MULTI-AGENT VERIFICATION: SCALING TEST-TIME COMPUTE WITH MULTIPLE VERIFIERS (ABRIDGED)*

Shalev Lifshitz
ArdaLabs.AI
shalev@ardalabs.ai

Sheila A. McIlraith
University of Toronto
Vector Institute for AI
sheila@cs.toronto.edu

Yilun Du
Harvard University
ydu@seas.harvard.edu

ABSTRACT

By utilizing more computational resources at test-time, large language models (LLMs) can improve without additional training. One common strategy uses *verifiers* to evaluate candidate outputs. In this work, we propose a novel scaling dimension for test-time compute: *scaling the number of verifiers*. We introduce Multi-Agent Verification (MAV) as a test-time compute paradigm that combines multiple verifiers to improve performance. We propose using Aspect Verifiers (AVs), off-the-shelf LLMs prompted to verify different aspects of outputs, as one possible choice for the verifiers in a MAV system. AVs are a convenient building block for MAV since they can be easily combined without additional training. Moreover, we introduce BoN-MAV, a simple multi-agent verification algorithm that combines best-of-n sampling with multiple verifiers. BoN-MAV demonstrates stronger scaling patterns than self-consistency and reward model verification, and we demonstrate both weak-to-strong generalization, where combining weak verifiers improves even stronger LLMs, and self-improvement, where the same base model is used to both generate and verify outputs. Our results establish scaling the number of verifiers as a promising new dimension for improving language model performance at test-time.

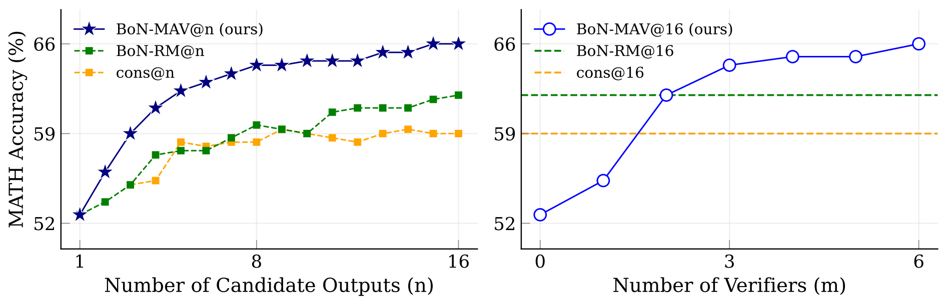

Figure 1: **Scaling test-time compute along two dimensions.** *Left:* Increasing the number of candidate outputs (n) and comparing three test-time methods: best-of-n with multi-agent verification (BoN-MAV@n), best-of-n with reward model verification (BoN-RM@n), and self-consistency (cons@n). *Right:* Increasing the number of verifiers (m) when selecting between $n = 16$ candidate outputs (BoN-MAV@16) surpasses the performance of reward model verification (BoN-RM@16) and self-consistency (cons@16). All candidate outputs are sampled from Gemini-1.5-Flash on the MATH benchmark (Hendrycks et al., 2021).

1 INTRODUCTION

Scaling the size of large language models (LLMs) and their training datasets has driven remarkable progress in artificial intelligence (Brown et al., 2020; Chowdhery et al., 2023; Hoffmann et al., 2022). However, the growing cost of scaling model size and obtaining unseen high-quality pretraining data has sparked growing interest in methods that improve LLM performance without simply scaling parameters or data. Among these, a promising new direction has emerged: *scaling test-time compute*, where models spend more computational resources during inference—much like humans spend more time thinking through harder problems.

*Full version of the paper is available at https://arxiv.org/abs/2502.20379, and see https://ardalabs.ai/MultiAgentVerification for the paper webpage.

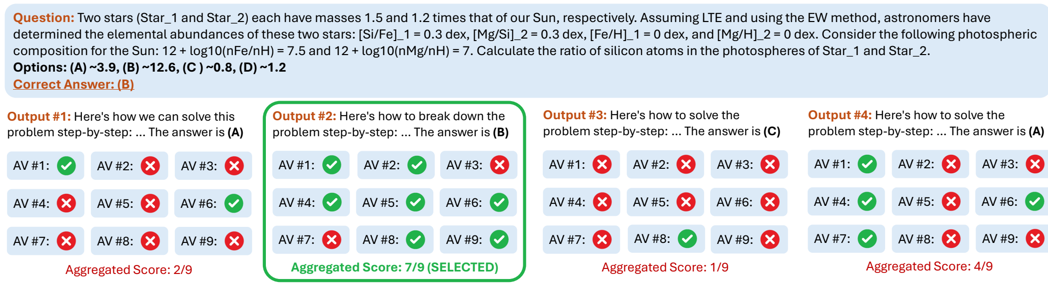

Figure 2: **Illustration of the BoN-MAV algorithm.** BoN-MAV combines best-of-n sampling with multi-agent verification: First, n candidate outputs are sampled from a generator LLM. Then, each output is evaluated by a set of Aspect Verifiers (AVs) that produce binary approvals. Finally, the candidate with the most approvals is selected as the final answer. See Section 2.3 for algorithm details.

A common strategy for scaling test-time compute is *best-of-n sampling* (Stiennon et al., 2020; Cobbe et al., 2021; Nakano et al., 2021), where n candidate outputs are sampled from a *generator* LLM and a *verifier* model scores each candidate output based on its quality or correctness. The highest-scoring output is then selected. Under this strategy, the amount of test-time compute can be scaled up by increasing the number of sampled outputs. However, in this work, we propose a new orthogonal scaling dimension: *scaling the number of verifiers*. We introduce **Multi-Agent Verification (MAV)**, a test-time compute paradigm that combines multiple verifiers to improve performance.

Typically, verifiers are implemented as reward models which are trained using reinforcement learning from human feedback (Christiano et al., 2017; Stiennon et al., 2020; Ouyang et al., 2022; Bai et al., 2022a). However, relying on reward models as verifiers introduces two crucial limitations for multi-agent verification: (1) each reward model has to be trained on expensive curated preference data, and (2) there is no straightforward way to combine scores generated by heterogeneous reward models trained on different datasets (they produce uncalibrated scores). These limitations make reward models poorly suited for multi-agent verification and restrict our ability to simply scale up the number and type of verifiers at test-time.

To address these limitations and enable scalable multi-agent verification, we propose using **Aspect Verifiers (AVs)** — off-the-shelf LLMs prompted to verify specific aspects of candidate outputs through binary True/False approvals. This approach is motivated by the observation that internet data contains abundant examples of humans providing binary evaluations with feedback (e.g., educational assessments, academic peer reviews, online forums, and automated code tests), which suggests that language models may be naturally suited for binary verification. Unlike reward models, AVs do not require additional training since producing binary approvals falls naturally within the training distribution of their base LLMs, and their binary outputs can be easily combined across multiple models through simple voting mechanisms. Thus, the number and type of aspect verifiers can be easily scaled up without additional training. We note that aspect verifiers are just one possible implementation choice for the verifiers in a MAV system, which address the two key limitations of typical reward model verifiers.

By aggregating binary signals across a diverse set of aspect verifiers, we can leverage the growing ecosystem of language models to produce a more robust verification signal. Each verifier can focus on different aspects of outputs like mathematical correctness or logical soundness, and employ different verification strategies such as direct yes/no approval, step-by-step analysis, solution rephrasing, or edge case checking. Thus, a diverse set of aspect verifiers can be obtained by varying three key axes: the base LLM, the aspect to verify, and the verification strategy.

To investigate scaling multi-agent verification, we introduce BoN-MAV as a specific algorithm which combines best-of-n sampling with aspect verifiers. This is one implementation of a MAV algorithm, combining traditional best-of-n sampling with multiple verifiers. Given an input, BoN-MAV (1) samples n outputs from a generator LLM, (2) collects binary approvals from a set of m aspect verifiers, and (3) selects the output with the most approvals. We investigate scaling test-time compute with this approach along two orthogonal dimensions: the traditional dimension of increasing the number of sampled candidate outputs n, and our novel test-time scaling dimension of increasing the number of verifiers m. We find that using multiple diverse verifiers to select between candidate outputs is an effective strategy, and that performance improves as we use more verifiers.

More specifically, across multiple domains and LLMs, BoN-MAV demonstrates more effective scaling patterns when we increase the number of sampled outputs, compared to best-of-n with reward model verification (Stiennon et al., 2020; Cobbe et al., 2021) and self-consistency (Wang et al., 2022; Li et al., 2022b; Thoppilan et al., 2022; Lewkowycz et al., 2022). We also demonstrate *weak-to-strong generalization* (Burns et al., 2023), whereby combining many small aspect verifiers can improve the performance of even stronger generator LLMs, and we show that BoN-MAV enables *self-improvement* by using the same base LLM for both the generator and set of aspect verifiers. Since BoN-MAV is just one simple approach to multi-agent verification, we expect that substantial improvements can be achieved using alternative methods.

Overall, our paper makes the following contributions:

- **(1)** We introduce Multi-Agent Verification (MAV) as a new test-time paradigm that combines multiple verifiers at test-time, opening a novel scaling dimension: *scaling the number of verifiers*.
- **(2)** We propose Aspect Verifiers (AVs), off-the-shelf LLMs which require no additional training and naturally support combining verification signals from multiple heterogeneous verifiers using voting mechanisms.
- **(3)** We demonstrate that BoN-MAV, a simple multi-agent verification algorithm which combines best-of-n with aspect verifiers, improves the performance of various generator LLMs as we scale up the number and type of aspect verifiers.

2 Multi-Agent Verification

Multi-Agent Verification (MAV) is a test-time compute paradigm where multiple verifiers are combined to evaluate outputs from a generator LLM. To implement a MAV algorithm, we must address two questions: **(1)** What type of verifiers can be easily combined and scaled up in number without additional training? **(2)** How should we aggregate verification signals from multiple verifiers? In this section, we propose answers to these questions and describe one simple implementation of a multi-agent verification algorithm called BoN-MAV. We discuss future directions for alternative multi-agent verification algorithms in Appendix D.

In Section 2.1, we propose Aspect Verifiers (AVs) as a convenient building block for MAV, since they require no additional training and naturally support combining multiple verification signals. In Section 2.2, we describe our approach to aggregating signals across multiple AVs. In Section 2.3, we outline the BoN-MAV algorithm, which combines best-of-n sampling with aspect verifiers. Section 2.4 proposes verifier engineering to select relevant verifiers for specific domains or tasks.

2.1 Aspect Verifiers

In the context of test-time computation with LLMs, a *verifier* typically refers to a model that evaluates the quality or correctness of an output sampled from a generator LLM. Here, we ask: *What type of verifiers can be easily combined and scaled up in number without additional training?*

Prior works have largely focused on using neural reward models as verifiers (Stiennon et al., 2020; Cobbe et al., 2021; Snell et al., 2024). However, these models present key challenges for scaling multi-agent verification. First, each reward model requires training on expensive curated preference data to produce reliable reward scores (Stiennon et al., 2020). Second, while ensembles of homogeneous reward models (identical model initializations trained on the same data but with different random seeds) have been proposed as a way to mitigate overoptimization (Coste et al., 2023; Eisenstein et al., 2023; Gao et al., 2023a), there is no straightforward way to combine scores from heterogeneous reward models trained on different datasets. This second limitation arises because scores from different reward models are uncalibrated——they operate on different numerical scales based on their distinct training setups. We wish to simply scale up the number and type of verifiers without additional training.

We propose Aspect Verifiers (AVs) as one possible implementation choice for the verifiers in a MAV system, which address the two mentioned limitations of typical reward model verifiers. AVs are off-the-shelf LLMs prompted to evaluate specific aspects of candidate outputs and produce binary True/False approvals. Unlike reward models, they require no additional training since binary evaluation is a natural task for LLMs (the internet contains abundant examples of humans providing binary approvals with explanation, such as educational assessments, academic peer reviews, or online forums), and their binary approvals can be easily combined through voting mechanisms, even when AVs are based on completely different models or training data. Moreover, since AVs are based on LLMs, they can produce CoT reasoning (Wei et al., 2021) to analyze outputs step-by-step before producing an approval, similar to recent work on generative reward models (Zhang et al., 2024b; Mahan et al., 2024). Using aspect verifiers, we can easily scale up the number and type of verifiers which may be based on different LLMs, training algorithms, architectures, data, or prompts.

Aspect Verifiers can be configured along three axes: (1) The base LLM—which model acts as the verifier (e.g., GPT-4o-mini or Gemini-1.5-Flash), (2) The aspects to verify—what qualities of the candidate output the verifier is prompted to evaluate (e.g., mathematical correctness, logical soundness, factuality, etc.), (3) The verification strategy—how the verifier reaches its decision (e.g., direct approval, going over the output step-by-step, rephrasing, checking edge cases, etc.). For example, an aspect verifier could be implemented as GPT-4o-mini evaluating the mathematical correctness of an output from a generator LLM by going over it step-by-step. By varying these three axes, we can create a diverse set of aspect verifiers with differing capabilities. Figure 6 illustrates how multiple aspect verifiers can evaluate a single candidate output, and Appendix F contains a full list of the verifiers used in this work and the prompts for each.

2.2 Combining Aspect Verifiers

With aspect verifiers as our building block, we ask: *How can we effectively aggregate verification signals across multiple AVs?* We take the simplest possible approach in our experiments: each binary True/False approval is a single vote, and the aggregated score for a candidate output is the sum of the positive votes from all AVs. That is, the aggregated verification score is the sum of the individual binary scores from each verifier:

$$\text{AggScore}(o^{(i)}) = \frac{1}{|\mathcal{M}|} \sum_{v \in \mathcal{M}} \text{BinaryScore}_v(o^{(i)}), \tag{1}$$

where $o^{(i)} \in \mathcal{O}$ is the ith candidate output from the set of sampled outputs $\mathcal{O}$, $\mathcal{M}$ is the set of aspect verifiers, and $\text{BinaryScore}_v : \mathcal{O} \to \{0, 1\}$ maps a candidate output from $\mathcal{O}$ to the binary approval produced by verifier $v \in \mathcal{M}$ for that output. This voting strategy gives equal weight to all verifiers in the final aggregated score, and it proves remarkably effective in our experiments (see Section 3.1). However, future works could investigate more sophisticated aggregation strategies such as grouping verifiers by aspect and then voting across aspects, or having aspect verifiers debate with each other (Du et al., 2023) before producing an approval. We discuss these and other potential directions for future work in Appendix D.

2.3 BoN-MAV

Best-of-n (BoN) sampling is a test-time optimization technique (Stiennon et al., 2020; Cobbe et al., 2021; Nakano et al., 2021) where n candidate outputs are sampled from a generator LLM, each candidate is scored by a verifier model, and the highest-scoring output is selected. We introduce BoN-MAV as a simple multi-agent verification algorithm that combines best-of-n sampling with aspect verifiers. It uses the simple aggregation strategy from Equation 1 and consists of three steps: (1) sampling n candidate outputs from a generator LLM, (2) collecting binary approvals from a set of m aspect verifiers, and (3) selecting the output with the most approvals. That is, $\hat{i} = \arg\max_{0<i<n} \left(\text{AggScore}(o^{(i)})\right)$ where $o^{(i)}$ is the ith candidate output, n is the total number of sampled candidate outputs, and $\hat{i}$ is the index of the output with the highest aggregated score (the selected output). Figure 2 illustrates how BoN-MAV can select between a set of candidates.

Using BoN-MAV, we can increase test-time computation by sampling more candidate outputs (increasing n) and by querying more verifiers (increasing $m = |\mathcal{M}|$), where test-time computation can be easily parallelized during generation as well as verification. In addition, BoN-MAV represents just one specific approach to multi-agent verification, and more nuanced aggregation algorithms or alternatives to aspect verifiers could further enhance performance (see Appendix D for discussion).

2.4 Verifier Engineering

Using aspect verifiers, we can create a diverse pool of verifiers with different capabilities. However, not all verifiers are equally relevant for every domain (e.g., math, coding, general knowledge). Thus, we propose *verifier engineering* as a process to select a subset of verifiers most effective for a particular domain (similar to prompt engineering, where prompts are engineered for specific domains or tasks).

	MATH			MMLU-Pro			GPQA (diamond)			HumanEval		
Generator LLM	B-MAV	Cons	RM	B-MAV	Cons	RM	B-MAV	Cons	RM	B-MAV	Cons	RM
Gemini-1.5-Flash	**66.0**	59.0	61.7	**66.7**	63.3	60.7	42.0	40.0	**46.0**	**80.0**	79.0	79.0
Gemini-1.5-Pro	**72.7**	70.3	71.0	**72.3**	71.7	69.3	**49.0**	45.0	**49.0**	**88.0**	84.0	**88.0**
GPT-4o-mini	73.0	**74.7**	72.3	**67.0**	63.7	62.7	**50.0**	48.0	44.0	84.0	**87.0**	85.0
GPT-4o	76.3	77.3	**80.7**	75.7	**76.3**	72.7	**59.0**	**59.0**	58.0	92.0	**95.0**	92.0
Mistral-7B	**26.0**	22.0	21.7	**36.7**	25.7	31.0	36.0	32.0	**37.0**	**59.0**	46.0	52.0
Llama-3.1-8B	**61.7**	61.0	54.7	**59.3**	55.3	51.3	**43.0**	36.0	41.0	**75.0**	62.0	64.0
Gemma-2-9B	**58.7**	51.7	55.0	**57.7**	54.3	54.7	34.0	36.0	**38.0**	32.0	25.0	**51.0**
Gemma-2-27B	**62.3**	55.7	59.3	**62.0**	58.3	60.0	**41.0**	40.0	**41.0**	**76.0**	66.0	**76.0**

Table 1: **Best-of-n with Multi-Agent Verification (BoN-MAV) across models and domains.** Performance (accuracy %) comparison of three test-time verification methods using $n = 16$ candidate outputs: the BoN-MAV algorithm (labeled as B-MAV in the table), reward model verification (RM), and self-consistency (Cons). Results are shown for eight generator LLMs across four domains, with BoN-MAV on each domain using the domain-specific aspect verifier subset $\mathcal{M}^d$. BoN-MAV outperforms self-consistency in nearly all cases, and generally outperforms RM except on GPQA and HumanEval, where BoN-MAV and RM are comparable.

We engineer domain-specific sets of verifiers by first creating a diverse initial set $\mathcal{M}$ and then selecting the subset $\mathcal{M}^d \subseteq \mathcal{M}$ which contains the most relevant verifiers for domain d. Specifically, for each domain d, we select the subset $\mathcal{M}^d \subseteq \mathcal{M}$ which maximizes the average performance across all generator LLMs evaluated on a validation set. Our current approach keeps the engineered set of verifiers fixed for all questions in a domain, but future works could explore dynamically customizing verifiers for particular questions, as we discuss in Appendix D.

3 Experiments

In our experiments, we investigate scaling test-time compute along two orthogonal dimensions: the traditional dimension of increasing the number of sampled candidate outputs n, and our novel test-time scaling dimension of increasing the number of verifiers m. We aim to address the following questions: **(1)** How well does multi-agent verification improve performance across diverse domains and various generator LLMs? **(2)** Can multi-agent verification facilitate weak-to-strong generalization and self-improvement? **(3)** How important is engineering a domain-specific set of verifiers and what are the important design choices? To address these questions, we evaluate the BoN-MAV algorithm described in Section 2 on the following four domains:

- **Mathematics.** The MATH dataset (Hendrycks et al., 2021) consists of competition-level math questions at five difficulty levels. For our experiments, we randomly sample 400 questions from the test set across all five levels: 100 for validation and 300 for testing.
- **General Knowledge & Reasoning.** MMLU-Pro (Wang et al., 2024c) is an enhanced version of the popular MMLU benchmark (Hendrycks et al., 2020) which features more challenging, reasoning-focused questions and expands the multiple-choice set from four to ten options. As with MATH, we sample 100 questions for validation and 300 for testing.
- **Graduate-Level Reasoning.** The GPQA dataset (Rein et al., 2023) consists of graduate-level, multiple-choice questions in biology, physics, and chemistry. For our experiments, we utilize GPQA's "diamond" subset — a collection of 198 high-quality and extremely challenging questions. We sample 98 questions for validation and 100 for testing.
- **Coding.** HumanEval (Chen et al., 2021) is a widely-used benchmark consisting of 164 Python programming questions. We sample 64 questions for validation and 100 for testing.

3.1 MAV Enables Scaling Along Two Dimensions

Baselines. Here, we investigate how BoN-MAV scales with the number of candidate outputs and number of verifiers. We compare Best-of-n sampling with Multi-Agent Verification (BoN-MAV) against two established test-time compute methods: (1) best-of-n sampling with reward model verification (Stiennon et al., 2020; Cobbe et al., 2021; Nakano et al., 2021), where we use a trained neural reward model as the external verifier to select the highest-scoring candidate output, and (2) self-consistency (Wang et al., 2022; Li et al., 2022b; Thoppilan et al., 2022; Lewkowycz et al., 2022),

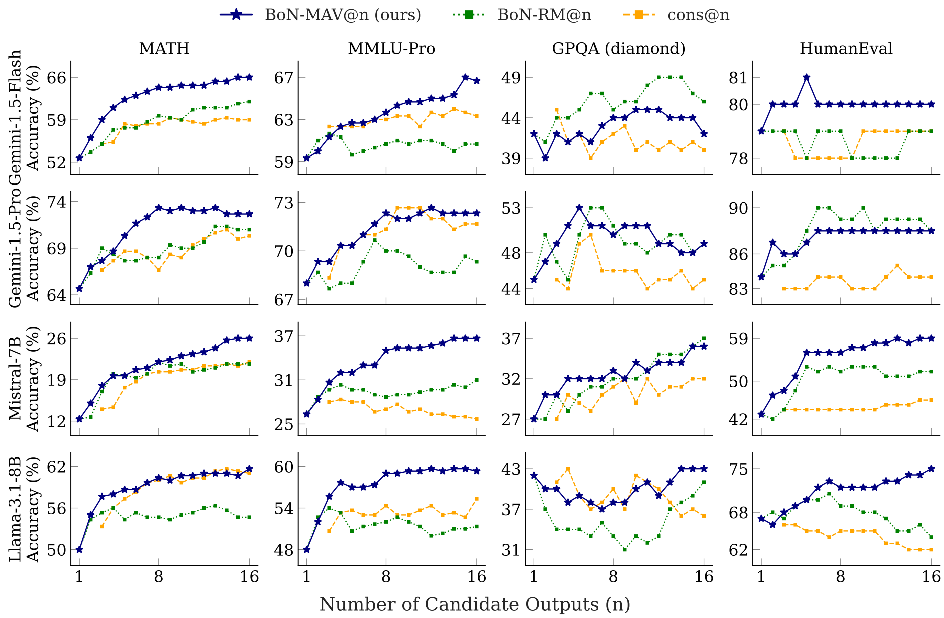

Figure 3: **Scaling the number of candidate outputs.** Performance (accuracy %) of test-time compute methods as we increase the number of sampled candidate outputs (n), shown for four generator LLMs (Gemini-1.5-Flash, Gemini-1.5-Pro, Mistral-7B, and Llama-3.1-8B) across all evaluation domains. The BoN-MAV algorithm demonstrates more effective scaling patterns than self-consistency (cons@n) across all domains, and stronger scaling than reward model verification (BoN-RM@n) except on GPQA (diamond).

which selects the most common answer from the set of candidates outputs. For reward model verification, we use the current top-performing open-source 8B reward model on RewardBench (Lambert et al., 2024). See Appendix F.3 for more details.

Verifier Engineering For our experiments, we implement the verifier engineering method described in Section 2.4. To create our initial diverse pool $\mathcal{M}$ of 20 aspect verifiers, we vary the three key axes that define aspect verifiers: (1) Base model: Gemini-1.5-Flash or GPT-4o-mini, (2) Aspect to verify: Mathematical correctness, logical soundness, factuality, etc., and (3) Verification strategy: Direct approval, step-by-step verification, solution rephrasing, edge case checking, etc. From this pool, we then select domain-specific subsets $\mathcal{M}^d \subseteq \mathcal{M}$ that maximize average performance across all generator LLMs on the corresponding validation sets. The complete list of verifiers and the domain-specific subsets are detailed in Table 4. We choose Gemini-1.5-Flash and GPT-4o-mini as the base LLMs for our aspect verifiers since they are cost-effective for large-scale verification and enable us to demonstrate that combining multiple weaker verifiers can improve the performance of even stronger generator LLMs (Section 3.2).

Quantitative Results. We evaluate BoN-MAV across four domains using eight generator LLMs (four closed-source and four open-source). For each model, we sample $n = 16$ candidate outputs per question and compare between best-of-n with Multi-Agent Verification (BoN-MAV), best-of-n with reward model verification (BoN-RM), and self-consistency (cons). As shown in Table 1, BoN-MAV outperforms self-consistency in nearly all cases, and outperforms reward model verification on MATH and MMLU-Pro, while achieving comparable results on GPQA (diamond) and HumanEval.

Qualitative Examples. Figure 6 illustrates how multiple aspect verifiers can be used to evaluate a single candidate output. The first aspect verifier uses direct yes/no approval without step-by-step thinking and incorrectly approves the output while additional aspect verifiers, using the same base model but with more thorough verification strategies, successfully identify the error. Additional examples are provided in Appendix H. Note that for the purposes of illustration, we visualize slightly different sets of verifiers than the final domain-specific sets used in our experiments.

Scaling the Number of Candidate Outputs. In Figure 3, we show the scaling patterns for various generator LLMs as we increase the number of sampled candidate outputs (n). Matching the results in Table 1, BoN-MAV demonstrates more effective scaling patterns than self-consistency across all domains, and stronger scaling than reward model verification on MATH and MMLU-Pro while achieving comparable scaling patterns on GPQA (diamond) and HumanEval.

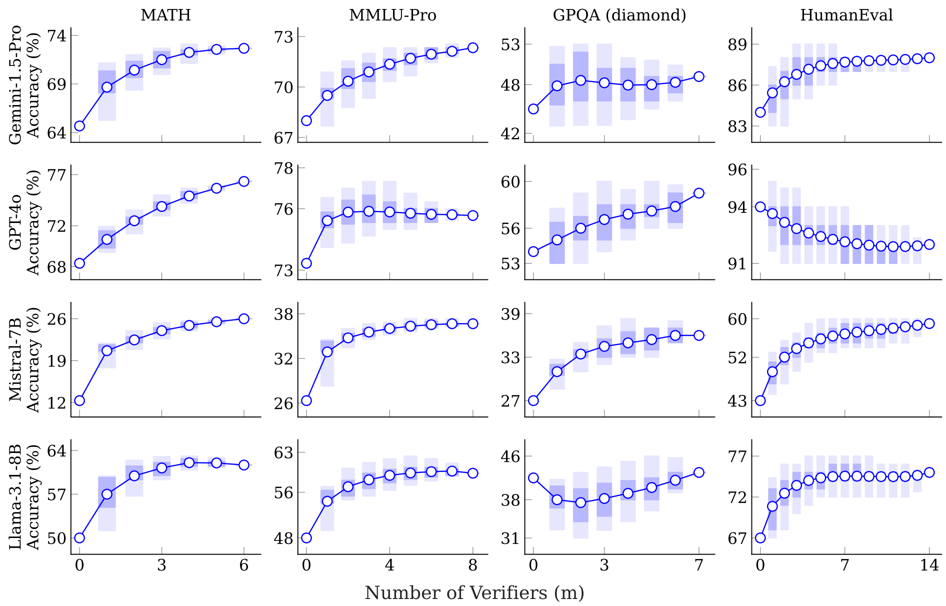

Figure 4: **Scaling the number of verifiers.** Performance (accuracy %) of BoN-MAV as we increase the number of verifiers (m) up to domain-specific subsets $\mathcal{M}^d$ (detailed in Section 3.1). For each m, we plot the performance of BoN-MAV averaged across all possible combinations of m verifiers drawn from $\mathcal{M}^d$, with shading indicating the spread of observed values — dark blue shows the 25-75th percentile range (middle 50%) while light blue shows the 5-95th percentile range (90% of outcomes). The leftmost point ($m = 0$) represents pass@1 accuracy without verification while the rightmost point ($m = |\mathcal{M}^d|$) uses all verifiers in the domain-specific set. Results demonstrate that increasing the number of verifiers is a promising test-time scaling dimension, with gains of up to 10% for large LLMs and up to 20% for small ones, even when stronger generator LLMs (Gemini-1.5-Pro, GPT-4o) are verified by our weaker aspect verifiers. We observe domain and model-dependent variation and some diminishing returns at higher verifier counts, and we expect better-engineered verifiers to unlock even stronger scaling patterns.

Scaling the Number of Verifiers. Multi-Agent Verification introduces a powerful new dimension for scaling test-time compute: *scaling the number of verifiers*. In Figure 4, we show how accuracy tends to improve as we increase the number of verifiers m from zero verifiers up to the full domain-specific subset $\mathcal{M}^d$. For each value of $m \in \{0, 1, 2, ..., |\mathcal{M}^d|\}$, we plot the average accuracy across all possible combinations of m verifiers drawn from $\mathcal{M}^d$, with the shaded regions indicating the spread of observed values. Note that Figure 1 shows just one randomly selected sequence of verifiers for illustration, rather than averaging across all possible combinations like in Figure 4.

Our results demonstrate that scaling verifier count is a promising new dimension for improving model performance at test-time. In most cases, accuracy improves as we add verifiers, with performance gains of up to 10% for large LLMs and up to 20% for small ones. Notably, performance gains persist even when strong generator LLMs (Gemini-1.5-Pro, GPT-4o) are verified by combinations of our weaker verifiers (Gemini-1.5-Flash, GPT-4o-mini), supporting our findings about weak-to-strong generalization in Section 3.2. However, the magnitude and pattern of improvement varies and, in some cases, accuracy initially decreases before improving with additional verifiers. We expect better-engineered verifiers to unlock even stronger scaling patterns.

Scaling Up to 256 Candidate Outputs. We extend our analysis to even larger scales by sampling 256 candidate outputs from Gemini-1.5-Flash on MATH. In Figure 5, we plot accuracy as a function of both the number of sampled candidate outputs n (left) and the total compute budget (right). The left plot demonstrates that BoN-MAV consistently improves with additional samples, while reward model verification and self-consistency plateau early on. Starting at 52.7% base accuracy, the baselines plateau around 61% while BoN-MAV continues to 69%—nearly double the improvement. The right plot shows computational efficiency by comparing accuracy against the total compute budget, measured as the combined number of queries to both the generator and verifier models. At low compute budgets, the overhead of querying multiple verifiers with BoN-MAV means we can sample fewer candidate solutions, leading to initially worse performance than the baselines. However, once we have sufficient compute, BoN-MAV significantly outperforms both baseline methods.

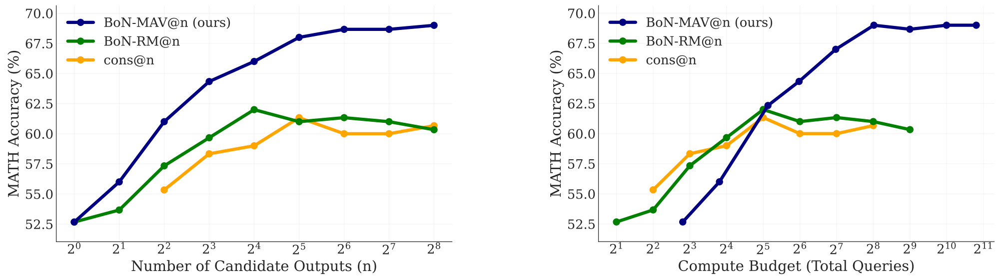

Figure 5: **Scaling to 256 candidate outputs.** Comparison of different test-time verification methods on MATH (Hendrycks et al., 2021) as we increase the number of candidate outputs sampled from Gemini-1.5-Flash up to 256. *Left:* Accuracy (%) versus number of sampled outputs n. BoN-MAV consistently improves with additional samples while reward model verification (BoN-RM) and self-consistency (cons) plateau much earlier. *Right:* Accuracy (%) versus total compute budget (number of queries to both generator LLM and each verifier). While BoN-MAV initially underperforms due to the overhead of querying multiple verifiers limiting the number of candidate outputs we can sample, it significantly outperforms both baseline methods once given sufficient compute to leverage multiple verifiers effectively. Note that both x-axes are on a log scale.

3.2 MAV Enables Weak-to-Strong Generalization and Self-Improvement

Weak-to-Strong Generalization. Prior work has shown that weak supervisors can improve the performance of strong pretrained models (Burns et al., 2023). Here, we show that multi-agent verification can be used to enhance the performance of strong generator LLMs by combining weaker verifiers. As shown in Table 2, our strongest generators (Gemini-1.5-Pro and GPT-4o) show substantial improvements over their base pass@1 accuracy when using verifiers based on weaker models (Gemini-1.5-Flash and GPT-4o-mini), and Figure 4 shows how the performance of Gemini-1.5-Pro and GPT-4o changes as we scale the number of verifiers. These results suggest that the diverse perspectives of multiple smaller, computationally cheaper models can collectively produce a verification signal robust enough to improve even state-of-the-art generators.

Self-Improvement. Multi-agent verification can also enable models to improve their own performance through self-verification. To demonstrate, we configure BoN-MAV to use the same base LLM for both generation and verification. That is, we sample outputs from a generator LLM (Gemini-1.5-Flash or GPT-4o-mini) and create multiple aspect verifiers derived from the same LLM. Following the verifier engineering procedure from Section 3.1, we select the best subset of self-verifiers based on validation performance. As shown in Table 2, this self-verification approach yields substantial improvements over base pass@1 accuracy across all domains except HumanEval. For instance, GPT-4o-mini shows particularly strong self-improvement on MATH (+7%) and GPQA (+8%).

4 Conclusion

We have introduced Multi-Agent Verification (MAV), a test-time compute paradigm that combines multiple verifiers to improve performance. MAV enables test-time scaling along two orthogonal dimensions: (1) the traditional dimension of increasing the number of candidate outputs sampled from a *generator* LLM, and (2) our novel test-time scaling dimension of increasing the number of *verifiers* evaluating each output. We propose Aspect Verifiers (AVs) as one possible implementation choice for the verifiers in a MAV system. AVs are off-the-shelf LLMs that require no additional training and naturally support combining verification signals from models based on different LLMs, training algorithms, architectures, data, or prompts. Thus, AVs are a convenient building block for multi-agent verification, allowing us to leverage the growing ecosystem of language models and their diverse capabilities. We introduce BoN-MAV as a simple multi-agent verification algorithm and our results indicate that increasing the number of diverse verifiers is a promising dimension for scaling test-time compute. Specifically, we demonstrate that this approach improves test-time performance across multiple domains and generator LLMs, enables weak-to-strong generalization by combining multiple weak verifiers to improve stronger generators, and facilitates self-improvement when the generator LLM is also used as the base LLM for each of the aspect verifiers. Moreover, BoN-MAV represents just one approach to multi-agent verification and we expect better-engineered verifiers and more nuanced aggregation strategies to unlock even stronger scaling patterns. We discuss the limitations of our approach and potential directions for future research in Appendix D. We hope that our work inspires future research into multi-agent verification algorithms and further exploration of scaling the number of verifiers as a powerful new dimension for test-time compute.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 18

Anurag Ajay, Seungwook Han, Yilun Du, Shuang Li, Abhi Gupta, Tommi Jaakkola, Josh Tenenbaum, Leslie Kaelbling, Akash Srivastava, and Pulkit Agrawal. Compositional foundation models for hierarchical planning. *Advances in Neural Information Processing Systems*, 36:22304–22325, 2023. 18

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016. 17

Kenneth J Arrow. *Social choice and individual values*, volume 12. Yale university press, 2012. 17

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022a. 2

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022b. 17

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024. 18

Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia. *Handbook of computational social choice*. Cambridge University Press, 2016. 17

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 1

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *arXiv preprint arXiv:2312.09390*, 2023. 3, 8, 17

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better LLM-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023. 18

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*, 2024. 18

Justin Chih-Yao Chen, Swarnadeep Saha, and Mohit Bansal. Reconcile: Round-table conference improves reasoning via consensus among diverse LLMs. *arXiv preprint arXiv:2309.13007*, 2023. 18

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 5, 19, 22

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240): 1–113, 2023. 1

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017. 2

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. 2, 3, 4, 5, 17

Roi Cohen, May Hamri, Mor Geva, and Amir Globerson. Lm vs lm: Detecting factual errors via cross examination. *arXiv preprint arXiv:2305.13281*, 2023. 18

Thomas Coste, Usman Anwar, Robert Kirk, and David Krueger. Reward model ensembles help mitigate overoptimization. *arXiv preprint arXiv:2310.02743*, 2023. 3, 17

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*, 2023. 17

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023. 4, 16, 18

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 18

Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D'Amour, DJ Dvijotham, Adam Fisch, Katherine Heller, Stephen Pfohl, Deepak Ramachandran, et al. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. *arXiv preprint arXiv:2312.09244*, 2023. 3, 17

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. *arXiv preprint arXiv:2309.17179*, 2023. 18

Peter C Fishburn. *The theory of social choice*. Princeton University Press, 2015. 17

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pp. 10835–10866. PMLR, 2023a. 3, 17

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023b. 17

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023. 17

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020. 5

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. *arXiv preprint arXiv:2103.03874*, 2021. 1, 5, 8, 15, 19, 21

David Herel and Tomas Mikolov. Thinking tokens for language modeling. *arXiv preprint arXiv:2405.08644*, 2024. 17

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. 1

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022. 17

Geoffrey Irving, Paul Christiano, and Dario Amodei. Ai safety via debate. *arXiv preprint arXiv:1805.00899*, 2018. 17, 18

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023. 18

Bowen Jiang, Yangxinyu Xie, Xiaomeng Wang, Weijie J Su, Camillo Jose Taylor, and Tanwi Mallick. Multi-modal and multi-agent systems meet rationality: A survey. In *ICML 2024 Workshop on LLMs and Cognition*, 2024. 18

Jerry S Kelly. *Social choice theory: An introduction*. Springer Science & Business Media, 2013. 17

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024. 18

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. Rewardbench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*, 2024. 6, 18

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022. 3, 5, 17

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 18

Shuang Li, Yilun Du, Joshua B Tenenbaum, Antonio Torralba, and Igor Mordatch. Composing ensembles of pre-trained models via iterative consensus. *arXiv preprint arXiv:2210.11522*, 2022a. 18

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022b. 3, 5, 17

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*, 2023. 18

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. 17

Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-reward: Bag of tricks for reward modeling in llms. *arXiv preprint arXiv:2410.18451*, 2024. 18

Jieyi Long. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023. 18

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024. 17

Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative reward models. *arXiv preprint arXiv:2410.12832*, 2024. 4, 18

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021. 2, 4, 5, 17

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. 17

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022. 2

Chau Pham, Boyi Liu, Yingxiang Yang, Zhengyu Chen, Tianyi Liu, Jianbo Yuan, Bryan A Plummer, Zhaoran Wang, and Hongxia Yang. Let models speak ciphers: Multiagent debate through embeddings. *arXiv preprint arXiv:2310.06272*, 2023. 18

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*, 2024. 18

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. 17

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025. 17

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. *arXiv preprint arXiv:2407.18219*, 2024. 17

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023. 5, 19, 23

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*, 2022. 17

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023. 17

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024. 17

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024. 3, 17

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020. 2, 3, 4, 5, 17

Vighnesh Subramaniam, Yilun Du, Joshua B Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. Multiagent finetuning: Self improvement with diverse reasoning chains. *arXiv preprint arXiv:2501.05707*, 2025. 18

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024a. 18

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024b. 18

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022. 3, 5, 17

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024a. 17

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. Rethinking the bounds of llm reasoning: Are multi-agent discussions the key? *arXiv preprint arXiv:2402.18272*, 2024b. 18

Xinyi Wang, Lucas Caccia, Oleksiy Ostapenko, Xingdi Yuan, William Yang Wang, and Alessandro Sordoni. Guiding language model reasoning with planning tokens. *arXiv preprint arXiv:2310.05707*, 2023a. 17

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. 3, 5, 17

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024c. 5

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*, 2023b. 18

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021. 4

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022. 17

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*, 36, 2024. 18

Zhenran Xu, Senbao Shi, Baotian Hu, Jindi Yu, Dongfang Li, Min Zhang, and Yuxiang Wu. Towards reasoning in large language models via multi-agent peer review collaboration. *arXiv preprint arXiv:2311.08152*, 2023. 18

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024. 18

Xiao Yu, Maximillian Chen, and Zhou Yu. Prompt-based monte-carlo tree search for goal-oriented dialogue policy planning. *arXiv preprint arXiv:2305.13660*, 2023. 18

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022. 18

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: LLM self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*, 2024a. 18

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024b. 4, 18

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging LLM-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023. 17

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023. 18

A Acknowledgments

We thank Romi Lifshitz, Andrew Li, Parand Alamdari, Claas Voelcker, Lev McKinney, Toryn Klassen, and David Glukhov for their helpful comments. We thank ArdaLabs for providing funding and compute resources for this project (https://ardalabs.ai/). The second author gratefully acknowledges funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada CIFAR AI Chairs Program. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence (https://vectorinstitute.ai/partnerships/).

B Visualization and Tables

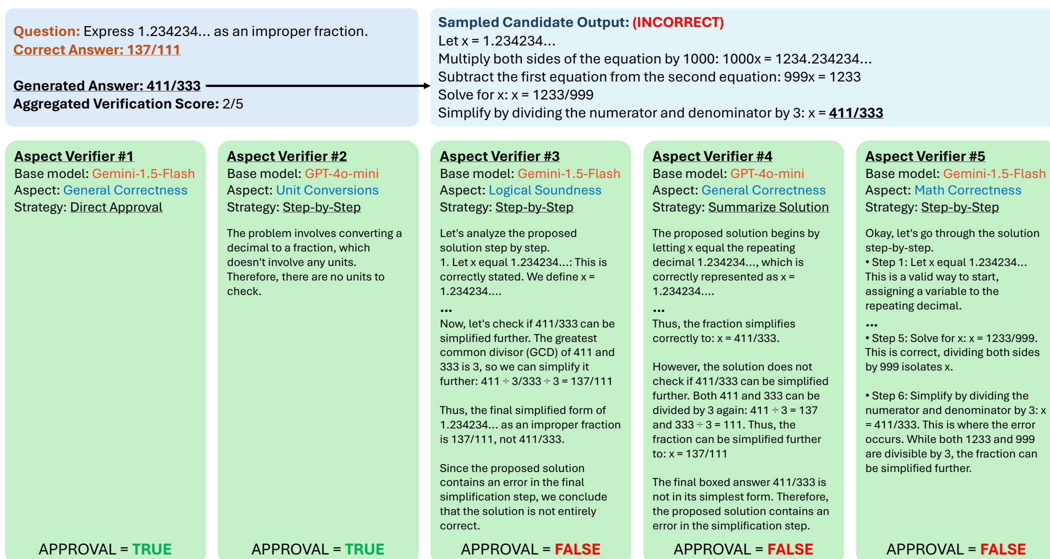

Figure 6: **Multi-agent verification for a single solution.** An illustration of how combining multiple aspect verifiers can produce a more robust verification signal. Five different aspect verifiers evaluate an incorrect MATH (Hendrycks et al., 2021) solution sampled from Gemini-1.5-Pro. The verifiers vary across three dimensions: base models (Gemini-1.5-Flash, GPT-4o-mini), aspects to verify (e.g., general correctness, mathematical correctness, unit conversions), and verification strategies (e.g., direct yes/no approval, step-by-step verification, summarization). Two verifiers miss the error: one using direct approval without step-by-step thinking, and another tasked with checking unit conversions (the problem contains no units to convert, so the verifier finds no errors and incorrectly approves the solution). The remaining three verifiers each identify the mistake through careful analysis. This demonstrates how combining diverse verification methods can produce a robust signal despite individual verifier failures, as the majority correctly identify the error.

		MATH		MMLU-Pro		GPQA (diamond)		HumanEval	
	Generator LLM	B-MAV	pass@1	B-MAV	pass@1	B-MAV	pass@1	B-MAV	pass@1
WtoS	**Gemini-1.5-Pro**	**72.7**	64.7	**72.3**	68.0	**49.0**	45.0	**88.0**	84.0
	GPT-4o	**76.3**	68.3	**75.7**	73.3	**59.0**	54.0	92.0	**94.0**
SI	**Gemini-1.5-Flash**	**59.0**	52.7	**64.0**	59.3	**43.0**	42.0	78.0	**79.0**
	GPT-4o-mini	**76.0**	69.0	**65.7**	62.3	**46.0**	38.0	**86.0**	**86.0**

Table 2: **Weak-to-strong generalization and self-improvement.** Performance (accuracy %) using the BoN-MAV algorithm (labeled as B-MAV in the table) compared to base pass@1 accuracy. For weak-to-strong generalization (top, "WtoS"), we use aspect verifiers based on weaker models (Gemini-1.5-Flash and GPT-4o-mini) to improve stronger generator LLMs. For self-improvement (bottom, "SI"), we use aspect verifiers based on the same model as the generator. BoN-MAV improves performance in nearly all cases.

	Ablation 1: Verifier-Engineering				*Ablation 2: Verifier Diversity*			
	MMLU-Pro		GPQA (diamond)		MMLU-Pro		GPQA (diamond)	
Generator LLM	Eng	All	Eng	All	Diverse	Same	Diverse	Same
Gemini-1.5-Flash	**66.7**	65.7	**42.0**	41.0	**66.7**	66.3	**42.0**	39.0
Gemini-1.5-Pro	**72.3**	70.3	**49.0**	**49.0**	**72.3**	71.0	49.0	**55.0**
GPT-4o-mini	**67.0**	65.3	**50.0**	49.0	**67.0**	64.7	**50.0**	42.0
GPT-4o	**75.7**	75.3	**59.0**	55.0	**75.7**	75.0	**59.0**	58.0

Table 3: **Ablation Studies.** *Left:* Performance comparison between using engineered domain-specific verifier subsets (Eng) versus using all verifiers without tuning (All). *Right:* Performance comparison between using diverse verifiers from $\mathcal{M}^d$ (Diverse) versus querying the single best-performing verifier multiple times (Same). All metrics are accuracy (%).

C Analysis: Understanding Multi-Agent Verification

To better understand the key design choices that impact multi-agent verification, we conduct two ablation studies on MMLU-Pro and GPQA (diamond)—the two most challenging domains in our evaluation. We investigate: **(1)** how performance depends on engineering domain-specific sets of verifiers, and **(2)** whether using diverse verifiers outperforms repeatedly querying the best verifier.

Effect of Verifier Engineering. In Section 3.1, we introduced verifier engineering as an approach for selecting a relevant subset of verifiers $\mathcal{M}^d \subseteq \mathcal{M}$ for each domain d. Here, we compare our engineered verifier subsets $\mathcal{M}^d$ against a simple baseline that uses all available aspect verifiers in $\mathcal{M}$ (see Appendix F.1 for a full list) without any domain-specific tuning. **??** (left) shows that engineering the set of verifiers is a more effective strategy. However, Table 5 in the Appendix shows that even the simple strategy of combining all verifiers in $\mathcal{M}$ remains competitive with both self-consistency and reward model verification baselines.

Effect of Verifier Diversity. Here, we investigate whether using diverse verifiers outperforms repeatedly querying a single verifier. Specifically, we compare the performance of our diverse domain-specific subsets $\mathcal{M}^d$ versus repeatedly querying the single best-performing verifier $v^* \in \mathcal{M}^d$ for domain d (where the number of queries to v^* equals $|\mathcal{M}^d|$). As shown in Table 3 (right), using diverse sets of verifiers generally outperforms querying the same verifier multiple times.

D Discussion

Multi-Agent Verification (MAV) introduces a promising dimension for scaling test-time compute: scaling the number of verifiers. In Section 3, we demonstrated that combining multiple verifiers enables more effective evaluation of candidate outputs, facilitates weak-to-strong generalization, and allows for self-improvement. However, our approach has important limitations and there are several opportunities for future work to explore.

First, our investigation is limited to a pool of 20 aspect verifiers based on just two base LLMs, and the design of our verifiers is constrained by our ability to come up with diverse verification strategies and relevant aspects. Future work could explore scaling to many more verifiers and try a more systematic exploration of the space of verifiers, potentially using LLMs themselves to generate diverse verification strategies and identify relevant aspects to verify. With better-engineered verifiers and more systematic exploration, we expect to observe stronger scaling patterns.

Second, our aggregation technique described in Section 2.2 uses a simple voting mechanism that directly sums the individual binary approvals from each verifier. This approach does not account for the confidence or relevance of each verifier, and verifiers do not observe each other's decisions or feedback. Future works could explore more sophisticated aggregation methods such as confidence-weighted voting or allowing verifiers to engage in debate (Du et al., 2023) before producing an approval. Moreover, our current approach uses a static engineered set of verifiers $\mathcal{M}^d$ for all questions in a domain d, even though it may be best to use fewer or different verifiers for specific questions. Future works could investigate dynamically selecting the best set of verifiers for particular problems or adaptively choosing additional verifiers based on the results of the first few verification queries. Additionally, the field of social choice theory (Arrow, 2012; Fishburn, 2015; Kelly, 2013; Brandt et al., 2016) is concerned with procedures for collective decision-making and might offer insights for aggregating the perspectives of diverse verifiers. Although, our setting differs in that we care more about verifier capabilities than preferences.

Next, our implementation of BoN-MAV is limited to only a single generator LLM. Thus, an interesting direction would be to explore sampling from multiple generators in addition to evaluating with multiple verifiers. Since different models may excel at solving different types of problems, this approach could make even better use of the growing ecosystem of LLMs and their diverse capabilities.

Furthermore, while our results show that BoN-MAV can improve language model performance at test-time, we did not investigate finetuning the generator LLM on the outputs selected by our verifiers. Similar to how prior works have finetuned on outputs selected through self-consistency (Huang et al., 2022) or reward models (Dong et al., 2023), training on outputs selected by MAV systems could be explored as a method to improve the generator LLM and also each of the LLM-based verifiers. Moreover, an interesting direction for future work is to directly use reinforcement learning to train both the generator and verifier models. That is, generator LLMs can be trained to maximize the scores across multiple verifiers, and the verifiers can simultaneously be trained to accurately verify individual aspects of responses.

Finally, multi-agent verification offers interesting opportunities for AI safety and oversight. The ability to combine multiple verifiers checking different aspects aligns with recent efforts towards safety checking the outputs of language models. That is, different verifiers can be engineered to check various safety and alignment properties, from basic constraints like avoiding harmful content to more nuanced properties like reasoning transparency. Our results on weak-to-strong generalization also align with recent work on scalable oversight, where weaker systems supervise stronger ones (Amodei et al., 2016; Saunders et al., 2022; Burns et al., 2023). In general, our work connects to broader ideas in AI alignment about using multiple models to improve safety (Irving et al., 2018).

An underlying thread throughout our work and discussion is the vision of a growing ecosystem of diverse language models that generate, verify, and learn from each other. Our work on multi-agent verification represents one step in this direction, and each of the future directions we have discussed offers a potential avenue for additional progress. We look forward to seeing how the research community advances these ideas.

E Related Works

Scaling Test-Time Compute. Recent work has demonstrated that increasing computational resources during inference can significantly improve LLM performance (e.g., Wei et al. 2022; Snell et al. 2024). One line of research focuses on techniques where a single *generator* LLM produces additional output tokens during inference. These include scratchpads or Chain-of-Thought prompting (Nye et al., 2021; Wei et al., 2022), self-consistency or majority voting techniques (Wang et al., 2022; Li et al., 2022b; Thoppilan et al., 2022; Lewkowycz et al., 2022), and various self-reflection methods (e.g., Shinn et al. 2024; Qu et al. 2024; Madaan et al. 2024; Saunders et al. 2022; Bai et al. 2022b). Other works have explored training LLMs to generate special tokens which enhance reasoning ability at test-time (e.g., Goyal et al. 2023; Wang et al. 2023a; Herel & Mikolov 2024) or augmenting language models with tool-use abilities (e.g., Schick et al. 2023; Gao et al. 2023b; Qin et al. 2023; Qu et al. 2025).

Another line of research focuses on using a *verifier* model to evaluate the quality or correctness of outputs sampled from generator models (Cobbe et al., 2021; Zheng et al., 2023; Snell et al., 2024). Typically, this is done through best-of-n sampling (Stiennon et al., 2020; Cobbe et al., 2021; Nakano et al., 2021), where n candidate outputs are generated and the highest-scoring output is selected based on some verifier. This verification can be performed at the outcome-level (Stiennon et al., 2020; Cobbe et al., 2021) or process-level (Lightman et al., 2023; Wang et al., 2024a). Recent works (Coste et al., 2023; Eisenstein et al., 2023) have also explored using ensembles of homogeneous reward models (identical model initializations trained on the same data but with different random seeds) to mitigate reward model overoptimization (Gao et al., 2023a). Additionally, some approaches allow reward models to produce their own Chain-of-Thought reasoning before scoring (Zhang et al., 2024b; Mahan et al., 2024). Various papers have combined language with search techniques at test-time, using verifiers to provide a heuristic signal. These verifiers may use LLMs as prompted value functions (e.g., Yu et al. 2023; Yao et al. 2024; Xie et al. 2024), incorporate real environment feedback (e.g., Zhou et al. 2023; Koh et al. 2024; Putta et al. 2024; Long 2023; Besta et al. 2024), or use trained value functions (e.g., Feng et al. 2023; Zhang et al. 2024a; Chen et al. 2024). Unlike prior works which typically rely on a single reward model verifier or homogeneous reward model ensembles trained on the same data, we propose a framework for combining multiple heterogeneous verifiers without additional training, and investigate scaling the number and type of verifiers as a novel test-time scaling dimension.

Multi-Agent Reasoning with Language Models. Recent works have investigated several approaches to multi-agent interaction for improving language model reasoning. Language model debate (e.g., Du et al. 2023; Chan et al. 2023; Pham et al. 2023; Liang et al. 2023; Subramaniam et al. 2025; Li et al. 2023; Cohen et al. 2023) and multi-agent discourse (e.g., Chen et al. 2023; Wang et al. 2023b; 2024b; Xu et al. 2023) have been studied as ways to enhance reasoning, and also as a direction for scalable oversight research (Irving et al., 2018). Prior works have also explored performing search with language models, which typically combines a generator LLM and a value model to guide exploration (see the previous paragraph). Moreover, some works have explored multi-modal reasoning through agent collaboration (e.g., Zeng et al. 2022; Li et al. 2022a; Ajay et al. 2023; Jiang et al. 2024). Unlike prior work on multi-agent reasoning which focuses on collaborative problem-solving, we introduce a framework specifically for scaling test-time verification by combining multiple verifiers without training.

F Experimental Setup

F.1 Aspect Verifier Subsets

Table 4 outlines all 20 aspect verifiers in $\mathcal{M}$ and which ones were selected for each domain-specific subset $\mathcal{M}^d$.

F.2 Generator LLMs

We evaluate eight generator LLMs (four closed-source models and four open-source models) and restrict our set of generator models to those released before September 2024. For closed-source models, we use gemini-1.5-flash-001 and gemini-1.5-pro-001 (Team et al., 2024a), as well as gpt-4o-mini-2024-07-18 and gpt-4o-2024-08-06 (Achiam et al., 2023). For open-source models, we use Mistral-7B-v0.3 (Jiang et al., 2023), Llama-3.1-8B (Dubey et al., 2024), Gemma-2-9B, and Gemma-2-27B (Team et al., 2024b).

F.3 Reward Model Baseline

Our reward model verification baseline (BoN-RM) uses `Skywork/Skywork-Reward-Llama-3.1-8B-v0.2` (Liu et al., 2024), the top scoring open-source 8B reward model on RewardBench (Lambert et al., 2024) at the time of writing. This pretrained reward model outperforms numerous larger models including 70B and 340B models, and can be run on academic-scale compute.

F.4 Prompts

For generator LLMs, we use a consistent prompt format across all models while varying the content by domain. Table 6 contains these domain-specific prompts.

For aspect verifiers, each prompt consists of two components:

1. A domain-dependent system prompt (Table 7) that establishes the verification context (e.g., mathematical problems, multiple-choice questions, or code implementations)
2. A domain-independent verification prompt (Table 8 and Table 9) that specifies the aspect to verify and verification strategy

Base Model	Aspect to Verify	Verification Strategy	MATH	MMLU-Pro	GPQA	HumanEval
GPT-4o-mini	Math Correctness	Step-by-Step		✓	✓	✓
	Logical Soundness	Step-by-Step		✓	✓	✓
	Factual Correctness	Step-by-Step				✓
	Unit Conversions	Step-by-Step	✓		✓	✓
	General Correctness	Direct Approval				✓
	General Correctness	Summarize Solution	✓			
	General Correctness	Explain Differently		✓	✓	✓
	General Correctness	Edge Cases	✓	✓		✓
	General Correctness	Common Mistakes	✓	✓		
	General Correctness	Domain Knowledge	✓	✓		✓
Gemini-1.5-Flash	Math Correctness	Step-by-Step				
	Logical Soundness	Step-by-Step				✓
	Factual Correctness	Step-by-Step				
	Unit Conversions	Step-by-Step		✓	✓	✓
	General Correctness	Direct Approval				✓
	General Correctness	Summarize Solution				✓
	General Correctness	Explain Differently			✓	✓
	General Correctness	Edge Cases	✓			
	General Correctness	Common Mistakes		✓	✓	
	General Correctness	Domain Knowledge				✓
Total Verifiers Used			**6**	**8**	**7**	**14**

Table 4: Overview of all aspect verifiers in $\mathcal{M}$. Checkmarks (✓) indicate which verifiers were selected for each domain-specific subset $\mathcal{M}^d$. The table shows all 20 combinations of base models, aspects to verify, and verification strategies that we created (10 per base model). The bottom row shows the number of verifiers $|\mathcal{M}^d|$ for each domain.

This two-part structure allows us to combine any aspect-strategy verification method with any domain while maintaining consistent evaluation criteria across base models.

G Additional Results

Table 5 compares BoN-MAV using all 20 aspect verifiers in $\mathcal{M}$ (without domain-specific engineering) against self-consistency and reward model verification. Even without engineering domain-specific subsets $\mathcal{M}^d$, combining all verifiers remains competitive with baseline methods.

H Additional Illustrations

Figure 7, Figure 8, and Figure 9 provide additional examples of how multiple aspect verifiers evaluate a single candidate output. Figure 7 demonstrates verification using multiple strategies with a single base model on MATH (Hendrycks et al., 2021). Figure 8 shows verification of a coding solution from HumanEval (Chen et al., 2021). Figure 9 illustrates verification of a correct solution from GPQA (diamond) (Rein et al., 2023), showing how different base models can assess the same aspect differently. Each figure follows the same format as Figure 6 from the main paper.

	MMLU-Pro				GPQA (diamond)			
Generator Model	MAV-All	Cons	RM	pass@1	MAV-All	Cons	RM	pass@1
Gemini-1.5-Flash	**65.7**	63.3	60.7	59.3	41.0	40.0	**46.0**	42.0
Gemini-1.5-Pro	70.3	**71.7**	69.3	68.0	**49.0**	45.0	**49.0**	45.0
GPT-4o-mini	**65.3**	63.7	62.7	62.3	**49.0**	48.0	44.0	38.0
GPT-4o	75.3	**76.3**	72.7	73.3	55.0	**59.0**	58.0	54.0

Table 5: Performance (accuracy %) of BoN-MAV with all 20 aspect verifiers (without any tuning, labeled as MAV-all in the table) compared to reward model verification (RM), self-consistency (Cons), and the base pass@1 accuracy of the generator LLM. Using all verifiers without domain-specific tuning remains competitive with reward model verification and self-consistency.

Domain	**Generator Prompt**
MATH	*You are a helpful assistant skilled in math problem-solving. Always end your solution with the final numerical answer enclosed in LaTeX \boxed{} notation. If there is no solution, reply with an empty \boxed{}. Please solve the following math problem step by step: < Question > Provide your detailed solution below:*
MMLU-Pro	*Answer the following multiple choice question. Think step by step before answering, and then output the answer in the format of "The answer is (X)" at the end, where X is the LETTER of the correct answer.* *QUESTION: < Question >* *Think step by step, then end with EXACTLY "The answer is (X)", where X is the LETTER of the correct answer. Do not include the answer text itself, only the letter.*
GPQA (diamond)	Same as MMLU-Pro.
HumanEval	*Read the following function signature and docstring, and fully implement the function described. Your response should only contain the code for this function.* *< Function Signature and Docstring >*

Table 6: **Generator Prompts.** Generator prompts by domain. Each domain uses one consistent prompt across all generator LLMs.

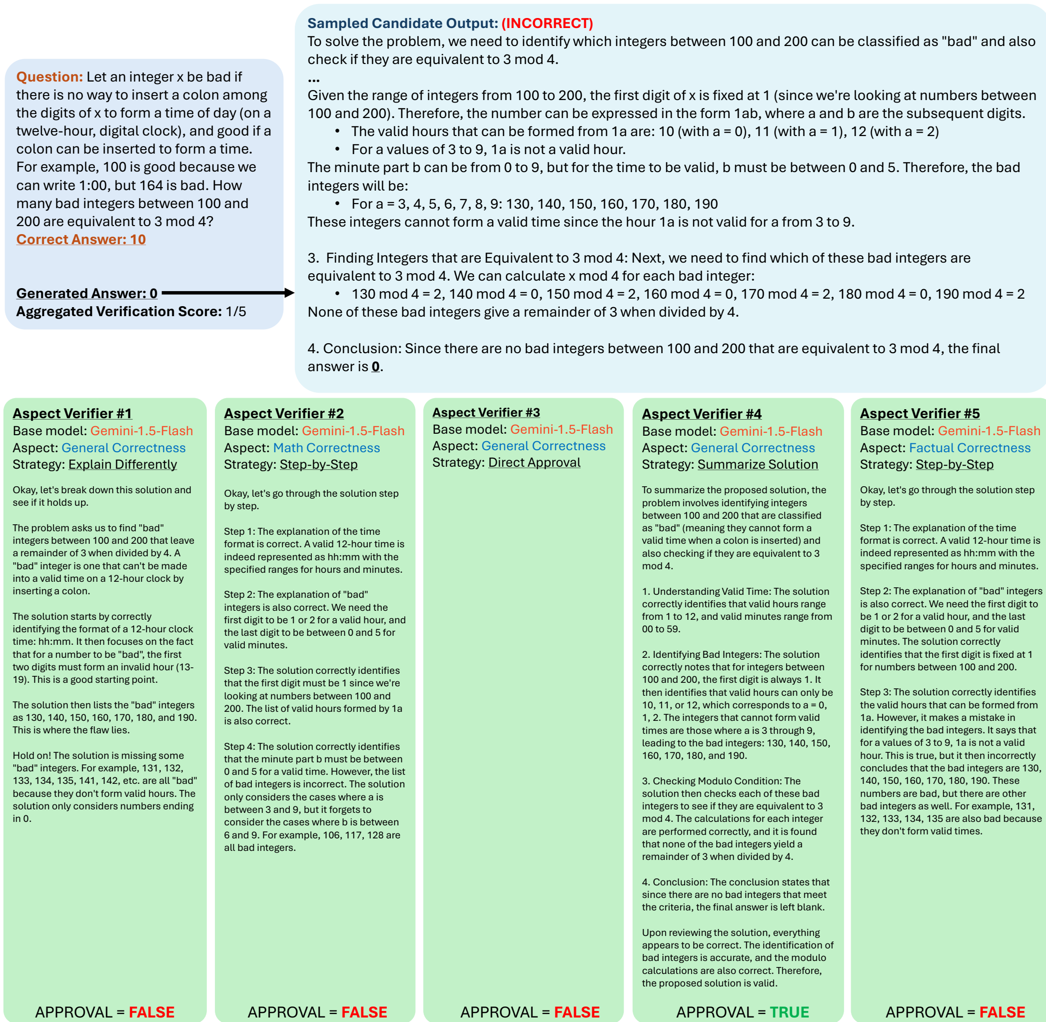

Figure 7: **Multi-agent verification for a single solution (additional example).** An illustration similar to Figure 6. Five different aspect verifiers evaluate an incorrect MATH (Hendrycks et al., 2021) solution sampled from Gemini-1.5-Pro. All verifiers use Gemini-1.5-Flash as the base model but vary in their aspects to verify (e.g., general correctness, mathematical correctness) and verification strategies (e.g., direct approval, step-by-step verification). Four verifiers correctly identify the error, while one verifier using general correctness through summarization incorrectly approves the solution. This demonstrates how diverse verification methods can produce more reliable signals even when using a single base model, as multiple verifiers can compensate when another fails.

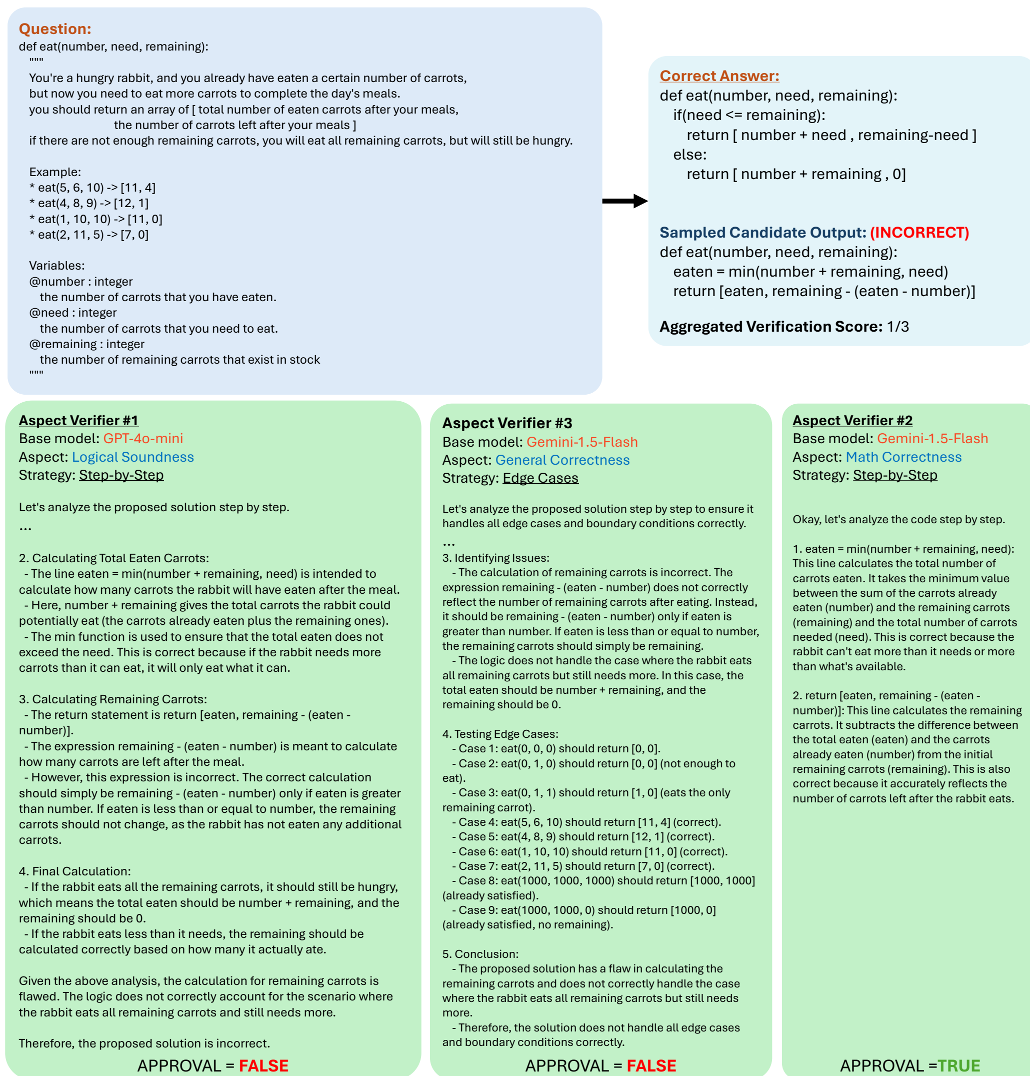

Figure 8: **Multi-agent verification for a single solution (additional example).** An illustration similar to Figure 6. Here, three different aspect verifiers evaluate an incorrect HumanEval (Chen et al., 2021) solution sampled from Gemini-1.5-Flash. Two verifiers correctly identify the error through careful analysis, while one verifier incorrectly approves the solution.

Figure 9: **Multi-agent verification for a single solution (additional example).** An illustration similar to Figure 6. Here, three different aspect verifiers evaluate a correct GPQA (diamond) (Rein et al., 2023) solution sampled from Gemma-2-27B. The verifiers vary in their base models, aspects to verify, and verification strategies. Notice that Gemini-1.5-Flash incorrectly rejects the solution when evaluating logical soundness but correctly approves it when prompted to explain the solution differently. Meanwhile, GPT-4o-mini correctly approves the solution when evaluating logical soundness. Different base models can produce different evaluations of the same aspect.

Domain	Aspect Verifier System Prompt
MATH	*You are a critical verifier tasked with evaluating mathematical problem-solving. You will be presented with a question and a proposed solution. Your job is to carefully go over and analyze the solution. Follow the instructions.*
MMLU-Pro	*You are a critical verifier tasked with evaluating multiple-choice question-answering. You will be presented with a question, the multiple-choice options, and a proposed solution. Your job is to carefully go over and analyze the solution. Follow the instructions.*
GPQA (diamond)	Same as MMLU-Pro.
HumanEval	*You are a critical verifier tasked with evaluating code implementations. You will be presented with a prompt and a code implementation. Your job is to carefully go over and analyze the code. Follow the instructions.*

Table 7: **Aspect Verifier System Prompts.** System prompts for aspect verifiers. These provide domain-specific context for the verification instructions in Table 8 and Table 9.

Aspect to Verify	Verification Strategy	Aspect Verifier Prompt
Mathematical Correctness	Step-by-Step	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Go over each step in the proposed solution and check whether it is mathematically correct. Think out load. If you reach a step that is incorrect, stop and reply 'FINAL VERIFICATION ANSWER: False'. If you get to the end of all the steps and each step was correct, reply 'FINAL VERIFICATION ANSWER: True'.*
Logical Soundness	Step-by-Step	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Go over each step in the proposed solution and check whether it is logically sound. Think out load. If you reach a step that is not logically sound, stop and reply 'FINAL VERIFICATION ANSWER: False'. If you get to the end of all the steps and each step was logically sound, reply 'FINAL VERIFICATION ANSWER: True'.*
Factual Correctness	Step-by-Step	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Go over each step in the proposed solution and check whether the facts presented are correct. Think out load. If you reach a step with incorrect facts, stop and reply 'FINAL VERIFICATION ANSWER: False'. If you get to the end of all the steps and each step had correct facts, reply 'FINAL VERIFICATION ANSWER: True'.*
Unit Conversions	Step-by-Step	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Check if the units are handled correctly in each step of the solution. Think out loud. If you find any issues with the units, stop and reply 'FINAL VERIFICATION ANSWER: False'. If all units are handled correctly, reply 'FINAL VERIFICATION ANSWER: True'.*

Table 8: **Aspect Verifier Prompts (Part 1).** Aspect verifier prompts for each aspect-strategy combination. These prompts follow the system prompts in Table 7.

Aspect to Verify	Verification Strategy	Aspect Verifier Prompt
General Correctness	Direct Approval	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Is this solution correct for the given question? Respond with ONLY 'FINAL VERIFICATION ANSWER: True' or ONLY 'FINAL VERIFICATION ANSWER: False'. Do not provide any explanation or additional text.*
General Correctness	Summarize Solution	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Summarize the solution in your own words, explore anything you think may be incorrect. Think out load. If you find something that's incorrect, stop and reply 'FINAL VERIFICATION ANSWER: False'. If you've gone over the solution and everything seems correct, reply 'FINAL VERIFICATION ANSWER: True'.*
General Correctness	Explain Differently	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Explain the solution in a different way than it was presented. Try to find any flaws in the solution. Think out load. If you find something that's incorrect, stop and reply 'FINAL VERIFICATION ANSWER: False'. If you've gone over the solution and everything seems correct, reply 'FINAL VERIFICATION ANSWER: True'.*
General Correctness	Edge Cases	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Check if the solution handles edge cases and boundary conditions, test extreme values or special cases. Think out loud. If any boundary conditions or edge cases fail, stop and reply 'FINAL VERIFICATION ANSWER: False'. If all boundary conditions and edge cases are handled correctly, reply 'FINAL VERIFICATION ANSWER: True'.*
General Correctness	Common Mistakes	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Check if the solution has any common mistakes, calculation errors, or misconceptions that typically found in this type of problem. Think out loud. If you find any common mistakes, stop and reply 'FINAL VERIFICATION ANSWER: False'. If no common mistakes are found, reply 'FINAL VERIFICATION ANSWER: True'.*
General Correctness	Domain Knowledge	*QUESTION: <Question>* *PROPOSED SOLUTION: <Solution>* *INSTRUCTIONS: Check if the solution correctly applies relevant domain-knowledge, established theories, and standard practices for this type of problem. Think out loud. If any domain knowledge is misapplied or violated, stop and reply 'FINAL VERIFICATION ANSWER: False'. If all domain-specific knowledge is correctly applied, reply 'FINAL VERIFICATION ANSWER: True'.*

Table 9: **Aspect Verifier Prompts (Part 2).** Aspect verifier prompts for each aspect-strategy combination. These prompts follow the system prompts in Table 7.