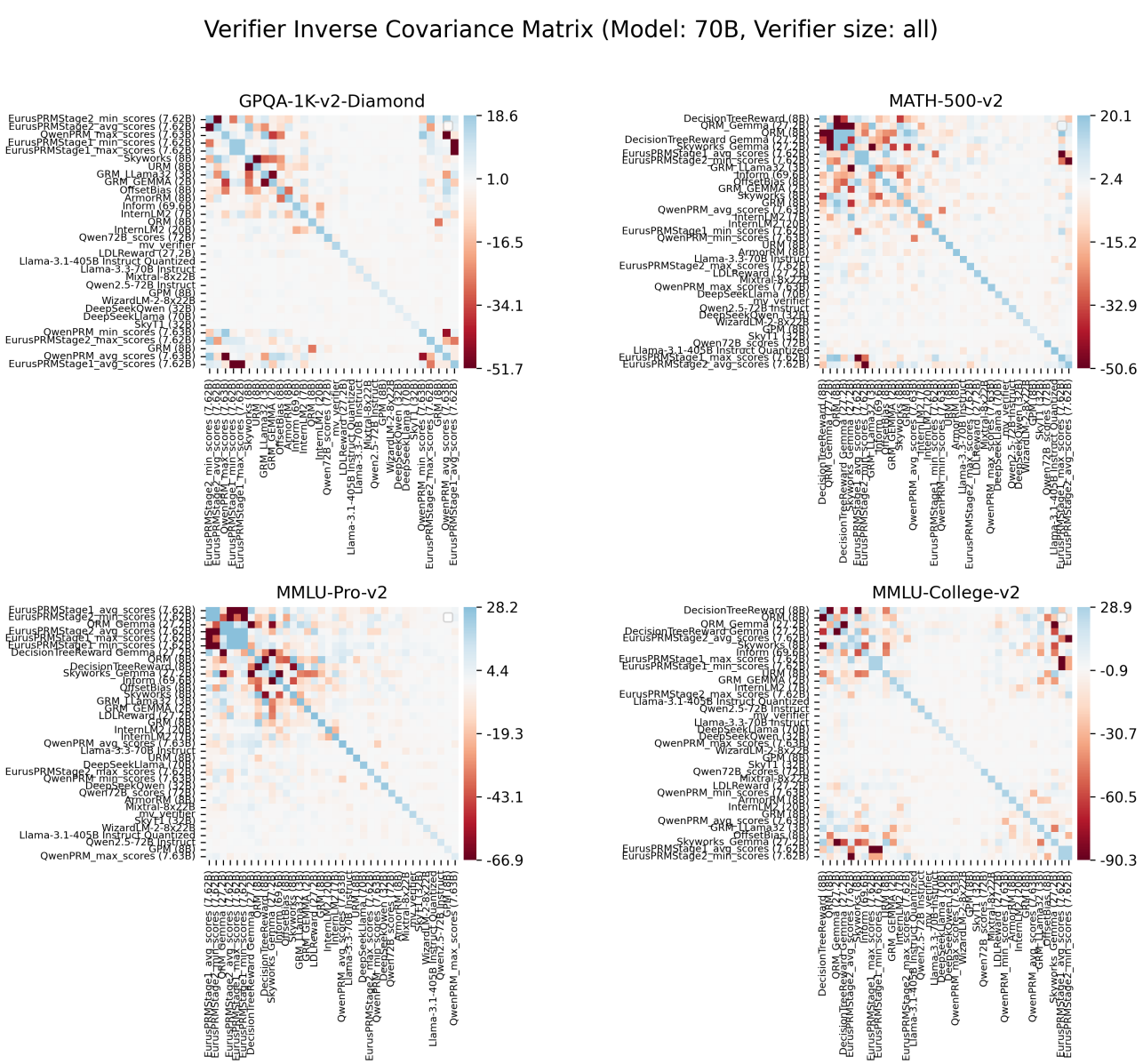

Figure 11: Inverse covariance matrix for each verifier across datasets given Llama-70B responses scores.

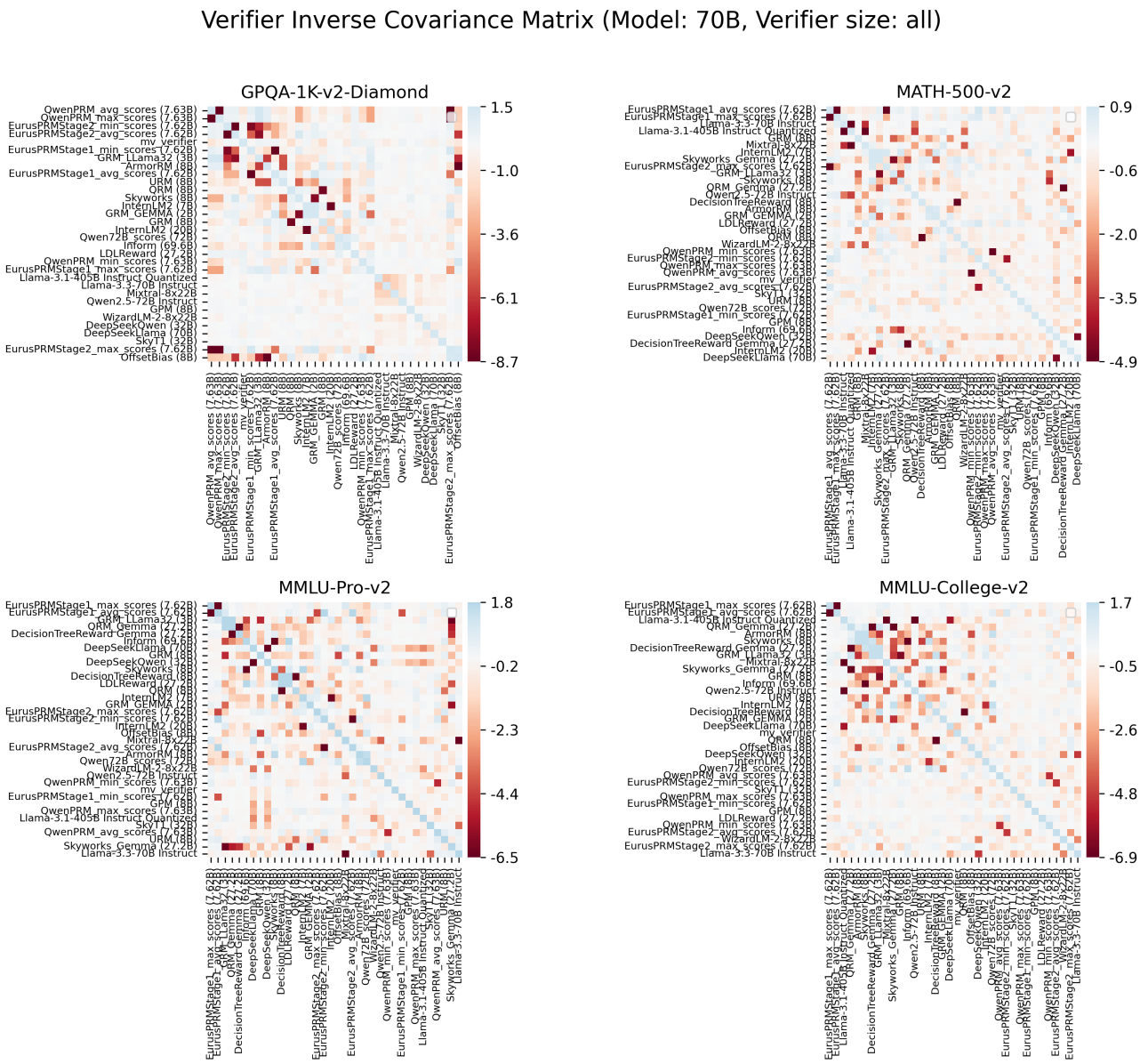

Figure 12: Inverse covariance matrix for each verifier across datasets given Llama-70B responses scores, after binarization

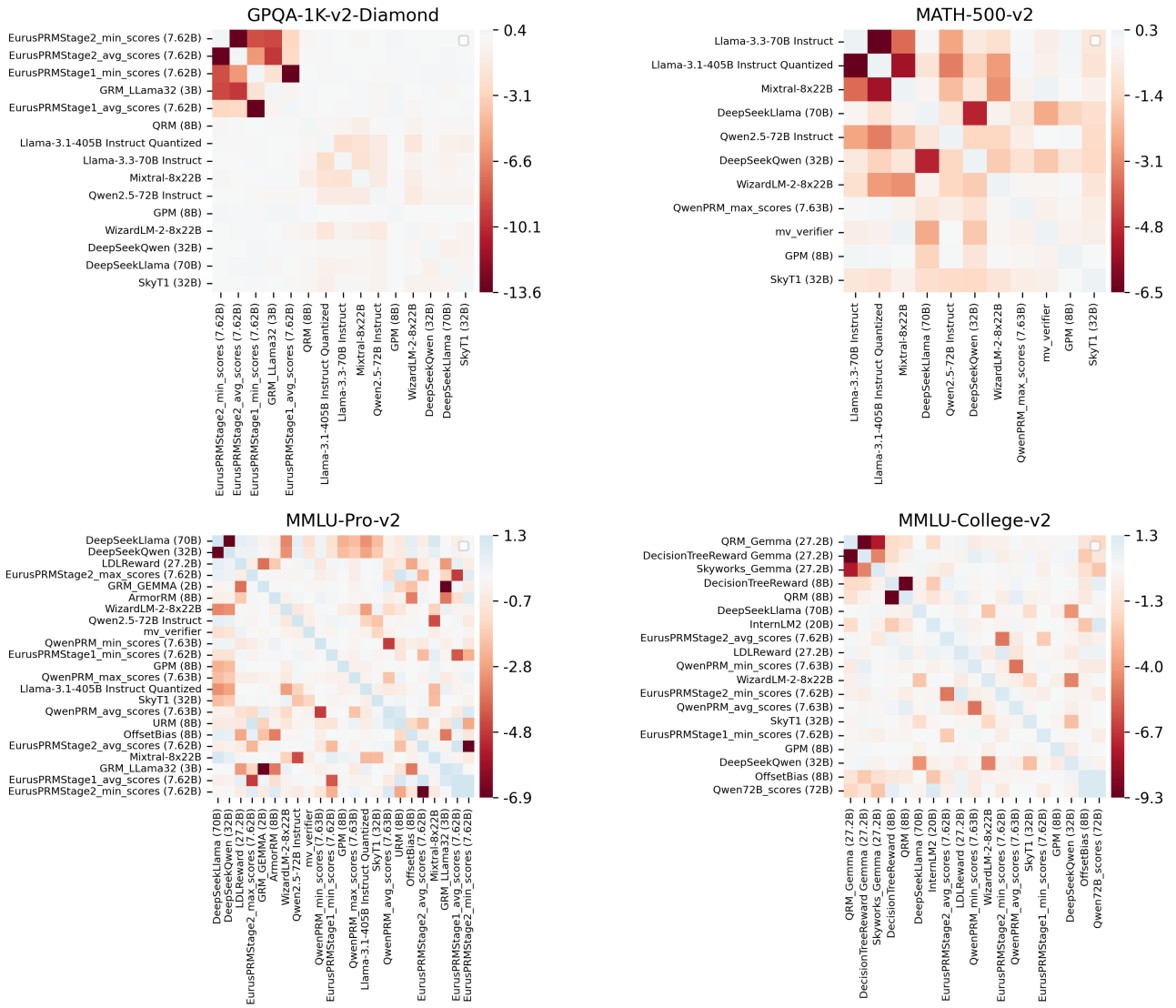

Figure 13: Inverse covariance matrix for each verifier across datasets given Llama-70B responses scores, after binarization and dropping.

We now describe our proposed method for normalization, binarization and filtering of verifiers, after which the Weak Supervision algorithm described in Appendix B.1.

1. **Normalization**: To make verifier outputs comparable, we apply min-max normalization to each verifier:

$$s' = \frac{s - \min(s)}{\max(s) - \min(s)} \in [0, 1].$$

This ensures that all scores lie within the same numerical range and preserves relative orderings. For regression-style verifiers, normalization aligns their outputs with the scale of the labels and avoids numerical instability due to unbounded score ranges. Without normalization, aggregation methods may become biased or ill-conditioned due to disproportionate score magnitudes.

Table 4: **Ablation of Development Set used for WEAVER Class Balance Estimation**: As our reward model threshold, we set it to the default of the 0.5.

Approach	Model Size	Dev Set Size	Benchmarks				
			MATH500	GPQA Diamond	MMLU	MMLU Pro	**Average**
WEAVER	70B	Naive Threshold (0.5 Threshold)	88.1%	52.0%	92.4%	83.5%	79.0%
WEAVER	70B	1%	90.4%	67.1%	91.1%	87.0%	84.5%
WEAVER	70B	5%	92.4%	72.7%	93.5%	90.4%	87.5%
WEAVER	70B	20%	92.4%	72.7%	93.5%	90.4%	87.5%
WEAVER	70B	100%	92.4%	72.7%	93.5%	90.4%	87.5%

Table 5: **Ablation of Verifier Selection Strategies for WEAVER**: Comparison of different strategies for dropping faulty verifiers by dataset using each verifier's marginal probability.

Approach	Model Size	Verifier Selection	Benchmarks				
			MATH500	GPQA Diamond	MMLU	MMLU Pro	**Average**
WEAVER	70B	No Dropped Verifiers	90.4%	52.0%	91.1%	84.2%	79.4%
WEAVER	70B	Low Marginals Dropped (Mostly Negative Verifiers)	**93.4%**	60.6%	91.7%	**91.0%**	84.2%
WEAVER	70B	High Marginals Dropped (Mostly Positive Verifiers)	83.4%	69.7%	87.9%	78.4%	79.9%
WEAVER	70B	Extreme Marginals Dropped (Mostly Positive or Negative)	90.8%	**72.7%**	**92.4%**	85.0%	**85.2%**

2. **Binarization**: We use a small amount of labeled samples (which we already are using to compute $\Pr(y = 1)$) to determine a threshold for converting continuous verifier outputs to binary outputs. Figure 12 illustrates the precision matrix of the verifier scores after binarization. It shows a damping of large off-diagonal dependencies and improved condition numbers. Table 4 shows that with only 5 to 10 labeled queries from benchmark development sets (which $\leq 1\%$ of the evaluation set), we can estimate binarization thresholds that bolster performance by averages of 8.4% for the 70B models when compared to binary splitting along the median score of the dataset.

3. **Filtering out low-quality verifiers**: To mitigate the impact of low-quality verifiers, we prune verifiers with extreme marginal behavior, depending on the class balance. For datasets with estimated class balance between 20% and 80%, we filter out verifiers with positive rates outside this range. If a dataset has fewer than 20% positive samples overall, we remove verifiers that predict positives more than 80% of the time. Conversely, for datasets with more than 80% positives, we drop verifiers that predict positives less than 20% of the time.

   Figure 13 illustrates the precision matrix of the verifier scores after both binarization and dropping low-signal or redundant verifiers. Compared to Figure 12, it shows further attenuation of off-diagonal structure. Illustrating that dropping contributes substantially to decorrelating the verifier set, which can improve identifiability and numerical stability for downstream weak supervision. As shown in Table 5, verifier pruning leads to 12.5% performance improvement for the 70B model setting.

Table 6: **Ablation of Adaptive Threshold Dev Set Size for WEAVER**.

Approach	Model Size	Adaptive Threshold Dev Set Size	Benchmarks				
			MATH500	GPQA Diamond	MMLU	MMLU Pro	Average
WEAVER	70B	0.01	92.4%	72.7%	93.5%	90.4%	87.5%
WEAVER	70B	0.05	92.4%	72.7%	93.5%	90.4%	87.5%
WEAVER	70B	0.2	92.4%	72.7%	93.5%	90.4%	87.5%
WEAVER	70B	1.0	92.4%	72.7%	93.5%	90.4%	87.5%

Table 7: **Performance Comparison Between Continuous and Discrete Logistic Regression**: For supervised fine-tuning on the verifier scores, the continuous model consistently outperforms the discrete variant across all datasets by avoiding the lossy conversion of floats to binary votes required for the discrete variant.

Continuous vs. Discrete Logistic Regression Performance (%)					
Method	**Dataset**				
	MATH 500	GPQA	MMLU College	MMLU Pro	BBH
Discrete LR	93.1	74.3	87.5	87.1	90.1
Continuous LR	**97.2**	**78.1**	**90.4**	**92.0**	**96.5**
Improvement	+4.1	+3.8	+2.9	+4.9	+6.4

Table 8: **Number of Unique Extracted Answers vs. Positive**: Negative Sample Ratio per Query - Correlations for Llama 3.1 Instruct Models

Llama 3.1 8B Instruct Correlation Metrics			
Dataset	**Metric Type**		
	Pearson	Spearman	Kendall's Tau
MATH 500	-0.676	-0.745	-0.565
GPQA	-0.312	-0.117	-0.096
MMLU College	-0.595	-0.700	-0.591
MMLU Pro	-0.590	-0.555	-0.425
BBH	-0.365	-0.386	-0.300

Llama 3.1 70B Instruct Correlation Metrics			
Dataset	**Metric Type**		
	Pearson	Spearman	Kendall's Tau
MATH 500	-0.631	-0.842	-0.709
GPQA	-0.148	-0.093	-0.089
MMLU College	-0.551	-0.862	-0.769
MMLU Pro	-0.446	-0.693	-0.585
BBH	-0.268	-0.594	-0.474

B.4. Exploration: Clustering by Difficulty to Improve WEAVER

Weak verifiers often behave inconsistently across the difficulty spectrum of input queries. For instance, most verifiers may have very high accuracy on easy queries and low accuracy on more difficult queries. Figure 10 and Figure 9 illustrate the average and selection accuracy of each verifier across multiple problems and datasets. This confirms that there is significant variation in verifier performance at the per-query level. To capture this heterogeneity, we explore clustering queries by difficulty and fitting one WEAVER's weak supervision model per cluster, independently.

We define *query difficulty* as the empirical ratio of correct to incorrect generations for each query. Using this as a proxy for problem hardness, we partition each dataset into evenly sized clusters along the difficulty distribution and learn separate weak supervision models for each cluster. This is done in an oracle setting, where difficulty is computed using ground-truth correctness, but no label information is used when training the cluster-specific verifier models.

This approach is adaptive in two senses: (1) we adapt the weak supervision model to the difficulty class of the query, and (2) we adapt the threshold of each reward model independently per cluster to better reflect local verifier behavior. For each cluster, we perform a grid search over reward model thresholds ranging from 0.05 to 0.95 in increments of 0.05 (19 values total), selecting the threshold that maximizes accuracy on a held-out development set. We experiment with clustering the queries into between 1 and 5 difficulty levels per dataset, using the oracle difficulty distribution to divide the queries into equally sized bins. While the clustering in our study uses oracle difficulty, future work could explore unsupervised

approximations or semi-supervised approaches using a small labeled subset (e.g., 10%) to estimate difficulty distributions.

In Tables 9 and 10,l, we analyze how difficulty-aware clustering and threshold adaptation affect WEAVER's performance at different model scales.

- **70B model:** At this scale, we find that optimizing a single reward model threshold captures most of the verification signal, with clustering yielding only marginal gains ($\sim$1%). This suggests that verifier behavior is relatively stable across query difficulties at higher model capacities.

- **8B model:** In contrast, the 8B setting exhibits a larger gain (4.8%) from clustering and adaptive thresholding. We attribute this to three key factors:

  1. **Higher verifier variance:** As shown in Table 16, verifier quality fluctuates more across queries in the 8B setting, making cluster-specific models more beneficial.

  2. **Fewer positive generations:** Table 13 shows that the 8B generator produces fewer correct answers overall, increasing difficulty heterogeneity.

  3. **Larger generation-verification gap:** The Pass@1-to-Pass@K gap is more pronounced for 8B (e.g., 49.8% to 99.2% on MATH500), indicating greater room for selection-based improvements.

Table 9 confirms that increasing cluster count improves accuracy for the 8B model but often degrades it for the 70B model. These results suggest that difficulty-aware modeling is especially useful when verifier behavior is unstable and when the generation model produces sparse correct candidates.

Further gains from per-model thresholding. Finally, in Table 11, we introduce a finer-grained tuning strategy where each reward model receives its own threshold, rather than using a single global threshold per cluster. This approach provides modest but consistent improvements for the 70B model (e.g., +0.5% on GPQA Diamond, +0.8% on MMLU Pro), and more substantial boosts for the 8B model across all datasets (+1.6% to +2.8%). These gains highlight the value of adapting verifier aggregation strategies not just by query difficulty but also by verifier-specific behavior, especially at smaller model scales where noise is more pronounced.

Table 9: **Performance of WEAVER with Different Clusters Counts**: Utilizes Llama 3.1 70B Instruct Generations with the verifier threshold optimized prior to clustering. We create the clusters based on query difficulty: *the ratio of correct to incorrect generations for each queries*. We create cluster in evenly sized chunks from the distribution of each task.

Clusters for WEAVER Dataset					
Dataset	**Cluster Count**				
	1	2	3	4	5
MATH 500	**93.4**	87.6	83.8	82.8	81.2
GPQA	**66.4**	**66.4**	**66.4**	**66.4**	**66.4**
MMLU College	**94.9**	91.7	90.1	89.6	89.8
MMLU Pro	88.4	**90.2**	87.1	84.6	79.8
Average	**85.8**	84.0	81.9	80.9	79.3

Table 10: **Optimizing Clusters and Adaptive Thresholds for WEAVER**. We report selection performance across different evaluation strategies using weak supervision, with and without difficulty-based clustering and threshold tuning. Clustering is based on oracle query difficulty. Thresholds are selected via grid search from 0.05 to 0.95 in increments of 0.05.

Performance Across Different Evaluation Methods with Llama 3.1 70B Instruct			
Dataset	**Clustering / Adaptive Threshold**		
	No Clusters / 0.5 Threshold	Best Found by Search	Pass@K
MATH 500	93.4%	95.2%	98.6%
GPQA Diamond	72.4%	74.1%	81.0%
MMLU College	94.9%	95.1%	96.0%
MMLU Pro	90.2%	90.2%	92.0%
Average	87.7%	88.7%	91.9%

Performance Across Different Evaluation Methods with Llama 3.1 8B Instruct			
Dataset	**Clustering / Adaptive Threshold**		
	No Clusters / 0.5 Threshold	Best Found by Search	Pass@K
MATH 500	80.0%	84.3%	99.2%
GPQA Diamond	47.1%	52.7%	95.2%
MMLU College	85.7%	89.9%	98.5%
MMLU Pro	67.2%	72.3%	96.8%
Average	70.0%	74.8%	97.4%

Table 11: **Optimizing Clusters and Per-Model Adaptive Thresholds for WEAVER**. In this setting, each reward model receives its own optimized threshold (rather than a global threshold per cluster). Thresholds are selected via grid search from 0.05 to 0.95 in steps of 0.05. Clustering is still based on oracle query difficulty. This finer-grained tuning yields small improvements for 70B models and more substantial gains for 8B models, especially where verifier accuracy is highly variable.

Performance Across Evaluation Methods with Llama 3.1 70B Instruct (Per-Model Thresholds)			
Dataset	**Clustering / Adaptive Threshold**		
	No Clusters / 0.5 Threshold	Best Found by Search	Pass@K
MATH 500	93.4%	95.2%	98.6%
GPQA Diamond	72.4%	74.6%	81.0%
MMLU College	94.9%	95.1%	96.0%
MMLU Pro	90.2%	91.0%	92.0%
Average	87.7%	89.0%	91.9%

Performance Across Evaluation Methods with Llama 3.1 8B Instruct (Per-Model Thresholds)			
Dataset	**Clustering / Adaptive Threshold**		
	No Clusters / 0.5 Threshold	Best Found by Search	Pass@K
MATH 500	80.0%	86.5%	99.2%
GPQA Diamond	47.1%	55.5%	95.2%
MMLU College	85.7%	91.5%	98.5%
MMLU Pro	67.2%	74.5%	96.8%
Average	70.0%	77.0%	97.4%

C. Experiments

C.1. Models and Datasets

Benchmarks: We evaluate our models with several benchmarks for instruction-following, reasoning, mathematics, and coding: MATH500 (Hendrycks et al., 2021), GPQA (Rein et al., 2024), MMLU (Hendrycks et al., 2021), MMLU Pro (Wang et al., 2024c), and BBH (Suzgun et al., 2022). We provide an overview of each dataset in Table 12. For MMLU, we selected the college-level questions for evaluation: biology, chemistry, physics, mathematics, computer science, and medicine. For MMLU Pro, we take a random sample of 500 queries out of the 12K queries available. For BBH, we take four tasks from the dataset of 6K queries available: Penguins in a Table, Causal Judgement, Logical Deduction (Five Objects), and Tracking Shuffled Objects (Five Objects).

Models: We evaluate candidate generations using a range of *weak verifiers*—models with imperfect but better-than-random accuracy. Our verification system $\mathcal{V}$ includes two primary classes of weak verifiers: *Reward Models* and *LM Judges*.

- **Reward Models**: A *reward model* (RM) is a trained language model that assigns a scalar score to candidate responses based on how well they align with human preferences (Lambert et al., 2024; Song et al., 2025a). Given a query and a candidate response, the RM outputs a value $V_{ij} \in [0, 1]$ representing the estimated quality of candidate j according to criteria such as correctness, helpfulness, and safety.
  - Examples of reward models include those from the RewardBench leaderboard (Lambert et al., 2024), such as INF-ORM (Minghao Yang, 2024), QRM Gemma (Dorka, 2024), and Skywork Reward (Liu et al., 2024). We also include *process reward models* (PRMs), which score the reasoning process itself—emphasizing step-by-step logic and coherence—rather than just the final answer (Cui et al., 2025; Yuan et al., 2024).

- For our study, we selected the top-20 reward models from RewardBench and the top-20 process reward models from Process Reward Bench (Song et al., 2025a) at both 8B and 70B parameter scales. We exclude any RM or PRM that fails to provide a positive learning signal—i.e., those whose rankings perform no better than random selection on benchmark train sets (Appendix C.1). The diverse training objectives and datasets used for these reward models introduce systematic biases that affect their verification capabilities (Lambert et al., 2024; Song et al., 2025a), with different loss functions—including Bradley-Terry loss for pairwise preferences (Bradley & Terry, 1952), margin loss for fixed score differences (Rosset et al., 2003), and pairwise ranking loss for relative ordering (Cao et al., 2007).
- Previous work has noted that it is nontrivial to combine the outputs of reward models and judges as they provide logits and binary decision rules (Verga et al., 2024; Xu et al., 2024). Instead, we find that we can normalize all RM scores to the range $[0, 1]$ using robust percentiles: the bottom 5th percentile is mapped to 0 and the top 95th percentile to 1. For models that provide multiple scoring dimensions (e.g., ArmoRM (Wang et al., 2024b)), we use only their primary output.

- **LM Judges**: An *LM judge* is a language model used to assess the correctness of a candidate response by generating a binary verdict: $V_{ij} \in \{0, 1\}$, where 1 indicates that the response is judged correct. These models typically apply chain-of-thought (CoT) reasoning to arrive at their decisions (Wei et al., 2023). Each LM judge takes a query and a response as input and outputs a single binary verdict.
  - We use well-known chat models from ChatBotArena (Chiang et al., 2024) as LM judges, which are known for their general-purpose reasoning capabilities. To ensure consistency and determinism, we use greedy decoding (temperature $T = 0$) when generating judgments.

Table 12: **Benchmark Overview**: Evaluation configurations for AlpacaEval 2.0, Arena-Hard-Auto, AIMO, MATH500, GPQA, MMLU, MMLU Pro, and Big-Bench Hard (BBH).

Benchmark	Dataset Size	Scoring Type	Metric	License
MATH500	500	Ground Truth	Pass@1	Apache 2.0
GPQA	646	Ground Truth	Pass@1	CC BY 4.0
MMLU College	719	Ground Truth	Pass@1	MIT
MMLU Pro	500	Ground Truth	Pass@1	MIT

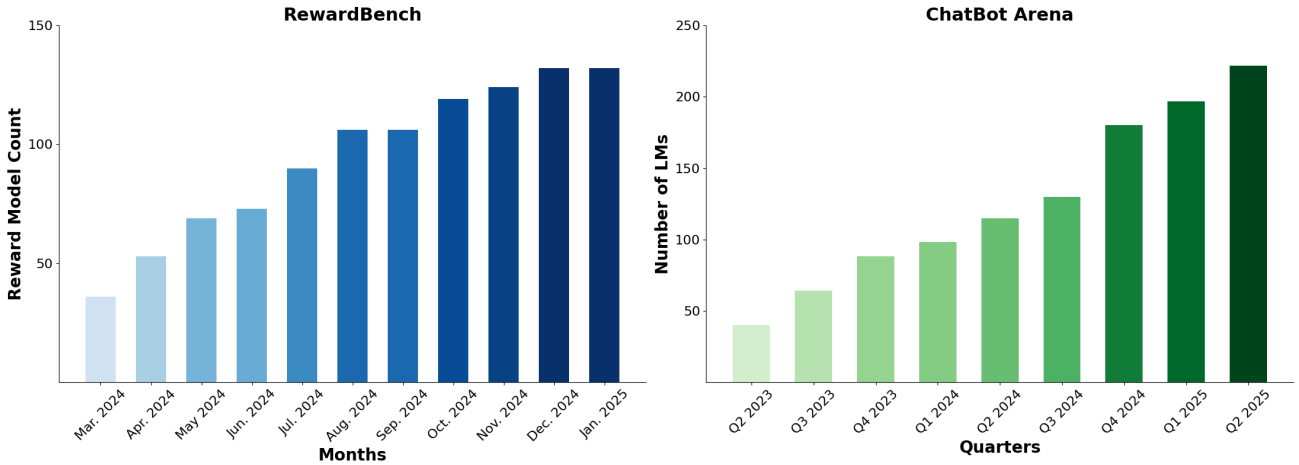

Figure 14: **Growth of Open-Source RMs and LMs**: As more and more RMs and LM judges become available, the need for better selection and utilization strategies for these models at test-time continues to grow.

Table 13: **Distribution of Generation Accuracy for Llama 3.1 8B Instruct.** Each row shows the fraction of queries falling into deciles of correctness—i.e., the proportion of correct generations out of 100 samples per query. The final column reports the overall correct-to-incorrect (**C/I**) ratio for each dataset. These distributions highlight variation in query difficulty and motivate our clustering-by-difficulty approach in Appendix B.4.

Llama 3.1 8B Instruct **Distribution of Correct/Incorrect Generations**											
Dataset	**Percentage of Total Dataset**										**C/I Ratio**
	0.0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0	
AIMO	72.2%	7.8%	3.3%	7.8%	2.2%	3.3%	2.2%	0.0%	1.1%	0.0%	10.3%
MATH 500	12.2%	11.4%	11.6%	8.2%	6.4%	8.0%	8.4%	6.60%	11.2%	15.0%	49.9%
GPQA	28.5%	21.2%	16.1%	7.4%	7.3%	5.3%	3.7%	3.3%	2.9%	2.6%	28.3%
MMLU College	7.8%	7.6%	7.1%	6.5%	6.7%	5.3%	7.2%	6.3%	8.6%	22.9%	64.1%
MMLU Pro	22.4%	9.8%	10.6%	5.4%	6.4%	7.2%	4.0%	7.6%	7.6%	15.2%	46.6%
BBH	3.2%	6.7%	10.3%	8.3%	10.3%	14.8%	11.6%	10.7%	10.4%	12.1%	56.9%

Table 14: **Distribution of Generation Accuracy for Llama 3.1 70B Instruct.** Each row shows the fraction of queries falling into deciles of correctness—i.e., the proportion of correct generations out of 100 samples per query. The final column reports the overall correct-to-incorrect (**C/I**) ratio for each dataset. These distributions highlight variation in query difficulty and motivate our clustering-by-difficulty approach in Appendix B.4.

Llama 3.1 70B Instruct **Distribution of Correct/Incorrect Generations**											
Dataset	**Percentage of Total Dataset**										**C/I Ratio**
	0.0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0	
MATH 500	7.0%	4.2%	3.8%	2.0%	2.2%	3.8%	4.2%	4.0%	7.8%	61.0%	78.0%
GPQA	36.8%	5.6%	5.1%	3.7%	6.2%	4.3%	4.5%	4.8%	8.0%	20.9%	42.9%
MMLU College	8.1%	3.2%	1.8%	2.2%	1.5%	1.7%	1.8%	3.2%	2.4%	74.1%	82.6%
MMLU Pro	16.4%	4.2%	1.8%	3.4%	3.0%	3.0%	3.2%	4.8%	6.0%	54.2%	69.9%

Table 16: **WEAVER Verifier Accuracies and Score Correlations.** We report the range of individual verifier accuracies and the average pairwise Pearson correlation between verifier scores. Each verifier's outputs are flattened across all query–candidate pairs, and correlations are computed across all $\binom{m}{2}$ verifier pairs. Lower correlation indicates greater diversity in how verifiers score responses, which supports the effectiveness of ensembling under WEAVER.

Metric	Model Size	Benchmarks			
		MATH500	GPQA	MMLU Pro	**Average**
Verifier Accuracy Range	8B	34.2%	40.7%	36.4%	**37.1%**
Avg. Score Correlation	8B	0.0253	0.0349	0.0312	**0.0305**
Verifier Accuracy Range	70B	27.4%	29.0%	31.6%	**29.3%**
Avg. Score Correlation	70B	0.0211	0.0372	0.0240	**0.0274**

Table 15: **Models Tested for WEAVER.**

	Model	Source Code	Parameter Count	License	Loss Function
LM Judges	Llama-3.1-70B-Instruct	Open-Source	70B	Llama 3.1 Community	Cross-Entropy Loss
	Llama-3.1-405B-Instruct	Open-Source	405B	Llama 3.1 Community	Cross-Entropy Loss
	Llama-3.3-70B-Instruct	Open-Source	70B	Llama 3.1 Community	Cross-Entropy Loss
	Meta-Llama-3.1-405B-Instruct-quantized.w8a16	Open-Source	405B	Llama 3.1 Community	Cross-Entropy Loss
	DeepSeek LLM 67B Chat	Open-Source	67B	DeepSeek License	Cross-Entropy Loss
	DeepSeekLlama70B	Open-Source	70B	DeepSeek License	Cross-Entropy Loss
	DeepSeekQwen32B	Open-Source	32B	DeepSeek License	Cross-Entropy Loss
	DeepSeekLlama8B	Open-Source	8B	DeepSeek License	Cross-Entropy Loss
	DeepSeekQwen7B	Open-Source	7B	DeepSeek License	Cross-Entropy Loss
	Qwen2 72B Instruct	Open-Source	72B	Tongyi Qianwen	Cross-Entropy Loss
	Qwen2.5-72B-Instruct	Open-Source	72B	Tongyi Qianwen	Cross-Entropy Loss
	Qwen/Qwen2.5-72B-Instruct	Open-Source	72B	Tongyi Qianwen	Cross-Entropy Loss
	QwQ-32B	Open-Source	32B	Apache 2.0	Cross-Entropy Loss
	Qwen1.5 110B Chat	Open-Source	110B	Tongyi Qianwen	Cross-Entropy Loss
	Qwen1.5 72B Chat	Open-Source	72B	Tongyi Qianwen	Cross-Entropy Loss
	Qwen-2.5-7B-Instruct	Open-Source	7B	Tongyi Qianwen	Cross-Entropy Loss
	Qwen-2.5-Math-7B-Instruct	Open-Source	7B	Tongyi Qianwen	Cross-Entropy Loss
	Mixtral 8x22B v0.1	Open-Source	176B	Apache 2.0	Cross-Entropy Loss
	Mixtral-8x22B-Instruct-v0.1	Open-Source	176B	Apache 2.0	Cross-Entropy Loss
	WizardLM 8x22B	Open-Source	176B	Apache 2.0	Cross-Entropy Loss
	WizardLM-2-8x22B	Open-Source	176B	Apache 2.0	Cross-Entropy Loss
	dbrx-instruct	Open-Source	132B	Databricks Open Model	Cross-Entropy Loss
	SkyT1	Open-Source	32B	Apache 2.0	Cross-Entropy Loss
RMs (8B and below)	GRM-Llama3-8B-rewardmodel-ft	Open-Source	8B	MIT	Pairwise Ranking Loss
	GRM-Llama3.2-3B-rewardmodel-ft	Open-Source	3B	Apache 2.0	Pairwise Ranking Loss
	GRM-Gemma2-2B-rewardmodel-ft	Open-Source	2B	Apache 2.0	Pairwise Ranking Loss
	Skywork-Reward-Llama-3.1-8B-v0.2	Open-Source	8B	Skywork License	Pairwise Ranking Loss
	QRM-Llama3.1-8B-v2	Open-Source	8B	MIT	Quantile Regression Loss
	URM-LLaMa-3.1-8B	Open-Source	8B	Skywork License	Uncertainty-Aware Loss
	GPM-Llama-3.1-8B	Open-Source	8B	MIT	Pairwise Ranking Loss
	Llama-3-OffsetBias-RM-8B	Open-Source	8B	Llama 3.1 Community	Pairwise Ranking Loss
	ArmoRM-Llama3-8B-v0.1	Open-Source	8B	Llama 3.1 Community	Pairwise Ranking Loss
	Qwen2.5-Math-PRM-7B	Open-Source	7B	Tongyi Qianwen	Cross-Entropy Loss
	EurusPRM-Stage1	Open-Source	7B	Apache 2.0	Cross-Entropy Loss
	EurusPRM-Stage2	Open-Source	7B	Apache 2.0	Cross-Entropy Loss
	internlm2-7b-reward	Open-Source	7B	Apache 2.0	Pairwise Ranking Loss
	Decision-Tree-Reward-Llama-3.1-8B	Open-Source	8B	Skywork License	Decision Tree Loss
RMs (27B–72B)	Skywork-Reward-Gemma-2-27B-v0.2	Open-Source	27B	Skywork License	Pairwise Ranking Loss
	QRM-Gemma-2-27B	Open-Source	27B	MIT	Quantile Regression Loss
	INF-ORM-Llama3.1-70B	Open-Source	70B	Custom License	Binary Cross-Entropy Loss
	Qwen2.5-Math-RM-72B	Open-Source	72B	Tongyi Qianwen	Cross-Entropy Loss
	Qwen2.5-Math-PRM-72B	Open-Source	72B	Tongyi Qianwen	Cross-Entropy Loss
	internlm2-20b-reward	Open-Source	20B	Apache 2.0	Pairwise Ranking Loss
	Decision-Tree-Reward-Gemma-2-27B	Open-Source	27B	Skywork License	Pairwise Ranking Loss

C.2. Verification Baselines

C.2.1. Verifier-Free Approaches

First Sample (Pass@1): This baseline uses only the first generated response without any verification or selection mechanism. It represents the standard approach where models generate a single response and provides a lower bound for performance comparison. This method does not scale test-time compute or employ verification.

Majority Voting: A verifier-free approach that generates multiple candidate responses and selects the most frequent final answer across all responses (Brown et al., 2024; Chen et al., 2024b; Snell et al., 2024). This method leverages repeated sampling but does not use verification models to assess response quality. Instead, it relies on the assumption that correct answers will appear more frequently than incorrect ones across multiple generations.

C.2.2. Alternative Verification Strategies

Naive Unweighted Aggregation: We consider three oracle configurations using the top-1, top-5, and top-10 verifiers (ranked by their agreement with ground-truth labels). Across all datasets, these oracle ensembles substantially outperform baselines. On average, the best-performing unweighted ensembles exceed first-sample performance by 20.3% and outperform majority voting by 15.0% (see Figure 2). For more difficult benchmarks such as GPQA and MMLU Pro, the top-5 and top-10 ensembles consistently outperform top-1, suggesting that verifier diversity is especially beneficial on challenging examples. However, these oracle ensembles rely on access to ground truth to rank verifiers, limiting their use in practice and motivating the need for learned, unsupervised weighting.

Naive Bayes: We implement a Naive Bayes classifier that models the probability of response correctness given verifier scores: $P(y_{ij} = 1|s_{ij1}, ..., s_{ijm}) = \frac{P(s_{ij1},...,s_{ijm}|y_{ij}=1)P(y_{ij}=1)}{P(s_{ij1},...,s_{ijm})}$. Under the conditional independence assumption, this factorizes as $P(s_{ij1}, ..., s_{ijm}|y_{ij} = 1) = \prod_{k=1}^{m} P(s_{ijk}|y_{ij} = 1)$. We estimate the parameters using labeled data from the development set. This approach provides a probabilistic framework for aggregating verifier outputs but requires labeled data for parameter estimation.

Logistic Regression: We train a logistic regression classifier where the input features are the verifier scores $[s_{ij1}, ..., s_{ijm}]$ and the output is the correctness of each response: $P(y_{ij} = 1|\mathbf{s}ij) = \sigma(\mathbf{w}^T \mathbf{s}ij + b)$, where σ is the sigmoid function. The weights $\mathbf{w}$ and bias b are learned using labeled training data. This supervised approach can capture more complex relationships between verifier outputs than naive averaging but requires substantial labeled data for effective training.

Multi-Agent Verification (MAV) (Lifshitz et al., 2025): This approach combines multiple "Aspect Verifiers" (AVs) - off-the-shelf LLMs prompted to verify specific aspects of candidate outputs through binary True/False approvals. Unlike reward models, AVs require no additional training and can be easily combined through voting mechanisms. The MAV framework uses BoN-MAV (Best-of-N with Multi-Agent Verification), which: **(1)** samples n candidate outputs from a generator LLM, **(2)** collects binary approvals from multiple aspect verifiers that vary across three dimensions (base LLM, aspect to verify, and verification strategy), and **(3)** selects the output with the most approvals. In our implementation, we use Llama 3.3 70B Instruct as the judge model rather than Gemini 1.5 Flash/Pro as used in the original paper.

Self-Verification (Zhao et al., 2025): This method implements a sophisticated sampling-based search approach where models verify their own responses through detailed natural language analysis. The approach goes beyond simple self-critique by using structured verification prompts that: **(1)** rewrite candidate responses in rigorous mathematical theorem-lemma-proof format, **(2)** systematically scan for errors through step-by-step analysis, and **(3)** compare responses to localize potential mistakes. The method leverages two key principles: comparing across responses provides signals about error locations (since models struggle with error recall but can identify errors when given their locations), and different output styles are optimal for different tasks (chain-of-thought for generation, rigorous mathematical format for verification). This approach differs from naive self-verification by using structured, multi-step verification protocols rather than simple correctness judgments.

Table 17: **Logistic Regression and Naive Bayes Performances across Datasets and Dev Set Sizes**

Dataset	Approach	Model Size	Dev Set Size				
			0.01	0.05	0.2	0.5	1.0
MATH-500	Logistic Regression	70B	70.5%	74.7%	81.4%	93.1%	97.2%
	Naive Bayes	70B	67.4%	78.1%	85.0%	89.2%	92.2%
GPQA Diamond	Logistic Regression	70B	55.9%	59.4%	69.8%	71.4%	72.9%
	Naive Bayes	70B	47.2%	49.2%	57.6%	62.1%	64.3%
MMLU Pro	Logistic Regression	70B	72.1%	81.0%	84.6%	86.0%	92.0%
	Naive Bayes	70B	60.2%	73.1%	73.1%	78.6%	78.6%

C.3. Scaling Trends of WEAVER

Scaling laws describe how performance metrics such as accuracy, sample efficiency or compute cost change as we scale controllable resources, i.e. the number of trials K, model capacity. (Kaplan et al., 2020) showed that, for fixed-parameter Transformer language models, the cross-entropy loss decreases as a power-law in both model size and data. This framework has since been extended to explore optimal tradeoffs between model and data scaling (Hoffmann et al., 2022b), as well as inference-time scaling with multiple samples (Chen et al., 2021; Brown et al., 2024).

First, we establish the power law scaling of the Pass@K rate. Assume the i-th problem has an unknown "difficulty" $p_i \in [0, 1]$, the probability that one response is correct. With K independent samples, the chance we get at least one correct response is

$$q_i(p_i, K) = 1 - (1 - p_i)^K \approx 1 - \exp(-p_i \cdot K) \quad \text{for small } p_i$$

Define the indicator variable:

$$X_i = \begin{cases} 1, & \text{if the i-th query is solved at least once (with probability } q_i\text{)}, \\ 0, & \text{otherwise,} \end{cases}$$

and let the total number of solved problems be $Y = \sum_{i=1}^{N} X_i$.

The expected coverage (Pass@K) is the expected fraction of problems solved after trying K times per problem:

$$\text{Pass@K} := \mathbb{E}[Y]/N = \frac{1}{N} \sum_{i=1}^{N} (1 - (1 - p_i)^K)$$

To model population-level variation in problem difficulty, we assume each problem's correctness probability p_i is drawn from a Beta distribution: $p_i \sim \text{Beta}(\alpha, \beta)$. This captures the idea that some problems are easier (high p_i) while others are harder (low p_i), with the overall distribution controlled by the shape parameters $\alpha,\ \beta$. Then, the fraction of problem that can be solved in K attempts follows,

$$\begin{aligned} \text{Pass@K} &= \mathbb{E}_{p \sim \text{Beta}(\alpha,\ \beta)}[1 - (1 - p)^K] \\ &= 1 - \mathbb{E}_{p \sim \text{Beta}(\alpha,\ \beta)}[(1 - p)^K] = 1 - \frac{B(\alpha,\ \beta + K)}{B(\alpha,\ \beta)} \end{aligned} \tag{14}$$

by the definition of the Beta function $B(\cdot, \cdot)$. Taking logarithm:

$$\log \text{Pass@K} = \log \left(1 - \frac{B(\alpha,\ \beta + K)}{B(\alpha,\ \beta)}\right) \approx -\frac{B(\alpha,\ \beta + K)}{B(\alpha,\ \beta)}$$

by $\log(1 - x) \approx -x$ when x is small, which holds for large K. Then, expressing the Beta function in terms of the Gamma function leads to:

$$\log \text{Pass@K} \approx -\frac{\Gamma(\beta + K)\Gamma(\alpha + \beta)}{\Gamma(\beta)\Gamma(\alpha + \beta + K)}$$

For large K, we can apply Stirling's approximation of the Gamma function $\log\Gamma(x) \approx x\log x - x + \frac{1}{2}\log(2\pi) + \frac{1}{2}\log x$:

$$\begin{aligned}\log[-\log \text{Pass@K}] &= \log\Gamma(\beta+K) + \log\Gamma(\alpha+\beta) - \log\Gamma(\beta) - \log\Gamma(\alpha+\beta+K) \\ &\approx (\beta+K)\log(\beta+K) - (\alpha+\beta+K)\log(\alpha+\beta+K) + \frac{1}{2}\log\left(\frac{\beta+K}{\alpha+\beta+K}\right) \\ &\approx (\beta+K)\log K - (\alpha+\beta+K)\log K + \text{const} \\ &= -\alpha\log K + \log\zeta\end{aligned}$$

when we retain the leading term. In turn, the log of the expected coverage follows a power law in K, scaling as:

$$\log \text{Pass@K} = -\exp\left(-\alpha\ \log K + \log\zeta\right) = -\zeta K^{-\alpha} \tag{15}$$

Verifier Success Modeling Now suppose we pass the K candidates through a scoring model ("verifier") which selects the top-scoring answer. The verification process succeeds if (i) at least one correct answer was generated and (ii) the verifier ranks a correct answer highest.

$$\text{Selection@1}(K) := \mathbb{P}[\text{top-scoring response is correct}] \tag{16}$$

Assume the verifier assigns scores such that correct responses are drawn from a score distribution f_1, and the incorrect responses from a distribution f_0. Let $s^{(1)} = \{s_j : y_j = 1\}$ and $s^{(0)} = \{s_j : y_j = 0\}$ denote the scores of correct and incorrect responses, respectively. Then a query is successfully verified if:

$$\text{Selection@1} = \mathbb{P}\left[\max s^{(1)} > \max s^{(0)}\right]$$

Our goal is to compute the probability that the maximum of c i.i.d draws from f_1 exceeds the maximum of $K-c$ draws from f_0.

To model the correctness of responses, we assume each query i has a latent correctness probability $p_i \sim \text{Beta}(\alpha,\beta)$, reflecting query-specific difficulty. Given p_i, each of the K responses is sampled independently as:

$$y_{ij} \sim \text{Bernoulli}(p_i), \quad j = 1,\ldots,K$$

This implies the number of correct responses follows a Binomial distribution:

$$C_i = \sum_{j=1}^{K} y_{ij} \sim \text{Binomial}(K, p_i)$$

assuming (1) conditional independence of responses given p_i, (2) identical correctness probabilities within a query, and (3) a fixed number of responses K.

Because the correctness probability p varies across queries, the dataset-level Selection@1 curve requires marginalizing over p:

$$\text{Selection@1}(K) = \mathbb{E}_{p\sim\text{Beta}(\alpha,\beta)}\left[\text{Selection@1}(K \mid p)\right]$$

Combined with the need to model max comparisons over verifier scores, it renders the exact calculation of Selection@1 analytically intractable.

To enable tractable, smooth modeling of Selection@1, we introduce the following parametric form:

$$\text{Selection@1}(K) \approx \exp(-\zeta K^{-\alpha}) \cdot \left(1 - (1-\pi)^{K^{\gamma}}\right) \tag{17}$$

- The **coverage term** $\exp(-\zeta K^{-\alpha})$ approximates the probability that at least one correct response is generated.
- The **verification term** $1-(1-\pi)^{K^{\gamma}}$ approximates the chance that the top-scoring response is correct, given that at least one correct response exists. The parameter γ controls whether verifier performance improves sublinearly or superlinearly with K. The parameter π represents the effective per-response probability that a correct response is successfully selected by the verifier, conditioned on the response being correct and included in the candidate set.

To obtain practical scaling trends, we fit parametric models in Equation (17) to the empirical averages computed from 5 independent runs for each value of K, across each dataset and verification strategy. Specifically, we use the L-BFGS-B algorithm to optimize a smooth approximation following (Hoffmann et al., 2022a). To ensure numerical stability and robustness to outliers or heavy-tailed noise in the observed selection accuracies, we minimize the Huber loss between the predicted values and the empirical means. The Huber loss behaves quadratically for small residuals and linearly for large ones, making it less sensitive to outliers than mean squared error (MSE) while maintaining smooth differentiability for gradient-based optimization. It is defined as,

$$L_\delta(r) = \begin{cases} \frac{1}{2}r^2 & \text{if } |r| \le \delta \\ \delta\left(|r| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases}$$

where $\delta > 0$ is a tunable threshold that controls the transition between the two regimes. We search over $\delta \in \{0.01, 0.05, 0.1, 0.25, 0.5\}$ to select the value that yields the best fit.

Additionally, we introduce floor and ceiling parameters to bound the predicted values and model saturation behavior. The floor accounts for the irreducible failure rate even at high K, while the ceiling models the upper bound on achievable performance (e.g., due to imperfect verifiers or ambiguous problems). The final fitted form is:

$$\text{Selection@1}(K) \approx \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta K^{-\alpha}) \cdot \left(1 - (1-\pi)^{K^\gamma}\right) \tag{18}$$

We can use an unbiased estimator to evaluate best-of-k selection accuracy when a fixed verifier is used to rank responses, as described in (Singhi et al., 2025b). However, in the case of WEAVER, the development set constitutes 1% of the data and is itself selected based on the value of K. In turn, the ranking of responses is no longer independent of K, introducing bias into the best-of-k estimate.As a result, we instead rely on Monte Carlo estimates to approximate best-of-k performance, sampling k responses multiple times and computing the average accuracy of the top-ranked output under the K-dependent verifier. We use an unbiased estimator for coverage, as described in (Chen et al., 2021).

Figure 15 and Figure 16 along with Table 18 illustrate how the different verification strategies scale with the number of generations and the fit to the parametric form in Equation (17). Each method exhibits characteristic scaling behavior that aligns with Equation (17). WEAVER demonstrates improved performance over naive ensembles and majority voting. The fitted parameters in Table 18 quantitatively capture these trends across datasets, providing evidence that the parametric from in Equation (18) closely model empirical outcomes. Figure 17 and Figure 18 along with Table 19 illustrate the predictive performance of the parametric form in Equation (18), showing that models fit on subsets of K can extrapolate to unseen values of K.

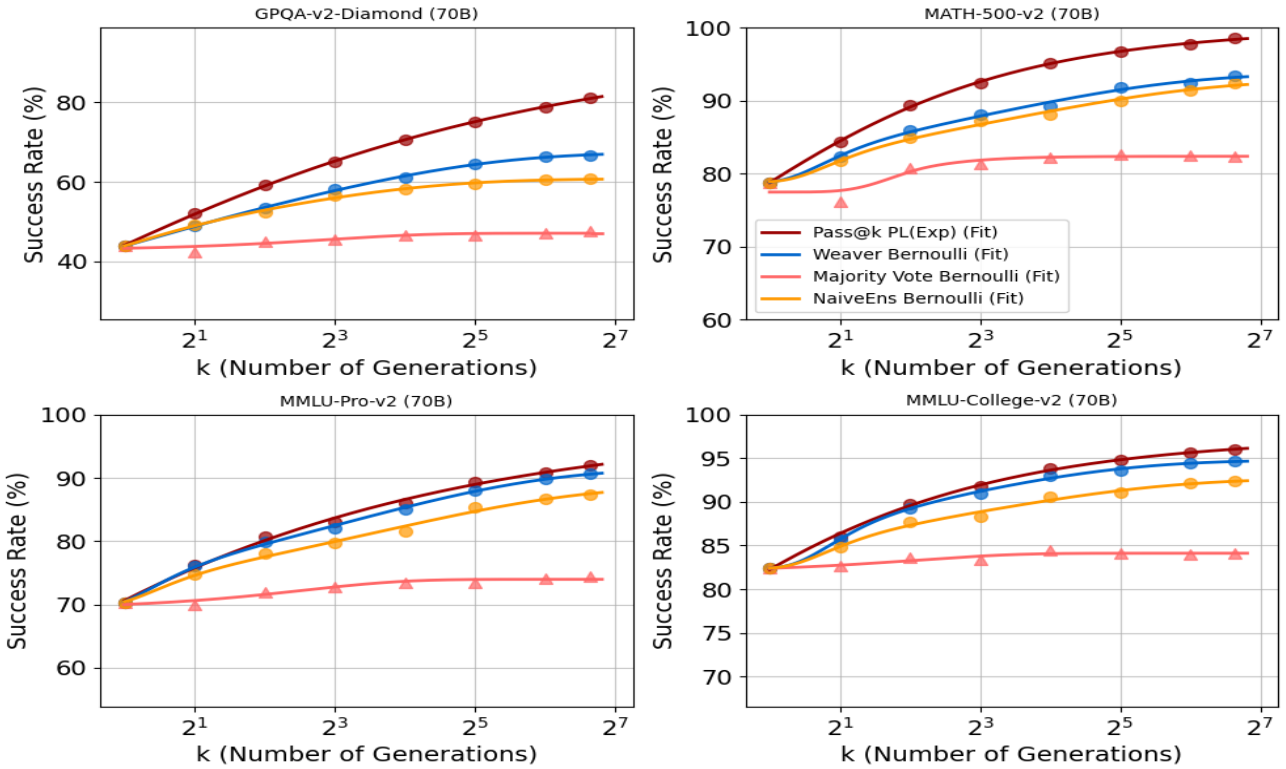

Figure 15: **WEAVER Scaling trend fit for 70B models**

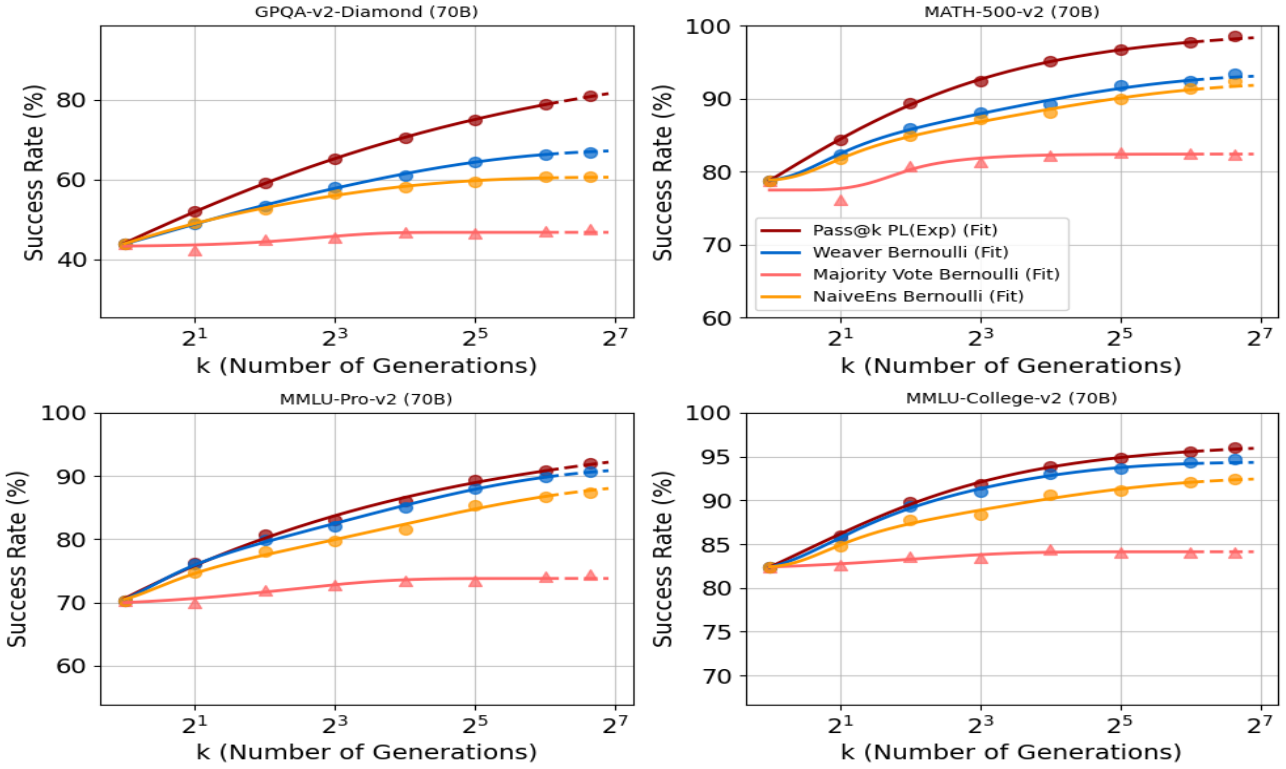

Figure 17: **WEAVER Scaling trend predicted for 70B models**

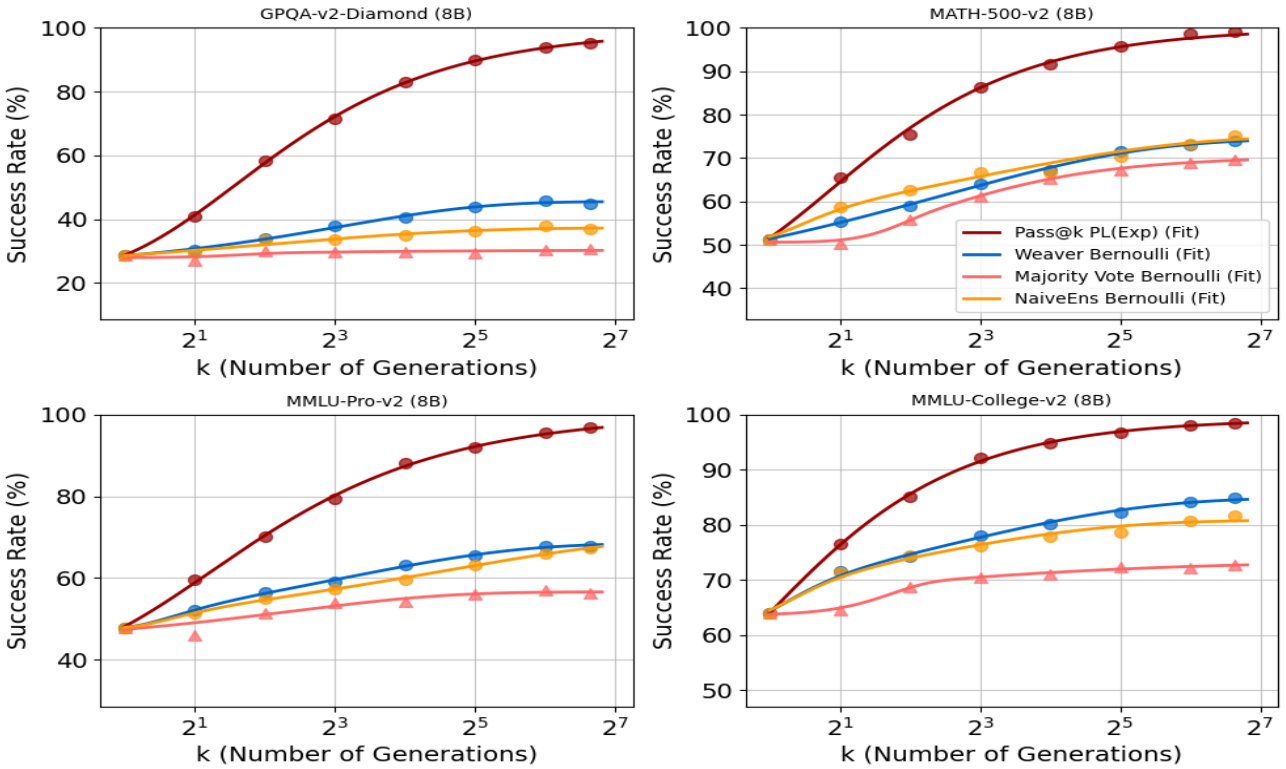

Figure 16: **WEAVER Scaling trend fit for 8B models.**

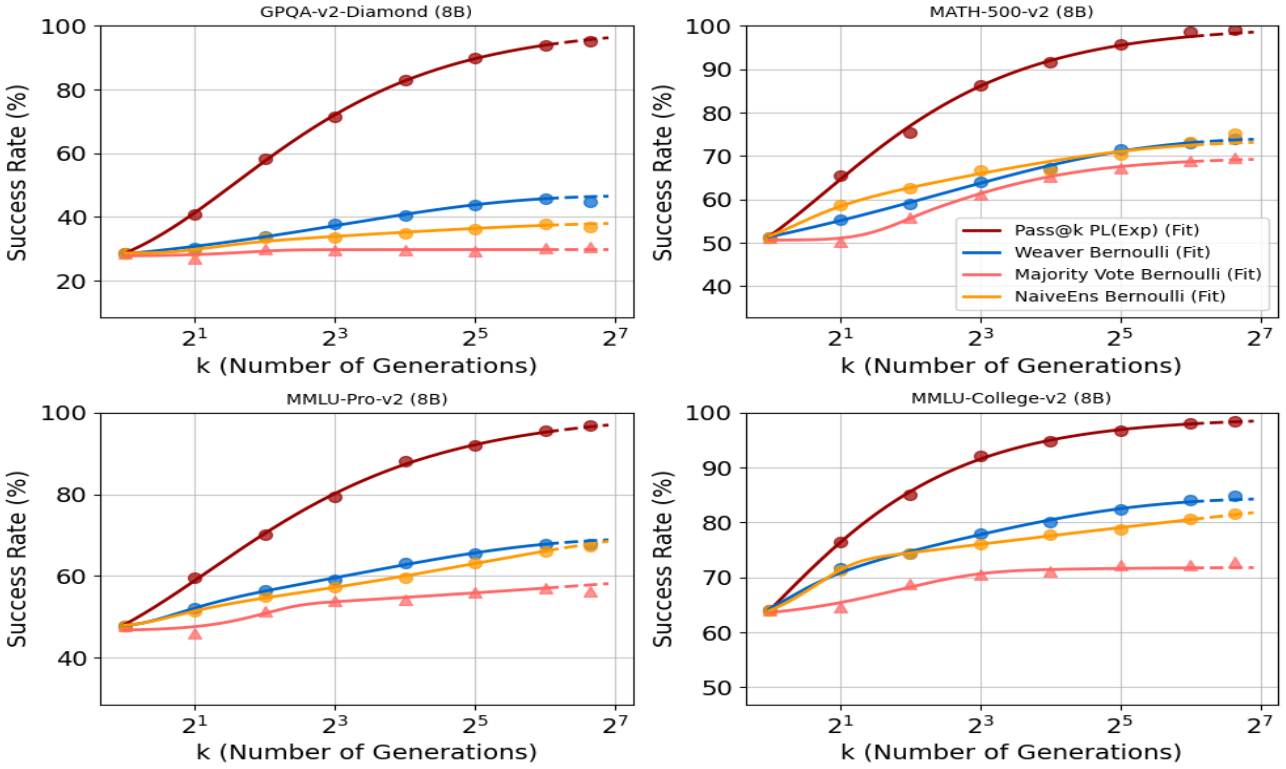

Figure 18: **WEAVER Scaling trend predicted for 8B models**

Table 18: Fitted parameters for Scaling Trends in Figure 15 and Figure 16.

Dataset	Approach	Equation	floor	ceil	ζ	α	π	γ	R2 fit	MSE fit	δ
GPQA-v2-Diamond (70B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.0000	0.9429	0.7603	0.3475	X	X	0.9999	0.0000	0.5000
GPQA-v2-Diamond (70B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.3958	0.6728	0.7320	1.5865	0.3250	0.5053	0.9994	0.0000	0.1000
GPQA-v2-Diamond (70B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4283	0.4710	0.0499	1.0000	0.1217	1.0091	0.8634	0.0000	0.0100
GPQA-v2-Diamond (70B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.3921	0.6071	0.6553	1.9147	0.4224	0.5000	0.9975	0.0000	0.2500
MATH-500-v2 (70B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.6262	1.0000	0.8394	0.6427	X	X	0.9994	0.0000	0.2500
MATH-500-v2 (70B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.7870	0.9371	3.3908	3.0000	0.2869	0.5000	0.9958	0.0000	0.1000
MATH-500-v2 (70B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.7747	0.8238	10.0000	2.1433	0.0885	2.4951	0.8655	0.0001	0.1000
MATH-500-v2 (70B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.7883	0.9282	4.0033	3.0000	0.2573	0.5000	0.9961	0.0000	0.0100
MMLU-Pro-v2 (70B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.0000	0.9828	0.3303	0.3465	X	X	0.9967	0.0000	0.2500
MMLU-Pro-v2 (70B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6912	0.9148	1.5284	3.0000	0.2764	0.5000	0.9987	0.0000	0.1000
MMLU-Pro-v2 (70B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6933	0.7399	0.0498	1.0001	0.1531	1.0123	0.9451	0.0000	0.0100
MMLU-Pro-v2 (70B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6969	0.8874	1.7834	3.0000	0.2403	0.5000	0.9944	0.0000	0.2500
MMLU-College-v2 (70B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.5924	0.9744	0.5071	0.5682	X	X	0.9982	0.0000	0.0100
MMLU-College-v2 (70B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.8234	0.9477	4.4129	3.0000	0.3622	0.5000	0.9987	0.0000	0.0100
MMLU-College-v2 (70B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.8197	0.8412	0.0498	1.0001	0.2057	1.0173	0.8912	0.0000	0.0100
MMLU-College-v2 (70B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.8235	0.9266	3.8766	3.0000	0.3012	0.5000	0.9925	0.0000	0.0500
GPQA-v2-Diamond (8B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.2262	0.9926	2.5454	0.8474	X	X	0.9996	0.0000	0.1000
GPQA-v2-Diamond (8B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.2463	0.4549	0.0534	1.0020	0.1953	0.7089	0.9948	0.0000	0.0500
GPQA-v2-Diamond (8B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.2783	0.3029	1.2431	0.6756	0.0630	2.5000	0.5929	0.0000	0.0500
GPQA-v2-Diamond (8B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.2359	0.3727	0.0701	1.0016	0.3871	0.5000	0.9408	0.0001	0.0100
MATH-500-v2 (8B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.4099	1.0000	1.7182	0.8949	X	X	0.9976	0.0001	0.0100
MATH-500-v2 (8B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4110	0.7440	0.0785	1.0033	0.3333	0.5036	0.9984	0.0000	0.0100
MATH-500-v2 (8B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.5058	0.7038	5.1187	1.0175	0.0666	2.5000	0.9964	0.0000	0.0500
MATH-500-v2 (8B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.5071	0.7508	1.8068	3.0000	0.2890	0.5000	0.9796	0.0001	0.0100
MMLU-Pro-v2 (8B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.3906	1.0000	1.9045	0.7590	X	X	0.9991	0.0000	0.0100
MMLU-Pro-v2 (8B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4764	0.6846	2.7876	3.0000	0.2136	0.6141	0.9985	0.0000	0.2500
MMLU-Pro-v2 (8B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4439	0.5662	0.1136	0.9916	0.2787	0.6659	0.9084	0.0001	0.0100
MMLU-Pro-v2 (8B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4771	0.7025	2.8863	3.0000	0.1728	0.5181	0.9986	0.0000	0.1000
MMLU-College-v2 (8B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.4316	0.9924	0.9887	0.9123	X	X	0.9994	0.0000	0.5000
MMLU-College-v2 (8B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6226	0.8494	1.2646	3.0000	0.3346	0.5000	0.9958	0.0000	0.2500
MMLU-College-v2 (8B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6359	0.7368	1.0929	0.5130	0.0576	2.5000	0.9949	0.0000	0.0500
MMLU-College-v2 (8B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6283	0.8085	1.4279	3.0000	0.3914	0.5000	0.9845	0.0000	0.0100

C.4. Scaling Candidate Generations

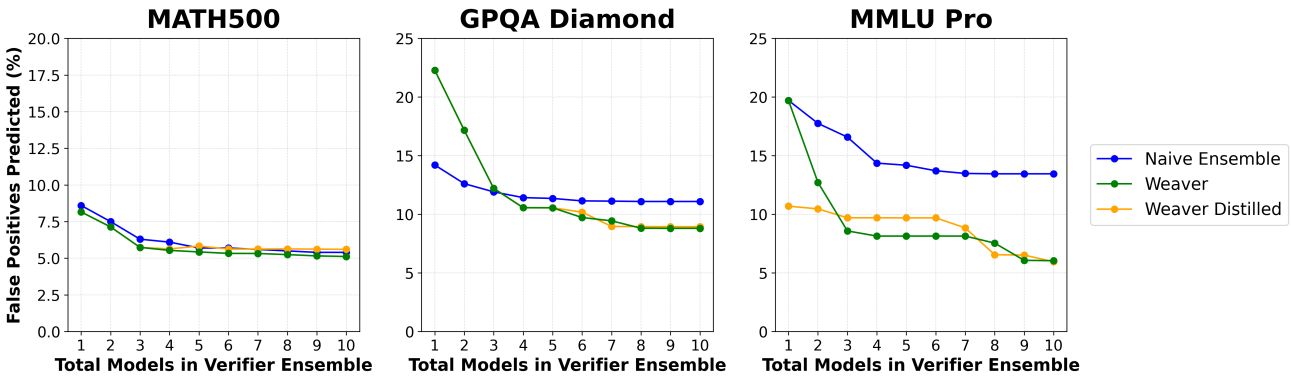

Figure 19: **False Positive Rates across Verification Systems**

Table 19: Fitted parameters for Scaling Trends with 90% of data in Figure 17 and Figure 18.

Dataset	Approach	Equation	floor	ceil	ζ	α	π	γ	R2 fit	MSE fit	MSE pred	δ
GPQA-v2-Diamond (70B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.0000	0.9357	0.7534	0.3537	X	X	0.9999	0.0000	0.0000	0.1000
GPQA-v2-Diamond (70B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4050	0.6756	0.9195	1.6227	0.3163	0.5000	0.9993	0.0000	0.0000	0.0500
GPQA-v2-Diamond (70B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4320	0.4678	0.0707	0.9864	0.0414	1.6541	0.8601	0.0000	0.0001	0.0100
GPQA-v2-Diamond (70B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.3809	0.6061	0.5211	1.9075	0.4360	0.5000	0.9972	0.0000	0.0000	0.0500
MATH-500-v2 (70B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.6639	0.9952	0.9836	0.6936	X	X	0.9995	0.0000	0.0000	0.0100
MATH-500-v2 (70B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.7869	0.9339	3.5000	3.0000	0.2985	0.5000	0.9954	0.0000	0.0000	0.0500
MATH-500-v2 (70B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.7747	0.8241	10.0000	2.1272	0.0888	2.4999	0.8560	0.0001	0.0000	0.0500
MATH-500-v2 (70B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.7879	0.9221	4.2081	3.0000	0.2789	0.5000	0.9966	0.0000	0.0000	0.1000
MMLU-Pro-v2 (70B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.0000	0.9785	0.3263	0.3543	X	X	0.9959	0.0000	0.0000	0.1000
MMLU-Pro-v2 (70B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6912	0.9148	1.5277	3.0000	0.2765	0.5000	0.9984	0.0000	0.0000	0.2500
MMLU-Pro-v2 (70B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6923	0.7379	0.0495	1.0001	0.1711	1.0148	0.9481	0.0000	0.0000	0.0500
MMLU-Pro-v2 (70B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6979	0.8907	1.8452	3.0000	0.2327	0.5000	0.9931	0.0000	0.0000	0.1000
MMLU-College-v2 (70B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.7723	0.9655	1.3237	0.7746	X	X	0.9987	0.0000	0.0000	0.1000
MMLU-College-v2 (70B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.8184	0.9436	2.2897	2.0761	0.4148	0.5000	0.9979	0.0000	0.0000	0.2500
MMLU-College-v2 (70B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.8196	0.8413	0.0498	1.0001	0.2051	1.0153	0.8814	0.0000	0.0000	0.0100
MMLU-College-v2 (70B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.8234	0.9262	3.9001	3.0000	0.3036	0.5000	0.9909	0.0000	0.0000	0.1000
GPQA-v2-Diamond (8B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.2223	0.9976	2.4975	0.8328	X	X	0.9996	0.0000	0.0000	0.2500
GPQA-v2-Diamond (8B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.2316	0.4678	0.0559	1.0027	0.2347	0.5940	0.9965	0.0000	0.0002	0.0100
GPQA-v2-Diamond (8B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.2773	0.2979	0.1335	0.9633	0.0479	2.5000	0.5476	0.0000	0.0001	0.0500
GPQA-v2-Diamond (8B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.2871	0.3845	8.5633	3.0000	0.2431	0.5000	0.9643	0.0000	0.0001	0.0100
MATH-500-v2 (8B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.3986	1.0000	1.6393	0.8786	X	X	0.9976	0.0001	0.0001	0.0100
MATH-500-v2 (8B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4149	0.7417	0.0750	1.0042	0.3261	0.5179	0.9981	0.0000	0.0000	0.0100
MATH-500-v2 (8B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.5061	0.6976	5.9901	1.1205	0.0782	2.5000	0.9960	0.0000	0.0000	0.0500
MATH-500-v2 (8B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.5009	0.7342	1.6358	3.0000	0.3321	0.5000	0.9803	0.0001	0.0005	0.0100
MMLU-Pro-v2 (8B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.3877	1.0000	1.8798	0.7549	X	X	0.9990	0.0000	0.0000	0.1000
MMLU-Pro-v2 (8B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4770	0.6937	3.1648	3.0000	0.2172	0.5705	0.9986	0.0000	0.0001	0.0500
MMLU-Pro-v2 (8B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4663	1.0000	2.4923	0.1016	0.0362	2.5000	0.9600	0.0001	0.0002	0.0500
MMLU-Pro-v2 (8B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.4777	0.7207	3.0170	3.0000	0.1613	0.5000	0.9987	0.0000	0.0000	0.0100
MMLU-College-v2 (8B)	Pass@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})$	0.4442	0.9912	1.0258	0.9265	X	X	0.9993	0.0000	0.0000	0.5000
MMLU-College-v2 (8B)	Weaver	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6178	0.8447	1.1473	3.0000	0.3523	0.5000	0.9957	0.0000	0.0001	0.2500
MMLU-College-v2 (8B)	Majority1@K	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6254	0.7186	0.5765	0.9281	0.2058	1.2287	0.9736	0.0000	0.0001	0.0100
MMLU-College-v2 (8B)	Naive Ensemble	$y = \text{floor} + (\text{ceil} - \text{floor}) \cdot \exp(-\zeta \cdot K^{-\alpha})(1-(1-\pi)^{K^{\gamma}})$	0.6074	0.9813	1.2560	0.1636	0.3060	2.4651	0.9988	0.0000	0.0000	0.0100

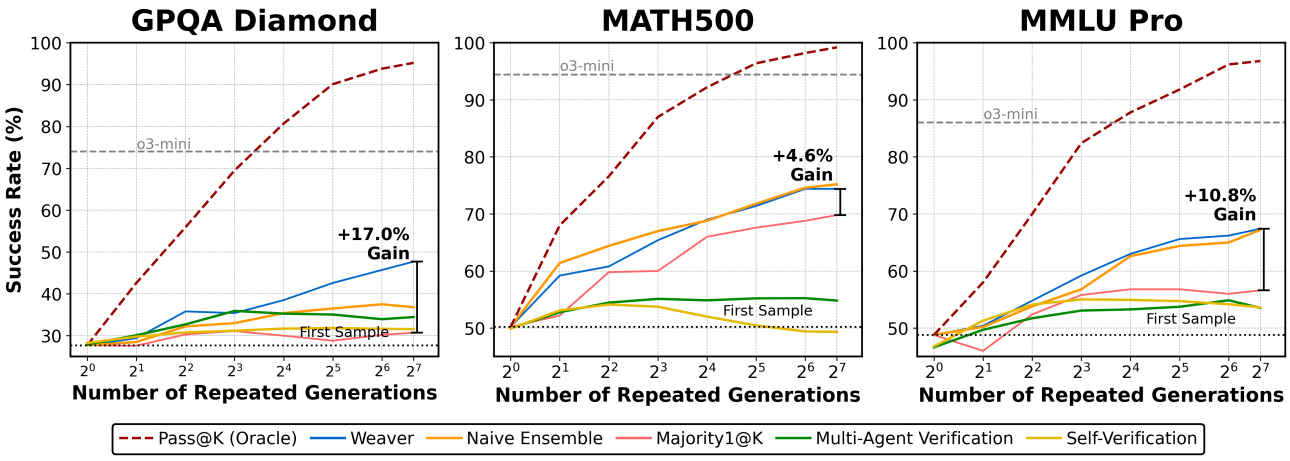

Figure 20: **WEAVER Scaling - 8B Generations and Models**

Table 20: **WEAVER with 8B Models Exceeds Majority Voting and Naive Ensemble across All Datasets**: Candidate are generated with Llama 3.1 8B Instruct while the weak verifiers are 8B parameters or smaller in size.

	Methodology	Generations (K)	Datasets				Average
			MATH 500	GPQA	MMLU College	MMLU Pro	
Baselines	First Sample	1	49.8%	28.3%	64.1%	46.6%	47.2%
	Majority Voting	100	69.0%	30.5%	72.7%	56.4%	**57.2%**
	Top-Ranked RM from RewardBench (Lambert et al., 2024)	100	73.8%	25.4%	70.1%	53.4%	55.7%
	Top-10 RM Ensemble from RewardBench (Lambert et al., 2024)	100	70.2%	22.1%	73.9%	49.4%	53.9%
	Multi-Agent Verification (Lifshitz et al., 2025)	100	65.4%	31.4%	70.5%	55.2%	55.6%
	Self-Verification (Zhao et al., 2025)	100	71.4%	32.2%	70.4%	53.0%	56.8%
	WEAVER	100	80.0%	47.1%	85.7%	67.2%	**70.0%**
	GPT-4o-mini	1	76.8%	38.4%	82.2%	61.8%	64.8%
	Claude 3.5 Haiku	1	70.0%	36.4%	75.9%	65.2%	61.9%
	Oracle Verifier (Pass@100)	100	99.2%	95.2%	98.5%	96.8%	**97.4%**

Table 21: **Ensembling with Multiple Verifiers Outperforms Increased Sampling with Single Verifier**: Candidate responses are generated with Llama 3.3 70B Instruct while the weak verifiers range in size from 8B to 72B parameters. For details on prompting, please see Appendix C.7.

Methodology	Benchmarks		
	MATH500	GPQA	MMLU Pro
First Sample	78.0%	42.9%	69.9%
Majority Voting	83.0%	47.4%	74.4%
Best Reward Model (1 Score)	94.4%	58.4%	81.8%
Best Reward Model (5 Scores, 5 Prompts)	93.2%	55.3%	82.5%
Top-5 Most Accurate Reward Models	**95.4%**	64.1%	**87.3%**
Best LM Judge (1 Score)	90.2%	61.1%	79.5%
Best LM Judge (5 Scores, 5 Prompts)	88.1%	57.2%	80.8%
Top-5 Most Accurate LM Judges	93.4%	**65.2%**	85.4%

C.5. Scaling Verifier Count

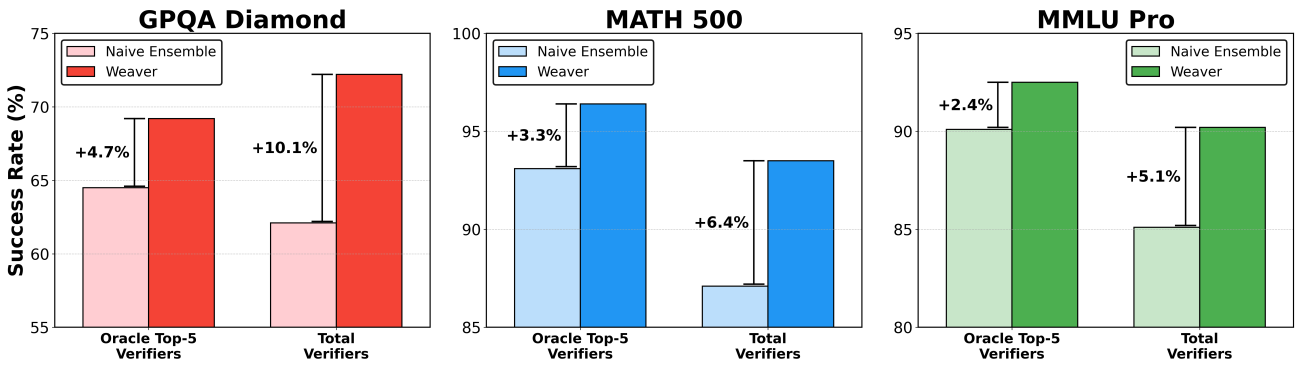

Figure 21: **WEAVER Outperforms Naive Ensemble across Oracle Top-5 Verifiers and Total Verifiers Configurations:** Results are shown for WEAVER ensembles and naive ensembles of the *Oracle Top-5 Verifiers* (highest-performing verifiers on dataset selected using ground truth) and *Total Verifiers* (all available verifiers). WEAVER consistently outperforms naive ensemble averaging, with improvements ranging from +2.4% to +10.1%.

In Table 21, we include results of scaling verifier scores. We note that for reward models (RMs), which are typically deterministic (Lambert et al., 2024; Song et al., 2025a), multiple scores must be obtained by varying the prompt; for LM Judges, we can vary either the prompt or the sampling temperature to generate diverse outputs from the same model (Table 21). We find that for both types of weak verifiers, RMs and LM judges, scaling the number of models yields better performance than sampling multiple evaluations from the same model via prompt tuning or temperature variation. However, we note that these approaches are complementary.

When breaking down weak verifiers into RMs or LM Judges, individually, we find that additional LMs leads average gains of 5.4% and 6.1%, respectively (Table 21). In contrast, sampling additional scores from a single RM or LM judge yields only 0.8% and 1.1% gains on average. These results suggest that leveraging the complementary strengths of multiple verifiers can be more effective than eliciting multiple judgments from a single verifier. Appendix C.7 provides additional details on the verifier prompting. Finally, Figure 22 illustrates the tradeoff of scaling the number of verifiers versus increasing the number of scores from a single verifier, showing that scaling verifiers is helpful when the coverage increases as we increase sample count.

In Figure 22 illustrates how the number of verifiers and repeated generations interact to influence success rate. We observe

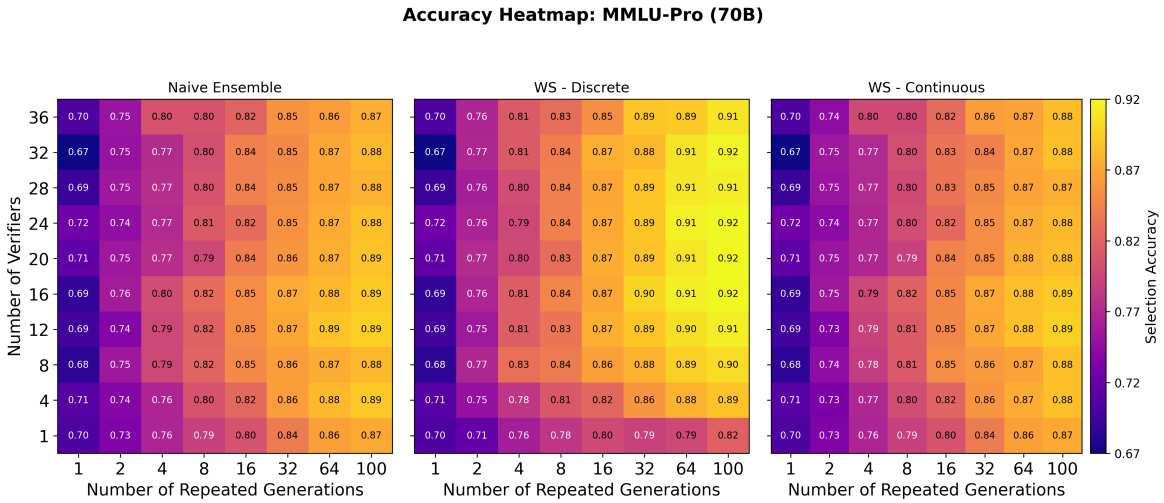

Figure 22: **WEAVER Performance Improvements from Scaling Generations and Verifiers**: Increased candidate generations and weak verifiers available generally improves performance.

that increasing the number of generations tends to be more effective than increasing the number of verifiers alone—but only when paired with the right verification strategy. For example, naive ensembling of verifiers plateaus in performance even as more generations are added, whereas WEAVER continues to improve with both axes. This highlights that generation diversity is a stronger driver of performance than verifier count alone, and that weak supervision methods like WEAVER are essential to fully leverage this diversity. We illustrate the verification generation tradeoff for additional datasets in Appendix C.3.

C.6. WEAVER Distillation

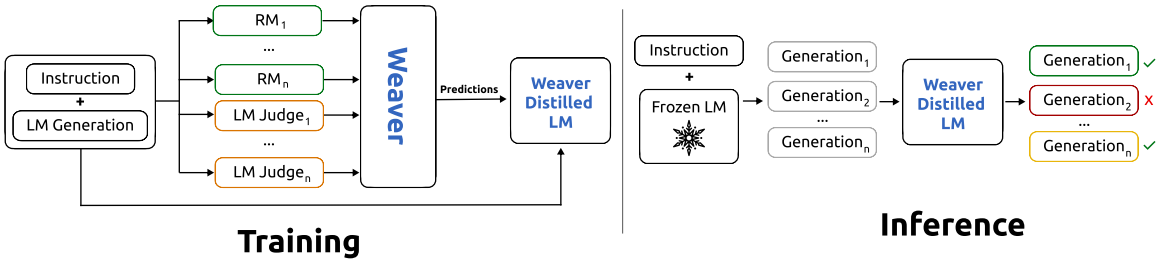

Figure 23: **Overview of WEAVER Distillation (Section 6)**

For the loss function in WEAVER distillation, we utilized cross-entropy loss with Adam (Kingma & Ba, 2017). Our classification architecture comprises a single linear classification layer with 0.1 dropout applied to the input, which consists of the final hidden state from the $[CLS]$ token. Regarding learning dynamics, we implemented linear warmup and linear decay via the Sentence-Transformers library (Reimers & Gurevych, 2020), employing a learning rate of 5e-6 and training batch size of 64 across all experimental setups.

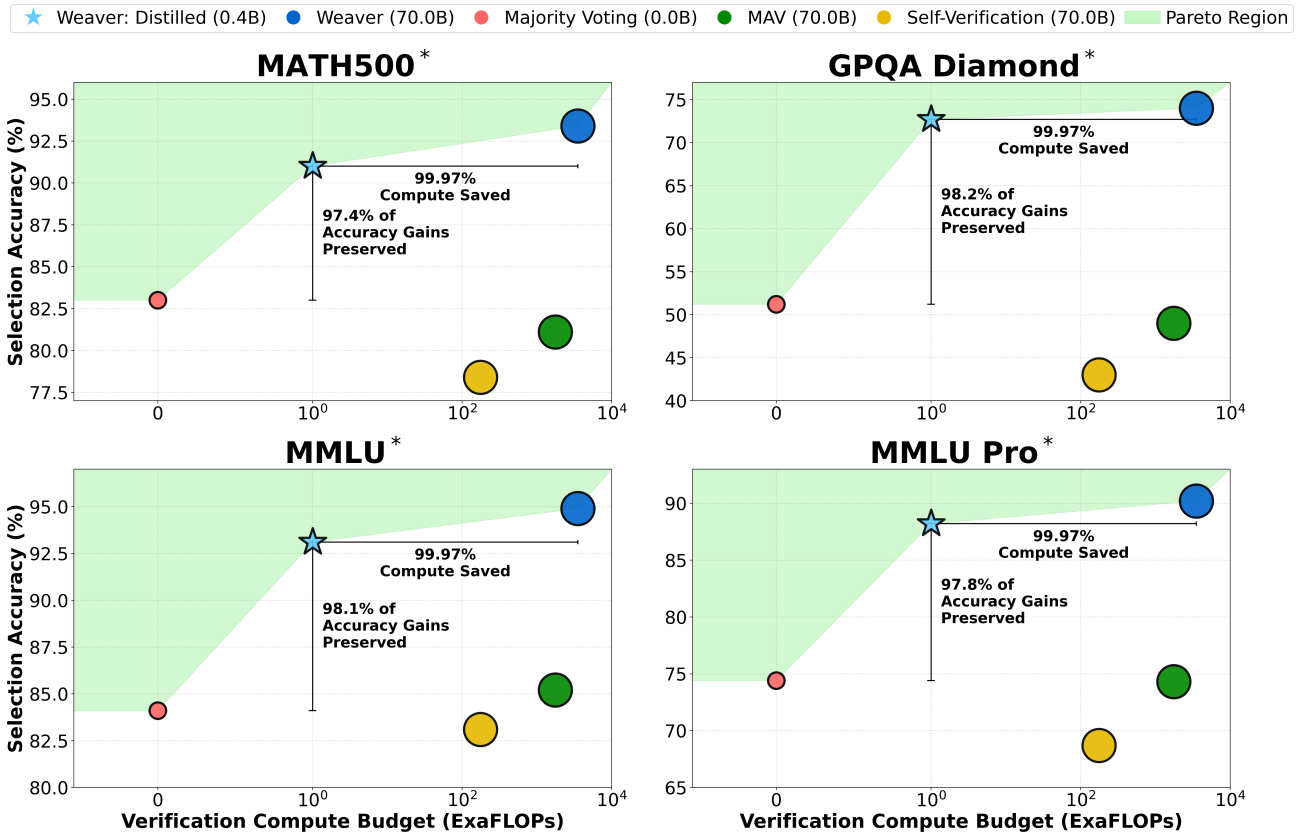

Figure 24: **WEAVER Distilled - Pareto Frontiers**: *We train/evaluate on an 80:20 split.

C.7. Individual Verifier Optimization

While WEAVER primarily focuses on aggregating multiple weak verifiers to improve overall verification quality, this appendix explores complementary techniques for optimizing individual verifiers. As mentioned earlier in the paper, existing weak verifiers often suffer from high false positive rates (Stroebl et al., 2024), which can limit their effectiveness, even within an ensemble.

As we scale the number of repeated samples and employ multiple verifiers, the precision of each individual verifier becomes increasingly important relative to recall. When many candidate solutions are available, a verifier can afford to miss some correct solutions (false negatives) as long as its positive predictions are highly reliable (high precision).

This observation motivates exploring methods to enhance individual verifier quality through methods such as prompt optimization — tailoring verifier prompts to maximize performance, particularly precision, with minimal or no labeled data.

C.7.1. LM JUDGE PROMPT OPTIMIZATION

LM judges often suffer from biases such as position bias (favoring answers in certain positions), verbosity bias (preferring longer answers), and self-enhancement bias (preferring answers similar to their own generation patterns) (Zheng et al., 2023a; Li et al., 2023), suggesting sensitivity to system and input prompt design.

Throughout our WEAVER experiments, we used fixed, manually engineered prompts for our LM judge verifiers. However, optimizing these prompts could potentially improve individual verifier precision and reliability. Multi-Agent Verification (Lifshitz et al., 2025) demonstrates this by crafting specialized prompts for specific verification aspects.

We explored systematically optimizing verifier prompts using DSPy (Khattab et al., 2023), an open-source library that provides algorithms for optimizing language model prompts through discrete search over prompt candidates guided by a

Table 22: **Distillation Comparison of WEAVER and Naive Ensemble Across Different Training Set Sizes**

Methodology	Dataset	Training Set as Percentage of Entire Dataset					Full System
		5%	10%	20%	50%	80%	
WEAVER	MATH500	78.4%	80.7%	83.9%	88.2%	91.4%	93.4%
	GPQA Diamond	42.6%	46.8%	52.7%	63.1%	71.8%	73.2%
	MMLU College	83.5%	85.2%	87.6%	91.0%	93.1%	94.9%
	MMLU Pro	69.2%	72.5%	76.8%	83.7%	87.8%	90.2%
NaiveEnsemble	MATH500	77.8%	79.6%	82.1%	86.4%	89.1%	92.4%
	GPQA Diamond	42.1%	44.7%	48.9%	56.2%	62.8%	66.2%
	MMLU College	84.0%	85.3%	87.2%	90.8%	93.5%	95.1%
	MMLU Pro	69.5%	71.8%	74.9%	80.3%	84.7%	87.4%

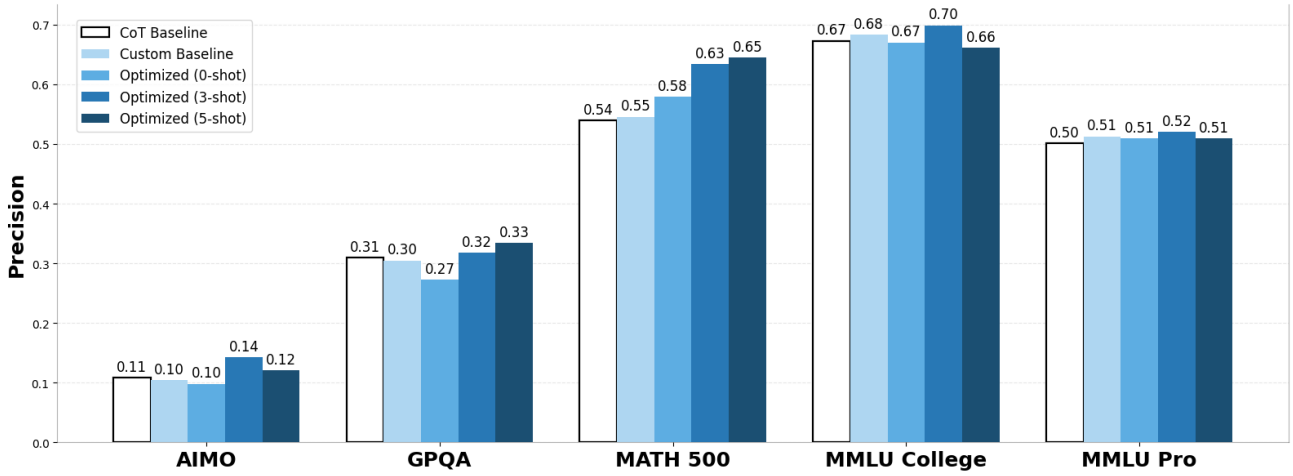

Figure 25: **LM judge prompt optimization using 250 labeled examples consistently yields precision gains**. Baseline methods (CoT and Custom) are compared against DSPy-optimized prompts with varying numbers of demonstrations (0-shot, 3-shot, and 5-shot).

metric function. DSPy optimization works by generating, evaluating, and refining prompts that maximize task performance on a small labeled dataset.

Experimental Setup: We investigate two dimensions of prompt optimization: (1) **optimization space scaling**, where we progressively expand what the optimizer can modify from system instruction only (0-shot) to including 3 demonstrations (3-shot) and 5 demonstrations (5-shot); and (2) **training data size scaling**, where we vary labeled data from 1% to 16% to determine how much data is necessary for effective prompt optimization.

Our experimental setup uses training examples containing instruction-generation pairs. Since our datasets have multiple generations per instruction (up to 100), we group examples by instruction before splitting to prevent data leakage between train and validation sets. We hold out 50% of the dataset instructions (each paired with 100 candidate generations) for evaluation. For the optimization space scaling experiment, we randomly select n generations such that $n \times \text{len(dataset)}/2 = 250$, maximizing training set diversity while maintaining a fixed training set size. For the data scaling experiment, we train on different percentages (1%, 2%, 4%, and 16%) of the dataset by calculating the number of instructions as $\lceil \text{num\_problems\_in\_dataset} \times (\text{train\_percentage}/100) \rceil$ and selecting repeated samples for each instruction with samples $= \min(\max(4, \text{num\_problems} \times 2), 20)$ to avoid overfitting. We use a consistent random seed to ensure identical dataset splits between optimization runs.

Results: Figure 25 shows results across different datasets and optimization configurations. While we don't observe clear scaling relationships across all datasets (possibly due to the increased stochasticity of LLM-based optimization), we observe an average precision gain of 3.8% of the best judge over the chain-of-thought (CoT) baseline judge. MATH500 shows the largest jump in precision of 9% and shows clear improvement in precision as the optimization space is scaled.

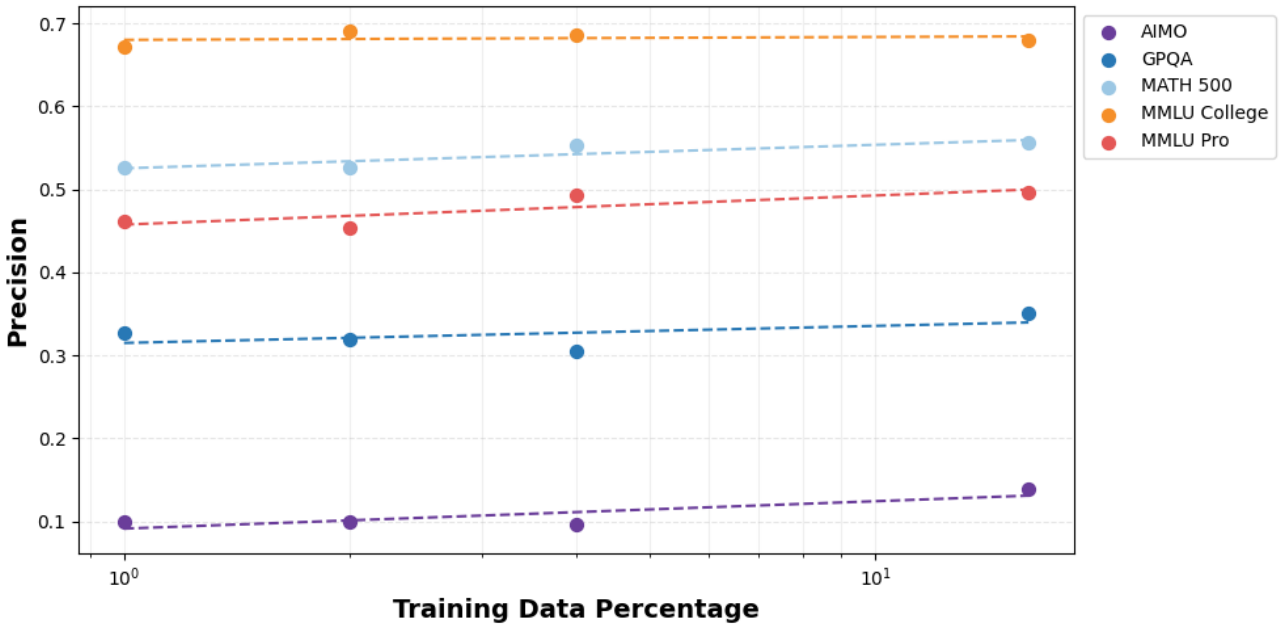

Figure 26: **Scaling LM judge prompt optimization training data leads to modest precision gains.** The x-axis shows the percentage of training data used (log scale), and the y-axis shows precision.

The scaling behavior with training data size (Figure 26) shows slight log-linear improvements in precision as we increase training data, though gains differ by dataset. MMLU-College shows minimal benefit from additional data, while the remaining datasets see an average boost of 3.2% in precision when scaling the training data size from 1% to 16% of the original dataset.

(Figure 27) reveals that optimized prompts often improve both precision and accuracy by reducing false positive rates - essentially making judges more conservative in their correctness assessments. This is particularly valuable in the repeated sampling regime, where higher precision improves overall verification quality.

These findings suggest that prompt optimization can be a valuable complement to WEAVER's aggregation approach. Even with limited labeled data, targeted prompt engineering can enhance individual verifier quality, benefiting the ensemble as a whole. Further research is needed to define a more systematic recipe for verifier prompt optimization. Additionally, it remains a question of whether we can extend prompt optimization to discriminative reward models to enjoy similar gains in performance.

D. Miscellaneous

D.1. Compute Requirements

Hardware Infrastructure. Our experiments were conducted using 4 compute nodes, each equipped with 8 NVIDIA H100 GPUs (80GB HBM3 memory per GPU), for a total of 32 H100 GPUs. Each node was configured with high-bandwidth NVLink connections between GPUs and inter-node communication was facilitated via NVIDIA NVLink Switch System to minimize communication overhead during distributed training and inference.

Model Parallelism and Distribution. For our 72B parameter language models, we employed a hybrid parallelism strategy combining tensor parallelism, pipeline parallelism, and data parallelism:

- 8-way tensor parallelism across GPUs within each node
- 4-way pipeline parallelism across nodes
- Data parallelism for batch processing

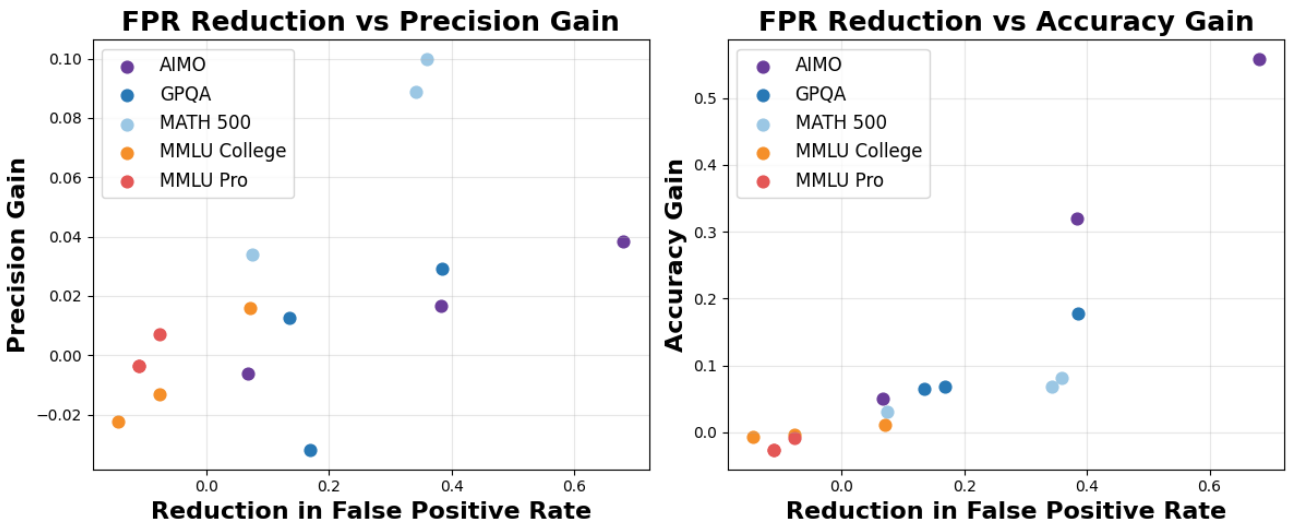

Figure 27: **Optimized prompts often improve LM judge performance by reducing false positive rates.**

Storage Requirements. Processing datasets of 100GB+ required significant storage infrastructure:

- 4TB NVMe SSDs per node for dataset caching and checkpoints
- 100TB shared network storage for full dataset repository

Software Stack. Our experiments were powered by:

- NVIDIA CUDA 12.2
- PyTorch 2.1 with NVIDIA NCCL for distributed communication
- DeepSpeed ZeRO Stage 3 for memory optimization
- Distributed data loading with webdataset format for efficient streaming