LIGHTWEIGHT LATENT VERIFIERS FOR EFFICIENT META-GENERATION STRATEGIES

Bartosz Piotrowski
IDEAS NCBR

Witold Drzewakowski
University of Warsaw
IDEAS NCBR

Konrad Staniszewski
University of Warsaw
NVIDIA

Piotr Miłoś
IMPAN
IDEAS NCBR

ABSTRACT

We study verifiers understood as auxiliary models estimating the correctness of outputs generated by base large language models (LLMs). Such approximate verifiers are crucial in many strategies for solving reasoning-intensive problems with LLMs. Typically, verifiers are LLMs themselves, often as large (or larger) than the base model they support, making them computationally expensive. In this work, we introduce a novel lightweight verification approach, LiLaVe, which reliably extracts correctness signals from the hidden states of the base LLM. A key advantage of LiLaVe is its ability to operate with only a small fraction of the computational budget required by traditional LLM-based verifiers. To demonstrate its practicality, we couple LiLaVe with popular meta-generation strategies, like best-of-n or self-consistency. We also design novel LiLaVe-based approaches, like *conditional self-correction* or *conditional majority voting*, that improve both accuracy and efficiency in generation tasks with smaller LLMs. Our work opens the door to scalable and resource-efficient solutions for reasoning-intensive applications.

1 INTRODUCTION

Large language models (LLMs) have shown unprecedented performance in a plethora of tasks related to processing natural language and knowledge retrieval. Recently, there has been substantial interest in enhancing *reasoning capabilities* of LLMs. Specifically, this effort includes applying LLMs to solve mathematical problems (Cobbe et al., 2021; Trinh et al., 2024; Glazer et al., 2024), writing code (Ahn et al., 2024), performing numerical computations (Charton, 2024), recognizing spatial patterns (Chollet et al., 2024), and predicting proof steps in proof assistants (Mikuła et al., 2024).

Efforts to improve LLM performance on reasoning-intensive tasks have followed two primary directions. First, there is a substantial body of work focusing on *pre-training* or *fine-tuning* models targeting reasoning-intensive tasks. To this end, high-quality, reasoning-focused data are collected, like OpenWebMath (Paster et al., 2023), or Proof Pile (Azerbayev et al., 2023). In addition to that, new training methodologies are being developed, such as self-improvement loops (Zelikman et al., 2022), or reinforcement-learning-based approaches (Guo et al., 2025).

Second, there is ongoing research into designing *inference-time techniques* to enhance the performance of LLMs on reasoning-focused tasks, where an LLM is already pre-trained and fixed. Examples of such simple yet effective techniques are chain-of-thought prompting (Wei et al., 2022), and self-consistency decoding (Wang et al., 2023), also known as *majority voting*. More advanced inference-time approaches often combine the decoding process from the base LLM with a *verifier*, trained to assess the correctness of individual reasoning steps – or entire reasoning trajectories – in order to enhance the base model's performance (Cobbe et al., 2021; Lightman et al., 2023). Typically, verifiers are LLMs themselves, often as large – or larger – than the base model they support, making them computationally expensive. For instance, in recent work by Wu et al. (2024), an LLM verifier of size 34B of parameters is paired with base models of size 7B and 34B parameters.

To overcome this limitation, our work aims to develop computationally efficient verifiers, which can be used to enhance the performance of the base LLMs in reasoning-intensive tasks. To this end, we develop LiLaVe – Lightweight Latent Verifier, which is a simple and practical method for extracting

Corresponding author: bartoszpiotrowski@post.pl

the correctness signal from the *hidden states* of the base LLM. Subsequently, through a series of experiments, we demonstrate how our verifiers can be practically and effectively used to implement various *meta-generation strategies* focused both on *correctness of the inferred answers* as well as on *inference-time compute-efficiency*. In summary, our contributions are as follows:

- We introduce LiLaVe, a novel lightweight verifier that extracts correctness signals from the hidden states of the base LLM; we show that it outperforms other approaches in the AUC metric.
- We experimentally study which hidden states across the model's layers and the output sequence's tokens provide the optimal correctness signal.
- We demonstrate that LiLaVe can be used to significantly improve the accuracy and inference-time compute-efficiency of smaller LLMs on reasoning tasks via meta-generation strategies:
  - We introduce *conditional majority voting* approach which reduces the average inference cost while maintaining high accuracy.
  - We demonstrate the effectiveness of the *conditional self-correction* approach, in which the base model is asked to self-correct only when the verifier's score is low.

2 Related work

Reasoning and large language models Step-by-step problem-solving is fundamental to human intelligence and scientific discovery. Mathematical problems are often considered a hallmark of reasoning and have been extensively studied in the context of LLMs (Lewkowycz et al., 2022; Cobbe et al., 2021; Hendrycks et al., 2021). The field is advancing rapidly, with models like OpenAI's o3 solving certain research-level problems from the FrontierMath benchmark (Glazer et al., 2024). Although o3's training details remain undisclosed, conjecturally similar DeepSeek-R1 (Guo et al., 2025) exemplifies the class of "thinking models", typically trained with reinforcement learning to conduct extensive searches over the space of solutions. The flip side is the high inference cost; o3 reportedly used 33M tokens to solve a single ARC-AGI puzzle (Chollet, 2019; Chollet et al., 2024). This underscores the need for efficient inference, which became a growing research focus. Snell et al. (2024) and Wu et al. (2024) explore trade-offs between model size and inference time, aiming to establish compute-optimal strategies. Our work similarly prioritizes inference-time efficiency.

Inference time techniques Chain-of-Thought (CoT) prompting (Wei et al., 2022; Nye et al., 2021) is arguably the most widely adopted technique for improving LLM reasoning. Self-consistency decoding (Wang et al., 2023) involves generating multiple answers and applying majority voting. Furthermore, tree and graph search methods, including Monte Carlo Tree Search and AlphaZero-inspired techniques, have been widely studied (Yao et al., 2023; Besta et al., 2024; Feng et al., 2024; Welleck et al., 2022; Lample et al., 2022). Another research direction focuses on iterative refinement (Havrilla et al., 2024; Madaan et al., 2023; Shinn et al., 2023). However, the computational effectiveness of these methods is still unclear (Huang et al., 2024; Havrilla et al., 2024). Our work contributes to the area of inference-time techniques by proposing a lightweight verifier, that achieves high accuracy with lower computational costs. For a broader overview, see (Welleck et al., 2024).

Approximate verifiers LLM-generated answers or reasoning process can be assessed by a fine-tuned model known as a *verifier*. Verifiers can be trained to predict the correctness of entire answers (Cobbe et al., 2021) or verify individual reasoning steps (Lightman et al., 2023). Step-by-step verification also appears in (Yu et al., 2024; Havrilla et al., 2024; Uesato et al., 2022). Acquiring training data remains the key challenge. Lightman et al. (2023) rely on costly human data, while Wang et al. (2024) and Havrilla et al. (2024) use synthetic data. Verifiers are typically larger than generators, but recently, Ye et al. (2024) examined LLM reasoning rationales and hidden mechanisms, suggesting that latent structure could support much simpler verifiers, which motivates our work.

Probing Probing the internal states of transformer models (Alain & Bengio, 2018) has become an established method of studying their latent representations (Gurnee & Tegmark, 2024), privacy leakage (Kim et al., 2023), and in-context algorithms (Akyürek et al., 2023). For a recent introduction to techniques for studying the internal workings of transformer-based language models, see (Ferrando et al., 2024). Hidden layer activations have been used to predict truthfulness of their generations especially in the context of hallucination detection (Azaria & Mitchell, 2023; Chen et al., 2024; He et al., 2024; Beigi et al., 2024). Outside of hallucination detection, OPENIA (Bui et al., 2025) notes that model internal representations encode information useful for predicting the correctness of generated code. While applying this insight to a different domain, we also use a different type of latent classifier and additionally study recipes for utilizing the verifiers to improve model generations.

3 METHOD

Our approach to improving the accuracy and efficiency of LLMs on reasoning-intensive tasks at test time involves two key components. First, we train a lightweight latent verifier (LiLaVe) using selected hidden states extracted from the LLM during the generation of CoT-style solutions of mathematical problems, labeled by the correctness of the final answers concluding them (see Section 3.1). Subsequently, we employ the verifier to estimate the probability of LLM's answers being correct and integrate it with various *meta-generation strategies* (see Section 3.2).

3.1 LIGHTWEIGHT LATENT VERIFIER – LILAVE

Data Given a question q, an LLM generates an answer sequentially as $y = y_1 y_2 \cdots y_m$, where y_is are individual tokens. During the decoding, we extract hidden states $h_t^l \in \mathbb{R}^n$ representing the activations from the l-th transformer's layer at the generation of the t-th token, where n is the hidden dimension of the model.[1] Rather than using all layer-token pairs (l, t), we restrict extraction to subsets of indices $l \in L, t \in T$. Section 4.1 details how we experimentally choose optimal L, T.

While the answer y contains the chain-of-thought style reasoning, we determine its correctness solely by looking at the final answer.[2] To evaluate correctness, we use an automated evaluator that compares the generated final answer to the ground truth, resulting in a binary correctness label c.[3] Finally, a dataset $\mathcal{D}$ for training LiLaVe consists of datapoints in the form of quadruples (h_t^l, l, t, c). Note that we extract $|L| \cdot |T|$ hidden states per one generation. Therefore, if Q is the dataset of questions and we sample k generations for each $q \in Q$, we have $|\mathcal{D}| = |L| \cdot |T| \cdot |Q| \cdot k$.

Training Having collected $\mathcal{D}$, we train an efficient classifier M to predict the binary label c given the hidden state h_t^l and its location given by the indices l, t. The output score $M(h_t^l, l, t) \in [0, 1]$ is to be interpreted as the probability of the response y to be correct.

We experimented with several classifiers suitable for such data, like logistic regression (Hastie et al., 2009), SwiGLU (Shazeer, 2020), and gradient boosted decision trees (Friedman, 2001). In our initial experiments, we observed that gradient-boosted decision trees (concretely, its XGBoost implementation by Chen & Guestrin (2016)) performed best and most robustly (see Appendix B.2). Therefore we chose to rely on this classifier.

Inference During inference, the base language model generates a response y along with a set of associated hidden states H_y, which are indexed by their locations (l, t). We then apply the trained XGBoost model M to predict a score s_h for each hidden state $h \in H_y$. Finally, these scores are aggregated, which results in the final correctness estimate, ***i.e.***, the LiLaVe score:

$$\text{LiLaVe}(y) = \text{aggregate}(\{s_h\}_{h \in H_y}) \in [0, 1].$$

After experimenting with several aggregation methods – taking minimum, maximum, or average score – we chose to use averaging as it performed best.

3.2 LILAVE-BASED META-GENERATION STRATEGIES

We consider several *meta-generation strategies*, i.e., strategies that build on top of the *base generator* (the base language model) and a trained LiLaVe verifier. First, we experiment with two standard approaches: *best-of-n* sampling and *weighted majority voting*. In both approaches, we first sample n responses from the base generator with fixed temperature $t > 0$. As the final response in **best-of-n**, we select the one with the highest LiLaVe score. In **weighted majority voting**, we perform a majority voting across the final answers extracted from n full responses, weighted by their LiLaVe scores.

Standard majority voting and its weighted variant are effective techniques; however, they are computationally expensive as they require generating multiple independent samples per question. In weighted voting, there is an additional cost of extracting hidden states from the decoded samples, which may cause a significant slowdown in practical settings.

[1] For example, for Llama 3.1 8B the dimensionality of the hidden states h_t^l is 4096, and the number of layers (*aka* transformer blocks) is 32.

[2] This does not exclude the possibility of *false positives*, where the final answer is correct but the rationale leading to it is flawed; this is, however, a rare situation.

[3] For datasets where the final answers are, e.g., integers, a direct comparison suffices; for some datasets, the answers may be a more complex mathematical expression and so more involved evaluation is needed – like in the case of the MATH dataset (see Section A).

This motivates our novel approach of **conditional majority voting**: first, we generate a single sample from the base generator, and we score it with LiLaVe. If the score is above a predetermined threshold $s \in [0,1]$, we consider it a final response. Otherwise, we interpret the low score as an indication of the base model's uncertainty and generate n additional samples to perform a majority voting.

Finally, we investigate a new meta-generation strategy of **conditional self-correction**. Prompting LLMs to verify and correct their responses gives varied results (Huang et al., 2024). LLMs often indeed can fix their mistakes, but at the same time, they tend to turn correct responses into incorrect ones in the process. This makes the self-correction procedure unreliable and, in most cases, unsuccessful. In the *conditional* self-correction, we leverage LiLaVe to achieve reliable accuracy improvements. First, we generate the initial response and score it with LiLaVe. Then, we prompt the model to self-correct its response only if the LiLaVe score is below a predetermined threshold s.

4 EXPERIMENTS

Here we present the experiments conducted in order to develop and evaluate LiLaVe. In Section 4.1, we study the influence of location of extracted hidden states as well as sampling temperature on the performance of LiLaVe. In Section 4.2, we introduce alternative baseline methods for estimating the correctness of the LLM reasoning, which we then compare with LiLaVe. In Section 4.3, we harness LiLaVe to the meta-generation strategies described in Section 3.2, and we demonstrate that despite being so lightweight, our verifier achieves substantial performance gains on the math benchmarks.

We evaluate LiLaVe and LiLaVe-based meta-generation strategies on four mathematical QA datasets: GSM8K (Cobbe et al., 2021), GSM-Symbolic (Mirzadeh et al., 2024), MATH (Hendrycks et al., 2021), and algebra_linear_1d (Saxton et al., 2019). For each of them we select 1000 training examples to train a dataset-specific LiLaVe. We test on sets of 500–1319 examples, depending on the dataset. In Appendix A we describe each of the four benchmarks.

In our main experimental line we use Llama 3.1 8B as the base language model. To test the universality of LiLaVe, we additionally experimented with Gemma 2 2B and Phi-3.5-mini – see Appendix B.

4.1 DEVELOPING LILAVE

Below, we describe experiments determining (1) the location of extracted internal language model information as well as (2) sampling temperatures resulting in optimal LiLaVe's performance.

Hidden states locations As described in Section 3.1, we train LiLaVe on hidden states extracted from the base language model. The hidden states we extract correspond to different layers of the transformer model as well as different tokens in the decoded sequences. It is not clear which of those locations can allow for extracting the best correctness signal, therefore we run experiment aiming to answer this question. We fix a set of layer indices L and token indices T as:

$$L = \{-1, -2, -4, -8, -16\}, \quad T = \{0, 1, 2, 3, \dots, 31, -32, -31, \dots, -3, -2, -1\}.$$

Negative indices follow the Python / Perl convention of list indexing: the element $-n$ is the nth element counting from the end of the list. For each $(l,t) \in L \times T$, we train a separate XGBoost model $M_{l,t}$ on hidden states corresponding to layer l and token t. Then, we evaluate each of the trained models $M_{l,t}$ on a testing partition, and calculate its testing AUC performance.[4]

Figure 1 presents results of the experiment for three datasets (algebra_linear_1d, GSM-Symbolic, and MATH). First, we observe that the correctness signal is better in the suffix of the decoded sequences (which is especially noticeable for algebra_linear_1d). However, curiously, the signal in the prefix of the decoded sequences is still significantly better than the random baseline (AUC $= 0.5$), which is especially visible for MATH. There is no significant distinction between different transformer's layers, and even layers as deep as -16 provide a good signal (Llama 3.1 8B used here has 32 layers.)

Based on the obtained results, we fix the following sets of indices of layers L_{LiLaVe} and tokens T_{LiLaVe} from which we extract hidden state to train and evaluate the LiLaVe verifier:

$$L_{\text{LiLaVe}} = (-1, -2, -4, -8, -16), \quad T_{\text{LiLaVe}} = (-1, -2, -3, \dots, -16).$$

In the LiLaVe's inference mode, for one LLM's decoding, we aggregate the scores of hidden states corresponding to those tokens and layers using arithmetic mean.

[4] The area under the ROC curve (AUC) represents the probability that the model, if given a randomly chosen positive and negative example, will rank the positive higher than the negative.

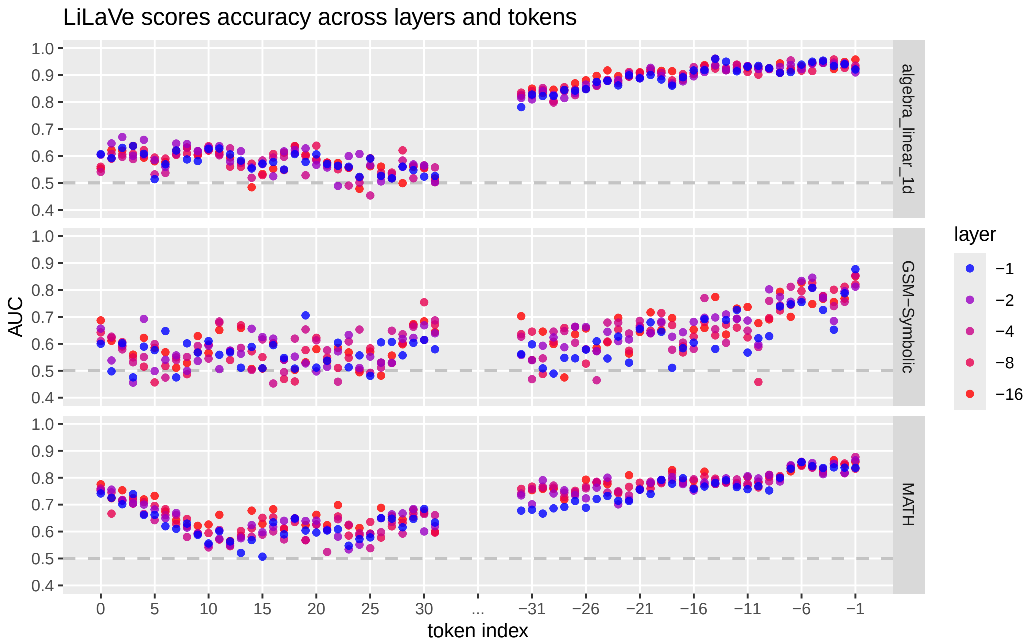

Figure 1: Predictive performance of LiLaVe on individual locations of hidden states determined by the indices of the transformer's layer and the sequence's token. We test the tokens from the prefix and suffix of the generated sequences, both of the length 32. It is visible that the higher-quality signal can be retrieved from the final tokens, however, interestingly, even the first tokens provide a signal significantly better than the random baseline (dashed lines). At the same time, we cannot conclude which transformer layers give the best signal.

Sampling temperature When generating samples for the LiLaVe training, it is not immediately clear which sampling temperatures should be used. On one hand, for reasoning-intensive problems low temperatures typically result in better performance. On the other hand, meta-generation techniques like majority voting require non-zero temperature to make the samples diverse (see Figure 12 in Appendix B demonstrating this trade-off for various datasets). Therefore, ideally, we want LiLaVe to perform well on samples generated across a range of temperatures. To check if it does, we experimentally study how the temperature of generations on which the verifiers are trained impacts their predictive ability when tested on answers to test questions, generated with various temperatures.

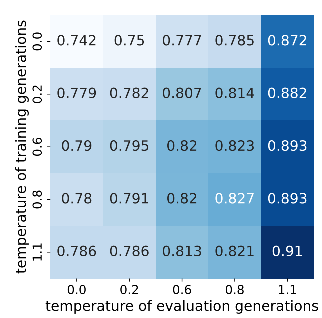

Figure 2: Performance (AUC) of LiLaVe trained and evaluated on hidden states of Llama 3.1 8B answers to GSM8K questions with various temperature settings.

Figure 8 shows a heatmap with the results of this analysis for hidden states of Llama 3.1 8B model and temperatures $\{0.0, 0.2, 0.6, 0.8, 1.1\}$ on the GSM8K dataset. Each cell of a heatmap is a mean of 16 experimental results: final AUC on the test set of 16 classifiers trained on hidden states from layers $\{-2, -4, -8, -16\}$ and tokens $\{-2, -4, -8, -16\}$. For all temperatures except 0, we generate 8 answers for each question.

We observe, that the predictive performance of the verifier increases both with the temperature of the evaluation samples as well as training samples. We hypothesize that increased temperature results in more diverse training examples and also examples with different correctness labels for one question, which is good for training the verifier. Higher temperature on the evaluation side likely results in samples that are incorrect in a way easier to detect by the verifier.

The experiment shows that increased temperature for generating training samples is beneficial. Given this result, and to ensure diversity in the training samples, we decide to train LiLaVe on samples generated with a mixture of five temperatures: $\{0, 0.25, 0.5, 0.75, 1.0\}$.

benchmark	LiLaVe	self-reflect	logprobs	ORM-Mistral	ORM-Deepseek
GSM8K (test)	0.86	0.68	0.78	0.81	**0.88**
GSM-Symbolic	0.84	0.70	0.78	0.85	**0.90**
GSM-Symbolic-p2	**0.78**	0.60	0.63	0.73	0.75
algebra_linear_1d	**0.93**	0.61	0.81	0.90	0.90
MATH500	0.88	0.79	0.67	0.79	**0.90**

Table 1: Performance (AUC) of five methods for predicting the correctness of the LLM's answers: LiLaVe and four baseline methods: self-reflection, logprob-based confidence estimation, and two LLM-based ORMs fine-tuned either on Mistral-7B or DeepSeekMath-Instruct data.

4.2 BASELINES

We compare LiLaVe with two natural baseline methods for estimating the probability of the correctness of the language model's answer: *logprob-based estimator* and *self-reflection prompting*, as well as two LLM-based verifiers. We evaluate the predictive power of LiLaVe and these four baselines using the AUC metric and display the results in Table 1.

Logprob-based estimator Assume that for a question q, a language model generates a response $y = y_1, y_2, \ldots, y_n$, where each decoded token y_i is given probability p_i. For each question, we compute the sum of log-probabilities over a k-suffix: $\sum_{i=0}^{k-1} \log p_{n-k}$. We treat this sum as an (uncalibrated) estimation of the output correctness. For each dataset, we choose the suffix length k, for which this estimator achieves the highest AUC score. We report results in Table 1. See Appendix B.4 for more details about this baseline, including a breakdown of performance over different suffix lengths.

Self-reflection prompting We prompt the same LLM that generated the answer to rate its confidence in the answer's correctness on a 1–10 scale. Similar methods were also used by Tian et al. (2023); Pawitan & Holmes (2024). We provide the specific prompt in Appendix C.

LLM-based verifiers We also benchmarked two LLM-based verifiers (aka outcome reward models, or ORMs), trained on over 250k synthetic examples generated from Mistral 7B and DeepSeekMath-Instruct 7B, as implemented by Xiong et al. (2024). Both of them are based of Llama 3.1 8B, and fine-tuned to return a real-valued score.

In Table 1 it can be seen the LiLaVe outperforms self-reflect, logprobs on all benchmarks, outperforms ORM-Mistral on all but one (GSM-Symbolic), and outperforms ORM-Deepseek on two benchmarks (GSM-Symbolic-p2, algebra_linear_1d), even though both ORMs are a few orders of magnitude bigger. Given these results, we conclude that LiLaVe excels at extracting useful signal estimating model's correctness. Additionally, in Table 2 in Appendix B we present LiLaVe's performance for two base LLMs different from Llama 3.1 8B used here: Gemma 2 2B (Mesnard et al., 2024) and Phi-3.5-mini (Abdin et al., 2024). For both of them, the results remain strong.

4.3 LILAVE-BASED META-GENERATION STRATEGIES

As shown above, LiLaVe proves to be effective in distinguishing correct and incorrect LLM's responses as measured by the AUC metric. In this subsection, we experimentally demonstrate that this statistical performance can be translated into efficient and practical meta-generation strategies.

Best-of-n and weighted majority voting First, we employ LiLaVe as a scoring function in best-of-n and weighted majority voting strategies (see Section 3.2). For both strategies, we generate between 1 and 16 samples per question with temperature 1.0, and score each of them with LiLaVe. In Figure 3, we show the results for both strategies comparing them with the baseline of standard majority voting.

The weighted majority voting strategy performs best across all numbers of votes, and for all datasets, whereas for MATH this dominance is the largest. For both GSM-Symbolic datasets, weighted majority voting is only slightly better than standard majority voting, and the difference diminishes with growing numbers of votes (samples). Best-of-n is weaker than weighted majority voting, and for higher numbers of samples also weaker than standard majority voting. This may be caused by *false positives*: responses appearing as correct to the verifier; the chance of encountering such examples grows with the number of samples.

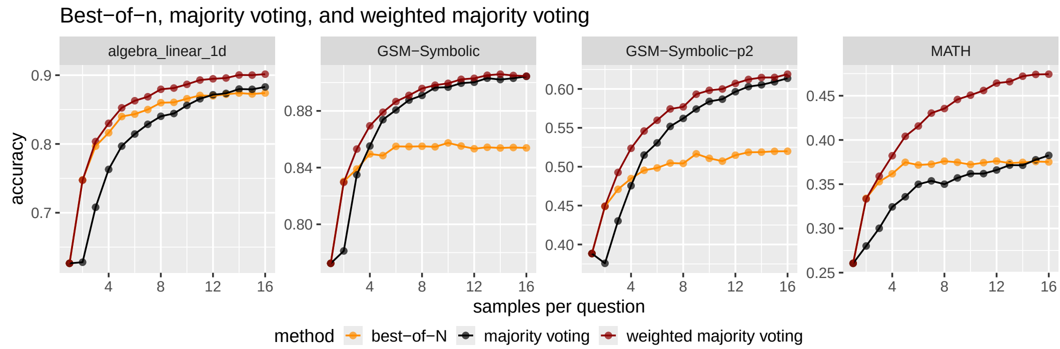

Figure 3: Best-of-n, majority voting, and weighted majority voting on four datasets. For each of the methods, the number of samples per question varied between 1 and 16. Weighted majority voting performs best for all the datasets, but the margin differs across the datasets.

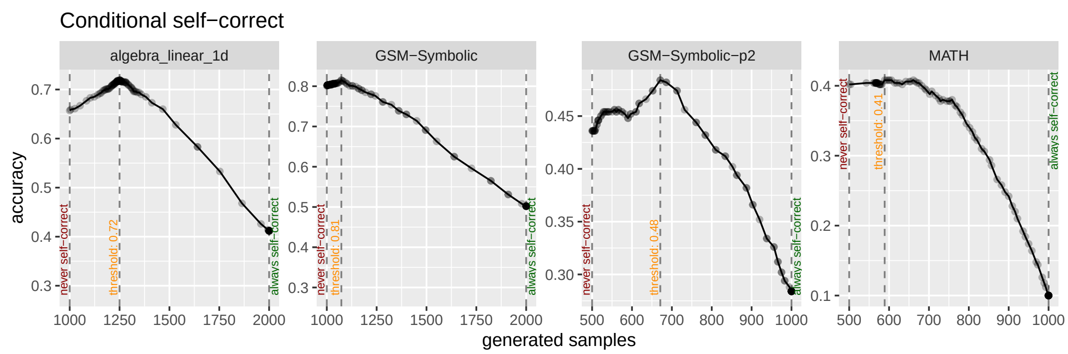

Figure 4: Conditional self-correction on four datasets. Points indicate the performance for different score thresholds. Left-most points correspond to no self-correction; right-most points correspond to unconditional self-correction. The optimal thresholds are indicated in orange color.

Conditional self-correction We evaluate the conditional self-correction strategy (Section 3.2) with a sampling temperature of 0. Figure 4 shows the performance across four datasets for varying thresholds $s \in [0, 1]$, which control how often self-correction is attempted.

A typical issue with self-correction is that while LLMs are often able to fix incorrect responses, they also turn many correct responses into incorrect ones. As seen in Figure 4, applying self-correction to all responses reduces accuracy by 15–30 percentage points. However, selectively correcting only low-scoring responses leads to significant gains for algebra_linear_1d and GSM-Symbolic-p2, with smaller improvements on other datasets. The optimal threshold varies per dataset (indicated in orange in Figure 4), so in practice, this hyperparameter must be tuned depending on the data.

Conditional majority voting In this meta-generation strategy (Section 3.2) four hyperparameters are involved: the temperature t_0 of generating the probe sample, the temperature of the samples for majority voting t_{mv}, the score threshold s below which the majority voting is triggered, and the number of majority voting samples n. We fix $t_0 = 0, t_{\mathrm{mv}} = 1$, and perform experiments with $n \in \{1, 2, 4, \dots, 256\}$ and a various $s \in [0, 1]$. Figure 5 presents results for two datasets.

In these plots, we do not explicitly show the n parameter but instead, on the x axis, we put the total number of samples generated when evaluating on all the examples (which is influenced by both n and s). This exposes an interesting fact: for a fixed budget (in terms of the number of generated samples), different combinations of n and s parameters of conditional majority voting give optimal accuracy. Importantly, conditional majority voting for lower budgets achieves better performance than standard majority voting (black line in the plots). This shows that LiLaVe-conditioned majority voting is a practical method allowing to trade between accuracy and efficiency in restricted budget settings.

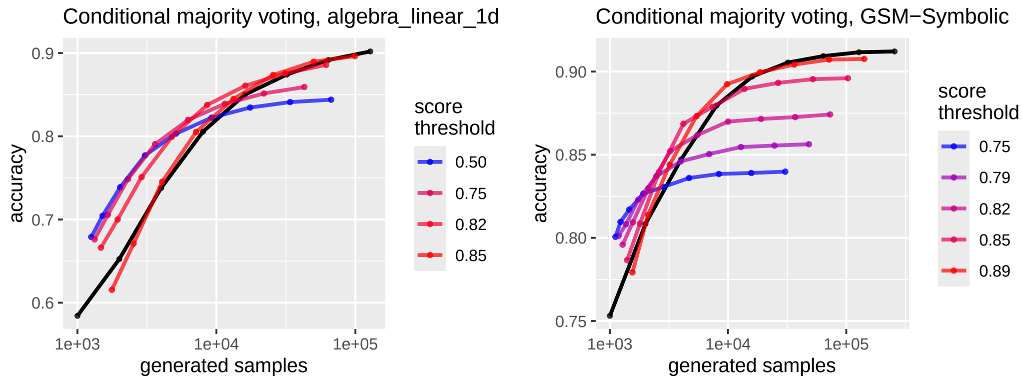

Figure 5: Conditional majority voting on two datasets with varying threshold s and the number of samples per question n varying between 1 and 256. The parameter n is shown implicitly as for fixed s it influences the total number of generated samples through the number of dataset questions scored below s (which for dataset $\mathcal{D}$ is equal $(n+1) \cdot |\mathcal{D}|$; the additional one sample per example is the probe sample). In black, the baseline of standard majority voting is shown. Conditional majority voting outperforms the baseline on a wide range of generation budgets.

5 Discussion

Our work introduces LiLaVe, a lightweight verifier that extracts correctness signals directly from the hidden states of a base LLM, reducing the need for expensive, separate verifier models. By integrating LiLaVe with meta-generation strategies like conditional majority voting and conditional self-correction, we improve both accuracy and computational efficiency in reasoning tasks. Our method is simple and robust, additionally exhibiting excellent transfer properties between datasets (see Table 4 in Appendix B.9). In our view this contributes to a new field of research focusing on efficiency (see Appendix D for quantitative discussion on LiLaVe's efficiency). In the long run, we believe that this may lead to more accurate models as the training data can be more diverse and larger.

6 Limitations and future work

Verifier-conditioned decoding In our experiments, the LiLaVe verifier scores answers after full generation. However, as shown in Figure 1, LiLaVe detects useful signal throughout the sequence, even at the first token. This suggests alternatively integrating the verifier directly into decoding as a *reward model*, to guide token selection toward high-certainty paths while avoiding erroneous trajectories. This direction is particularly promising, as LiLaVe is efficient and runs on CPU.

Verifier-oracle gap While our work advances test-time reasoning, there is still substantial room for improvement. In the best-of-n setting, where an oracle selects a correct answer if present among n samples, performance increases dramatically (see Figure 10 in Appendix B). This performance gap highlights the potential for improving verifiers, which could translate to significant gains.

We hypothesize that better verifiers can be obtained by possibly integrating information for a larger number of tokens and layers, as well as creating *ensemble models* combining LiLaVe with LLM-based ORMs, self-reflection prompting, and utilizing logprob information. LiLaVe is orthogonal to these other techniques which means it may bring decorrelated, valuable signal in an ensemble.

Moreover, LiLaVe can be combined with different base LLMs which may constitute multiple *standalone verifiers* that digest responses generated beforehand from an arbitrary model. See Appendix B.8 for a prototype experiment in that direction, which gave promising results.

Adaptive conditional majority voting In our conditional majority voting strategy, we fix the number of samples n to be generated per question beforehand. This could be optimized by allowing n to be selected *adaptively*, based on the score from the verifier. Our initial experiments have shown promising results: the verifier's score on the probe sample was inversely correlated with the entropy among the answers in the subsequently generated samples. This suggests a meta-generation strategy where lower scores of the probe sample imply larger numbers of samples for voting.

ACKNOWLEDGMENTS

We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2024/017647. BP gratefully acknowledges Benjamin Crouzier (Tufa Labs) for supporting this research with a research grant and helpful discussions. We also thank Henryk Michalewski (Google DeepMind) for helpful discussions in the initial phase of the project.

REFERENCES

Marah I Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat S. Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Parul Chopra, Allie Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Dan Iter, Amit Garg, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Chen Liang, Weishung Liu, Eric Lin, Zeqi Lin, Piyush Madan, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Xia Song, Masahiro Tanaka, Xin Wang, Rachel Ward, Guanhua Wang, Philipp Witte, Michael Wyatt, Can Xu, Jiahang Xu, Sonali Yadav, Fan Yang, Ziyi Yang, Donghan Yu, Chengruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone. *CoRR*, abs/2404.14219, 2024. doi: 10.48550/ARXIV.2404.14219. URL `https://doi.org/10.48550/arXiv.2404.14219`.

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*, 2024.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models, 2023. URL `https://arxiv.org/abs/2211.15661`.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2018. URL `https://arxiv.org/abs/1610.01644`.

Amos Azaria and Tom Mitchell. The internal state of an llm knows when it's lying, 2023. URL `https://arxiv.org/abs/2304.13734`.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. *CoRR*, abs/2310.10631, 2023. URL `https://doi.org/10.48550/arXiv.2310.10631`.

Mohammad Beigi, Ying Shen, Runing Yang, Zihao Lin, Qifan Wang, Ankith Mohan, Jianfeng He, Ming Jin, Chang-Tien Lu, and Lifu Huang. Internalinspector i^2: Robust confidence estimation in llms through internal states, 2024. URL `https://arxiv.org/abs/2406.12053`.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, March 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i16.29720. URL `http://dx.doi.org/10.1609/aaai.v38i16.29720`.

Tuan-Dung Bui, Thanh Trong Vu, Thu-Trang Nguyen, Son Nguyen, and Hieu Dinh Vo. Correctness assessment of code generated by large language models using internal representations, 2025. URL `https://arxiv.org/abs/2501.12934`.

François Charton. Learning the greatest common divisor: explaining transformer predictions. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=cmcD05NPKa.

Chao Chen, Kai Liu, Ze Chen, Yi Gu, Yue Wu, Mingyuan Tao, Zhihang Fu, and Jieping Ye. Inside: Llms' internal states retain the power of hallucination detection, 2024. URL https://arxiv.org/abs/2402.03744.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 785–794. ACM, 2016. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

François Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. ARC prize 2024: Technical report. *CoRR*, abs/2412.04604, 2024. doi: 10.48550/ARXIV.2412.04604. URL https://doi.org/10.48550/arXiv.2412.04604.

François Chollet. On the measure of intelligence, 2019. URL https://arxiv.org/abs/1911.01547.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training, 2024. URL https://arxiv.org/abs/2309.17179.

Javier Ferrando, Gabriele Sarti, Arianna Bisazza, and Marta R. Costa-jussà. A primer on the inner workings of transformer-based language models, 2024. URL https://arxiv.org/abs/2405.00208.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001. doi: 10.1214/aos/1013203451. URL https://doi.org/10.1214/aos/1013203451.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, Olli Järviniemi, Matthew Barnett, Robert Sandler, Matej Vrzala, Jaime Sevilla, Qiuyu Ren, Elizabeth Pratt, Lionel Levine, Grant Barkley, Natalie Stewart, Bogdan Grechuk, Tetiana Grechuk, Shreepranav Varma Enugandla, and Mark Wildon. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai, 2024. URL https://arxiv.org/abs/2411.04872.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Wes Gurnee and Max Tegmark. Language models represent space and time, 2024. URL https://arxiv.org/abs/2310.02207.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer, 2009. ISBN 9780387848570. doi: 10.1007/978-0-387-84858-7. URL https://doi.org/10.1007/978-0-387-84858-7.

Alex Havrilla, Sharath Raparthy, Christoforus Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, and Roberta Raileanu. Glore: When, where, and how to improve llm reasoning via global and local refinements, 2024. URL https://arxiv.org/abs/2402.10963.

Jinwen He, Yujia Gong, Kai Chen, Zijin Lin, Chengan Wei, and Yue Zhao. Llm factoscope: Uncovering llms' factual discernment through inner states analysis, 2024. URL https://arxiv.org/abs/2312.16374.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024. URL https://arxiv.org/abs/2310.01798.

Siwon Kim, Sangdoo Yun, Hwaran Lee, Martin Gubri, Sungroh Yoon, and Seong Joon Oh. Propile: Probing privacy leakage in large language models, 2023. URL https://arxiv.org/abs/2307.01881.

Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. Hypertree proof search for neural theorem proving, 2022. URL https://arxiv.org/abs/2205.11491.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *CoRR*, abs/2305.20050, 2023. URL https://doi.org/10.48550/arXiv.2305.20050.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Cristian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, and et al. Gemma: Open models based on gemini research and technology. *CoRR*, abs/2403.08295, 2024. doi: 10.48550/ARXIV.2403.08295. URL https://doi.org/10.48550/arXiv.2403.08295.

Maciej Mikuła, Szymon Tworkowski, Szymon Antoniak, Bartosz Piotrowski, Albert Qiaochu Jiang, Jin Peng Zhou, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. Magnushammer: A transformer-based approach to premise selection, 2024. URL https://arxiv.org/abs/2303.04488.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. URL https://arxiv.org/abs/2410.05229.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. OpenWebMath: An open dataset of high-quality mathematical web text. *CoRR*, abs/2310.06786, 2023. URL https://doi.org/10.48550/arXiv.2310.06786.

Yudi Pawitan and Chris Holmes. Confidence in the reasoning of large language models, 2024. URL https://arxiv.org/abs/2412.15296.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models, 2019. URL https://arxiv.org/abs/1904.01557.

Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D. Manning. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback, 2023. URL https://arxiv.org/abs/2305.14975.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024. URL https://arxiv.org/abs/2312.08935.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. Naturalprover: Grounded mathematical proof generation with language models, 2022. URL https://arxiv.org/abs/2205.12910.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaid Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models, 2024. URL https://arxiv.org/abs/2406.16838.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. An empirical analysis of compute-optimal inference for problem-solving with language models. *CoRR*, abs/2408.00724, 2024. doi: 10.48550/ARXIV.2408.00724. URL https://doi.org/10.48550/arXiv.2408.00724.

Wei Xiong, Hanning Zhang, Nan Jiang, and Tong Zhang. An implementation of generative prm. https://github.com/RLHFlow/RLHF-Reward-Modeling, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process, 2024. URL https://arxiv.org/abs/2407.20311.

Fei Yu, Anningzhe Gao, and Benyou Wang. Ovm, outcome-supervised value models for planning in mathematical reasoning, 2024. URL https://arxiv.org/abs/2311.09724.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

A Reasoning-focused benchmarks

We evaluate LiLaVe and LiLaVe-based meta-generation strategies on four mathematical QA datasets. For each of them we select 1000 training examples to train a dataset-specific LiLaVe. We test on sets of 500–1319 examples, depending on the dataset. Below, we describe each of the benchmarks.

Question: Travis has 10000 apples, and he is planning to sell these apples in boxes. Fifty apples can fit in each box. If he sells each box of apples for $35, how much will he be able to take home?

Rationale: The total of boxes of apples is 10000 / 50 = 200. Therefore the total amount he can take home is 200 $\times$ $35 = 7000.
Answer: 7000

Figure 6: An example of a question from the GSM8K benchmark, followed by a couple of reasoning steps – a rationale for the final answer, which is always a number.

GSM8K introduced be Cobbe et al. (2021), contains grade school math problems with integer answers. To fit LiLaVe, we select 1000 examples from its training partition, and in evaluation use its full test set of 1319 questions. For answer generation, we use the standard 8-shot chain-of-thought prompt used by Wei et al. (2022). While widely used in LLM reasoning research, GSM8K is a relatively easy benchmark for modern LLMs. Also, it is likely leaked into LLM pretraining data. In Figure 6 there is an example of a question and solution from GSM8K.

GSM-Symbolic by Mirzadeh et al. (2024), has been developed to mitigate data contamination problem of GSM8K by semi-automatically generating questions from question templates obatined from GSM8K. Additional variants **p1** and **p2** of this dataset add one or two extra clauses to questions, increasing reasoning complexity. When evaluating LiLaVe-based generation strategies on GSM-Symbolic, we reuse GSM8K's training set for training the verifier. We also apply the same 8-shot chain-of-thought prompt that we use for GSM8K.

algebra_linear_1d is a subset of a synthetic benchmark introduced by Saxton et al. (2019) to evaluate performance of language models on a broad range of common mathematical tasks. algebra_linear_1d evaluates models on solving single-variable linear equations with integer solutions. We generate training and test sets, each containing 1000 examples. To query an LLM for answers, we use a simple zero-shot CoT prompt (see Figure 18). In Figure 7 there is an example of a question and solution from algebra_linear_1d.

MATH by Hendrycks et al. (2021) contains competition-level math problems. We train LiLaVe on 1000 selected training questions, and in evaluation we use its **MATH500** subset used by Lightman et al. (2023). For inference we use the 4-shot CoT prompt used by Lewkowycz et al. (2022). The final answers to MATH's questions include expressions such as polynomials, fractions, or complex numbers. To evaluate the generated answers, they need to be properly parsed and semantically compared with the gorund truth. For that, we reuse the final answer extractor from Gao et al. (2024).[5]

B Additional results

B.1 Temperature of generations

Our latent verifiers are trained on hidden states gathered from intermediate layers of the LLama 3.1 8B model, using generations sampled at different temperatures. In this section, we extend the temperature sensitivity analysis (discussed earlier in Section 4.1) beyond GSM8K to additional datasets. We test, how well do verifiers trained on various temperatures transfer to verifying generations sampled from different ones. The heatmap for GSM8K is identical to the one presented in Figure 5, we include it here again for the comparizon with results on other datasets.

Figure 8 shows heatmaps with the results of this analysis for hidden states of LLama 3.1 8B model and temperatures from the set $0.0, 0.2, 0.6, 0.8, 1.1$ on three datasets: GSM8K, algebra_linear_1d, and MATH500. For all temperatures, except the temperature 0, we generate 8 answers for each question. Each cell in a heatmap represents the mean AUC of XGBoost verifier on an appropriate

[5]Specifically, we reuse the code available at `https://github.com/EleutherAI/lm-evaluation-harness/blob/main/lm_eval/tasks/minerva_math/utils.py`

Question: Solve $-78 = 30 * r + 150 - 78$ for r.

Rationale: First, let's simplify the right-hand side of the equation by combining the constants:

$$-78 = 30 * r + 72$$

Next, let's subtract 72 from both sides of the equation to isolate the term with r:

$$-78 - 72 = 30 * r$$
$$-150 = 30 * r$$

Now, let's divide both sides of the equation by 30 to solve for r:

$$-150/30 = r$$
$$-5 = r$$

Answer: -5

Figure 7: An example of a question from the algebra_linear_1d benchmark, and a solution followed by a correct answer generated by Llama 3.1 8B.

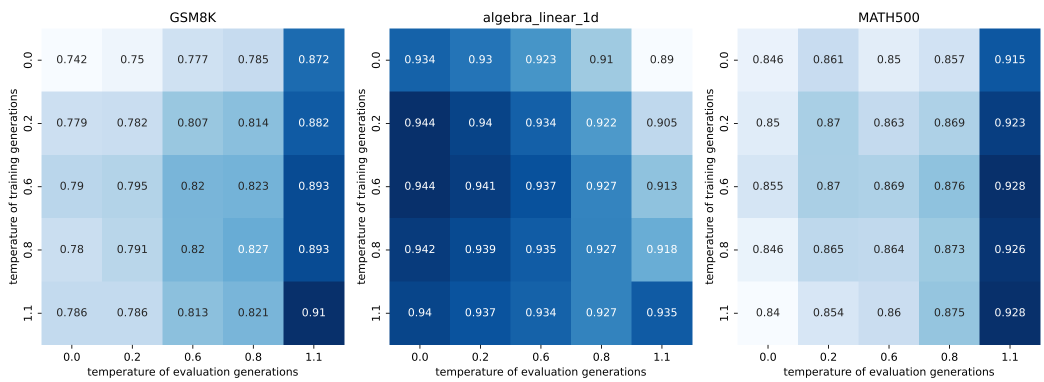

Figure 8: Transfer of performance (AUC) of LiLaVe trained and evaluated on hidden states extracted from answers generated from different temperatures, for three datasets. The base LLM used in this experiment is Llama 3.1 8B.

test set, averaged over 16 classifiers trained on hidden states from different layers $(2, 4, 8, 16)$ and different tokens $(2, 4, 8, 16$, counted from the end of the generated sequence).

Observations and conclusions for GSM8K are discussed in Section 4.1. Most importantly, the predictive performance of the verifier increases with both the training and evaluation temperatures. For algebra_linear_1d, most AUC values are very close to each other, but the variability trend differs from GSM8K: for a fixed training temperature, the verifier performs better when the evaluation temperature is lower. The heatmap for MATH follows a similar pattern to GSM8K, but the optimal training temperature for a fixed evaluation temperature is reached faster – AUC increases until $T = 0.6$ and then plateaus.

B.2 Choice of LiLaVe architecture

In all our main experiments, we instantiate our verifier as an XGBoost model Chen & Guestrin (2016). This choice is informed by our ablation experiments, which demonstrate its superior performance compared to alternative architectures. Additionally, XGBoost requires minimal hyperparameter tuning, making it a practical choice. We set the maximum tree depth to 5, selecting it as one of several equally well-performing candidates, and we use a learning rate (eta) of 0.1. All other hyperparameters

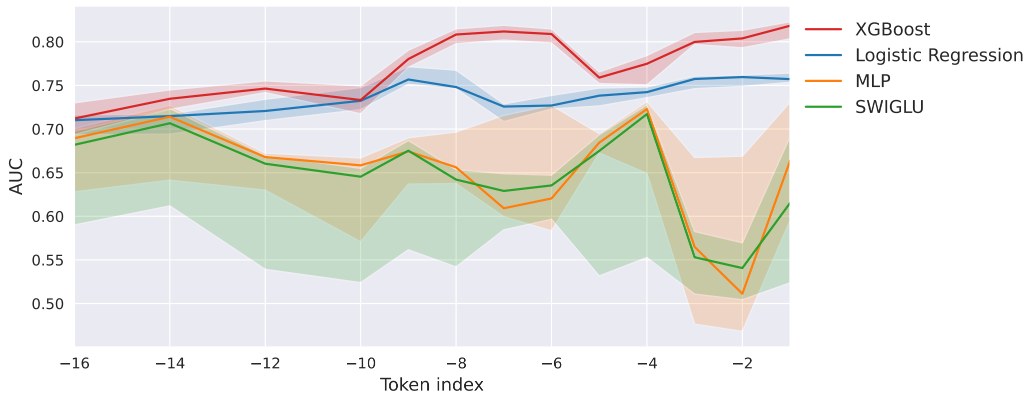

Figure 9: Ablation on the architecture of LiLaVe. Methods are compared on generations from Llama 3.1 8B on GSM8K dataset. The x-axis represents token index, while y-axis represents the value of AUC metric.

are the default ones. Training a single instance of XGBoost classifier in our setup is computationally efficient, taking only three minutes on our CPUs.

To validate our choice, we compare XGBoost against three other methods: Logistic Regression, a Multi-Layer Perceptron (MLP), and a SWIGLU-based MLP Shazeer (2020). Each method is trained on token-level features extracted from the token T ($T \in \{-1, -2, \ldots, -16\}$) and the layer L ($L \in \{-1, -2, \ldots, -5\}$). We run each method, for each T and L for 10 seeds. Figure 9 illustrates the comparative performance of these architectures, highlighting XGBoost's consistent superiority over the alternatives. For each line and plot The solid lines are medians and shadow region is nonsymmetric 90% confidence interval.

While hyperparameter tuning for MLP and SWIGLU could potentially improve their performance, we performed only a limited sweep over the number of layers and learning rates. However, the difficulty of tuning these models further underscores the advantage of XGBoost, which performs well out-of-the-box with minimal effort.

B.2.1 Hyperparameters of compared methods

Logistic Regression We use an `sklearn` implementation with a maximum iteration count of 1000 and balanced classes.

MLP The MLP consists of a hidden layer of size 16, and an output dimension of 1. It is trained using a logistic regression loss for 20 epochs with a batch size of 32. The model is optimized with Adam, using a learning rate of 10^{-4}.

SWIGLU This variant is a residual MLP using SWIGLU activations. It has two hidden layers of size 32, and an intermediate hidden dimension of 16. Like the standard MLP, it is trained with logistic regression loss for 20 epochs and a batch size of 32. The learning rate is 5×10^{-4}, and weight decay is set to 0.1.

XGBoost We in all our experiments we train XGBoost with the following hyperparameters:

- max_depth=10,
- eta=0.1,
- nrounds=30.

The rest of the hyperparameters use their default values set by the authors of the official XGBoost implementation.

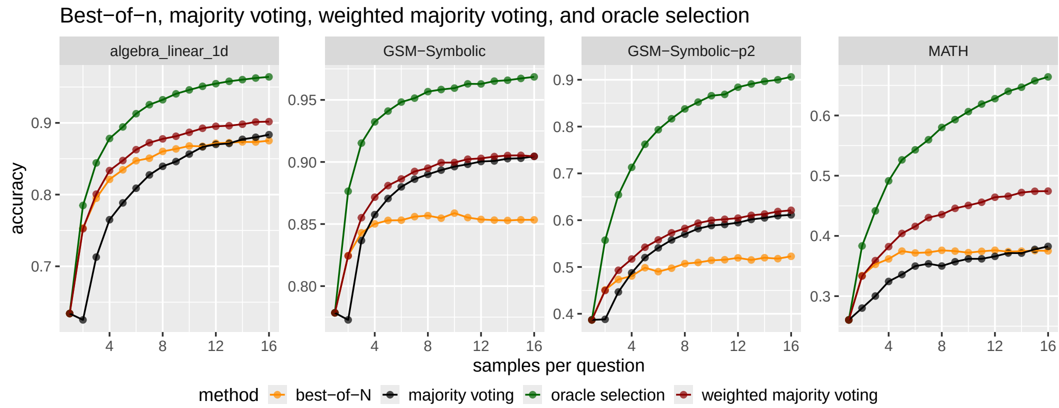

Figure 10: Comparison of meta-generation strategies to oracle selection.

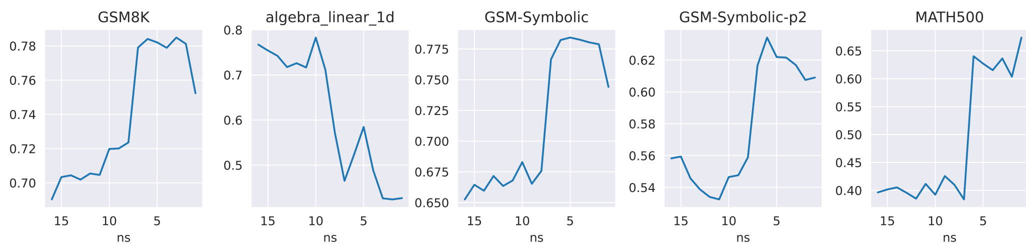

Figure 11: AUC of the sum of log probabilities over the answer suffix. The results correspond to zero-temperature generations from LLaMA 3.1 8B on the test sets of the respective datasets. The x-axis represents the length of the suffix considered.

All input sizes are equal to 4096, as this is the dimensionality of Llama 3.1 8B hidden states. For MLP and SWIGLU we report its test performance on an epoch after which validation performance is the best.

B.3 Accuracy of best-of-n given the oracle

We compare the results of meta generation strategies to oracle selection. This theoretical and practically impossible strategy assumes access to an omnipotent verifier, which always selects the correct answer from the set of LLM generated ones, if only such correct answer appears in this set. Otherwise, the strategy fails. Figure 10 presents the results of this experiment suggesting a gap between the best known meta-generation strategy and this theoretical upper bound, suggesting that further improving the verifiers has still a lot of potential.

B.4 Logprobs baseline results

This section provides a detailed analysis of the logprob-based estimator introduced in Section 4.2. The estimator is computed as the sum of log probabilities from Llama 3.1 8B over the final k tokens of a generated answer.

Figure 11 shows the AUC scores of this estimator across various (1-16) suffix lengths and datasets. For each dataset, we select the suffix length k that yields the highest AUC, and report these results in Table 1. Thus, this table reflects the best-performing suffix length for each dataset, giving an idealized upper bound on the estimator's performance.

In most datasets (GSM8K, GSM-Symbolic, GSM-Symbolic-p2, and MATH500), we observe a positive correlation between model confidence (measured by the sum of log probabilities) and answer correctness: higher confidence in the final tokens generally indicates a higher likelihood of correctness.

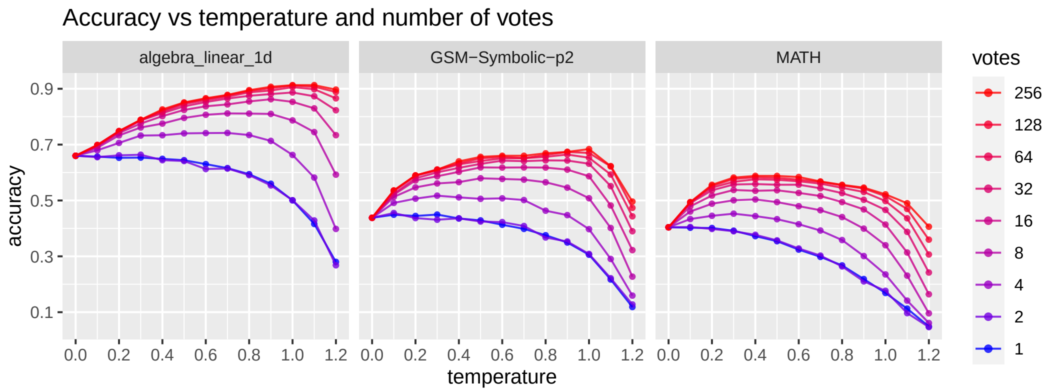

Figure 12: Accuracy of majority voting for different generation temperatures and number of votes. Base model is Llama 3.1 8B.

benchmark	Llama 3.1 8B	Gemma 2 2B	Phi-3.5-mini
GSM8K (test)	0.86	0.83	0.83
GSM-Symbolic	0.84	0.83	0.79
GSM-Symbolic-p2	0.78	0.84	0.78
algebra_linear_1d	0.93	0.86	0.96
MATH500	0.88	0.53	0.93

Table 2: Performance (AUC) of LiLaVe for three different base LLMs: Llama 3.1 8B, Gemma 2 2B, Phi-3.5-mini. LiLaVe preserves strong predictive performance across all the three models and all the benchmarks – with one exception of Gemma on MATH. The reason is likely because this model scored only ∼5% on MATH, which did not give enough positive examples for training LiLaVe.

Interestingly, an exception arises in the algebra_linear_1d dataset, where the relationship is inverted. Specifically, for short suffixes (lengths 1 to 8), AUC falls below 0.5. This implies that in this dataset, higher model confidence is actually indicative of a greater likelihood of error. In other words, a *lower* logprob sums serve as a better predictor of correctness, suggesting that the model exhibits overconfidence.

Since suffix length k is fixed, normalization of the sum is not necessary. We also verified that this sum-based estimator consistently outperforms the more commonly used average over all logprobs in the answer.

B.5 Optimal temperatures in majority voting

In this experiment, we evaluate the accuracy of majority voting (see Section 4.3) with respect to the temperature of generations and the number of votes. Results are presented in Figure 12. We observe that for different number of votes, different generation temperatures are optimal.

B.6 Impact of token and layer indices

In this experiment, we extend the results from Figure 1 to more datasets. The results are presented in Figure 13.

B.7 Performance of LiLaVe with other base LLMs

In Table 2 we present AUC performance of LiLaVe for three different base LLM: Llama 3.1 8B (used in the main experimental line presented in the main text), Gemma 2 2B, and Phi-3.5-mini.

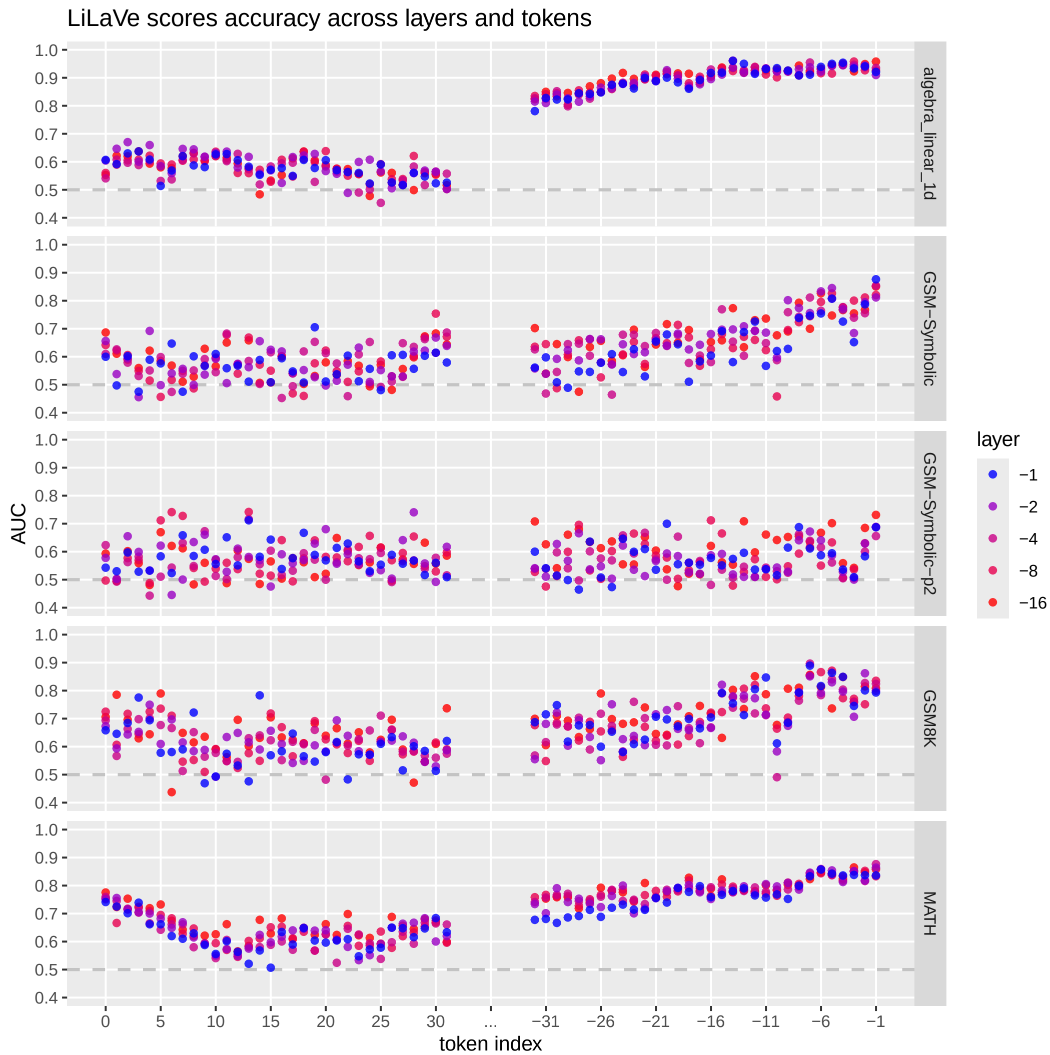

Figure 13: Predictive performance of LiLaVe on hidden states extracted from different tokens and layers.

benchmark	Phi-3.5-mini + Phi-LiLaVe	Phi-3.5-mini + Llama-LiLaVe
GSM8K (test)	**0.83**	**0.83**
GSM-Symbolic	0.79	**0.83**
GSM-Symbolic-p2	**0.78**	0.76
algebra_linear_1d	**0.96**	0.94
MATH500	**0.93**	0.91

Table 3: A comparison of two verification setups: the standard one, where responses generated by Phi-3.5-mini are scored based on the hidden states extracted from Phi during the generation (the middle column), *versus* responses generated by Phi, but later ingested by LLama 3.1 8B and scored based on the hidden states extracted from it (the right column). The latter setup in general performs worse – but not much worse, and for GSM-Symbolic actually better.

train \ test	GSM8K	algebra_linear_1d	MATH
GSM8K	**0.86**	0.87	0.84
algebra_linear_1d	0.75	**0.93**	0.71
MATH	0.72	0.53	**0.88**

Table 4: Transfer of performance (AUC) of LiLaVe trained and evaluated on different datasets. The base LLM used in this experiment is Llama 3.1 8B.

B.8 Performance of LiLaVe tested on responses originating from a different model

The standard mode of using LiLaVe is to apply it to the hidden states of the base LLM that generates the response. However, another setup is possible, where the responses are given without the hidden states and these are recreated by digesting the responses by an LLM for which a LiLaVe is available. In Table 3 we compare the results of two such approaches. The responses are coming from Phi-3.5-mini, and they are scored either by the LiLaVe trained for Phi (Phi-LiLaVe), or by Llama-LiLaVe, after retrieving the hidden states from Llama 3.1 8B that digested the Phi's responses. As can be seen, the latter setup gives good results, only slightly weaker than the original setup.

B.9 Transfer to other datasets

We evaluate the generalization ability of a verifier trained on one dataset when applied to another. Table 4 presents the AUC scores for different train-test combinations.

Our results indicate that while training and evaluating on the same dataset yields the highest performance, there is a significant cross-dataset generalization. For instance, a verifier trained on GSM8K achieves an AUC of 0.87 on algebra_linear_1d and 0.84 on MATH, which is better then baseline methods based on logprobs and self-reflection (see Table 1). Interestingly, the verifier trained on MATH generalizes less effectively, achieving only 0.53 AUC on algebra_linear_1d and 0.72 on GSM8K.

Overall, the results of this experiment suggest that some transferability across datasets exists, but we leave the exploration of transferability to other models for future work.

C Prompts

We present prompts used in our experiments in Figure 15, Figure 16, Figure 17, Figure 18, and Figure 19.

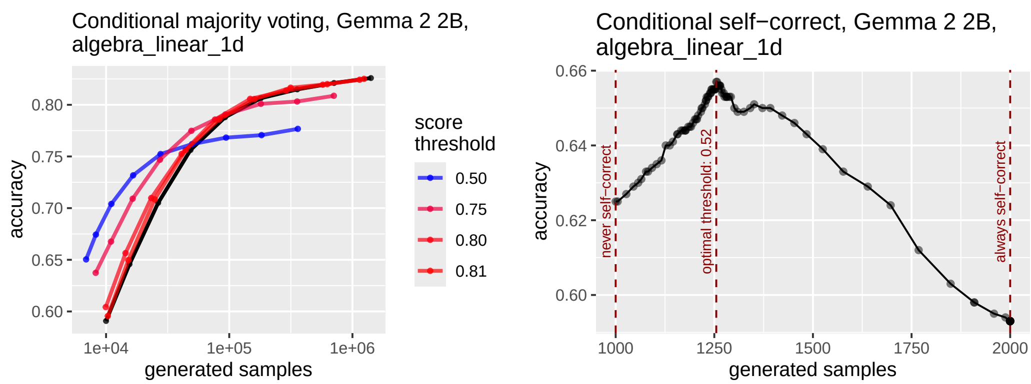

Figure 14: Conditional majority voting and conditional self-correction for **Gemma 2 2B** model, on **algebra_linear_1d** benchmark. The results are similarly good as for Llama 3.1 8B.

> The solution you provided contains mistakes and the answer is incorrect. Please, carefully review the solution and write a new, correct one.

Figure 15: Prompt used for the self-correction experiments.

D Efficiency

LLM-based verifiers typically require a large number of training examples, e.g. both models from Xiong et al. (2024) which we benchmark against, were trained on over 250k examples. In contrast, LiLaVe achieves comparable performance with just 5k samples per benchmark – two orders of magnitude less – making it a strong choice in data-scarce settings. Once the hidden states are collected, training LiLaVe takes only 15 minutes on a CPU, compared to the GPU-intensive fine-tuning required for LLM-based verifiers.

In terms of inference efficiency, scoring pre-generated Llama 3.1 8B's responses to 1319 GSM8K test questions using an LLM-based verifier (via the code from Xiong et al. (2024)) took nearly 20 minutes on an NVIDIA GH200 GPU. The same task (having the hidden states extracted) was completed by LiLaVe in only ~ 3.4s of wall clock time on CPUs of a Dell Precision 3561 laptop, yielding a $\sim 350\times$ speedup.

Of course, one could argue that, like other verifiers, LiLaVe still relies on a large generator to produce the answer to be verified. In scenarios where both generation and verification are benchmarked together, the speedup offered by LiLaVe may be limited to at most $2\times$, assuming verifier and generator are of similar size. However, even in this setting, LiLaVe provides important practical advantages. Unlike LLM-based verifiers that require GPUs, LiLaVe runs efficiently on CPU. This avoids the need to load large generator and verifier onto separate GPUs, which would double the required hardware, or to repeatedly load and unload model weights to and from GPU memory, which can significantly slow down the whole pipeline. It also introduces minimal compute overhead compared to LLM-based verifiers, which makes it much easier to integrate with more adaptive generation strategies.

> Please, rate on a scale of 1 to 10 how confident you are of the correctness of your answer.

Figure 16: Prompt used for the self-reflection confidence estimation.

Question: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?
Answer: There are 15 trees originally. Then there were 21 trees after some more were planted. So there must have been 21 - 15 = 6. The answer is 6.

Question: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?
Answer: There are originally 3 cars. 2 more cars arrive. 3 + 2 = 5. The answer is 5.

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
Answer: Originally, Leah had 32 chocolates. Her sister had 42. So in total they had 32 + 42 = 74. After eating 35, they had 74 - 35 = 39. The answer is 39.

Question: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?
Answer: Jason started with 20 lollipops. Then he had 12 after giving some to Denny. So he gave Denny 20 - 12 = 8. The answer is 8.

Question: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?
Answer: Shawn started with 5 toys. If he got 2 toys each from his mom and dad, then that is 4 more toys. 5 + 4 = 9. The answer is 9.

Question: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?
Answer: There were originally 9 computers. For each of 4 days, 5 more computers were added. So 5 * 4 = 20 computers were added. 9 + 20 is 29. The answer is 29.

Question: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?
Answer: Michael started with 58 golf balls. After losing 23 on tuesday, he had 58 - 23 = 35. After losing 2 more, he had 35 - 2 = 33 golf balls. The answer is 33.

Question: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?
Answer: Olivia had 23 dollars. 5 bagels for 3 dollars each will be 5 x 3 = 15 dollars. So she has 23 - 15 dollars left. 23 - 15 is 8. The answer is 8.

Question:

Figure 17: 8-shot prompt for GSM8K, GSM-Symbolic, and GSM-symbolic-p2 datasets.

Think step by step.

Figure 18: 0-shot prompt for the algebra_linear_1d dataset.

Problem: Find the domain of the expression $\frac{\sqrt{x-2}}{\sqrt{5-x}}$.
Solution: The expressions inside each square root must be non-negative. Therefore, $x-2 \geq 0$, so $x \geq 2$, and $5-x \geq 0$, so $x \leq 5$. Also, the denominator cannot be equal to zero, so $5-x > 0$, which gives $x < 5$. Therefore, the domain of the expression is $\boxed{[2,5)}$. Final Answer: The final answer is $\boxed{[2,5)}$. I hope it is correct.

Problem: If $\det \mathbf{A} = 2$ and $\det \mathbf{B} = 12$, then find $\det(\mathbf{A}\mathbf{B})$.
Solution: We have that $\det(\mathbf{A}\mathbf{B}) = (\det \mathbf{A})(\det \mathbf{B}) = (2)(12) = \boxed{24}$. Final Answer: The final answer is $\boxed{24}$. I hope it is correct.

Problem: Terrell usually lifts two 20-pound weights 12 times. If he uses two 15-pound weights instead, how many times must Terrell lift them in order to lift the same total weight?
Solution: If Terrell lifts two 20-pound weights 12 times, he lifts a total of $2 \cdot 12 \cdot 20 = 480$ pounds of weight. If he lifts two 15-pound weights instead for n times, he will lift a total of $2 \cdot 15 \cdot n = 30n$ pounds of weight. Equating this to 480 pounds, we can solve for n:

$$\begin{aligned} 30n &= 480 \\ \Rightarrow \qquad n &= 480/30 = \boxed{16} \end{aligned}$$

Final Answer: The final answer is $\boxed{16}$. I hope it is correct.

Problem: If the system of equations

$$\begin{aligned} 6x-4y &= a, \\ 6y-9x &= b. \end{aligned}$$

has a solution (x, y) where x and y are both nonzero, find $\frac{a}{b}$, assuming b is nonzero.
Solution: If we multiply the first equation by $-\frac{3}{2}$, we obtain

$$6y-9x = -\frac{3}{2}a.$$

Since we also know that $6y - 9x = b$, we have

$$-\frac{3}{2}a = b \Rightarrow \frac{a}{b} = \boxed{-\frac{2}{3}}.$$

Final Answer: The final answer is $\boxed{-\frac{2}{3}}$. I hope it is correct.

Problem:

Figure 19: 4-shot prompt for the MATH dataset.