Agentic Reward Modeling: Integrating Human Preferences with Verifiable Correctness Signals for Reliable Reward Systems

Anonymous ACL submission

Abstract

Reward models (RMs) are crucial for the training and inference-time scaling up of large language models (LLMs). However, existing reward models primarily focus on human preferences, neglecting verifiable correctness signals which have shown strong potential in training LLMs. In this paper, we propose *agentic reward modeling*, a reward system that combines reward models with verifiable correctness signals from different aspects to provide reliable rewards. We empirically implement a reward agent, named REWARDAGENT, that combines human preference rewards with two verifiable signals: factuality and instruction following, to provide more reliable rewards. We conduct comprehensive experiments on existing reward model benchmarks and inference time best-of-n searches on real-world downstream tasks. REWARDAGENT significantly outperforms vanilla reward models, demonstrating its effectiveness. We further construct training preference pairs using REWARDAGENT and train an LLM with the DPO objective, achieving superior performance on various NLP benchmarks compared to conventional reward models. We will release our code and data to facilitate further research.

1 Introduction

Reward models (RMs) are designed to score the quality of responses and are typically used in the post-training of large language models (LLMs), such as RL (Ouyang et al., 2022) and DPO training (Rafailov et al., 2024), and in inference-time scaling laws (Wu et al., 2024; Snell et al., 2024), such as best-of-n search (Brown et al., 2024). Reliable RMs are key to the success of modern LLMs.

Despite the success of reward models, existing RMs primarily focus on human preferences, which may be susceptible to subjective biases (Saito et al., 2023; Singhal et al., 2023), while neglecting verifiable correctness signals like factuality (Liu et al., 2024b; Tan et al., 2024). As illustrated in Figure 1, existing RMs may prefer the response *A* due to its language style and longer length (Singhal et al., 2023), overlooking factual errors and failure to follow instructions. This could affect the reliability of reward models and further influence the reliability of the trained LLMs (Singhal et al., 2023; Chen et al., 2024c). Conversely, verifiable correctness rewards exhibit notable potential in specific scenarios (Guo et al., 2025), providing a valuable complement to conventional reward models.

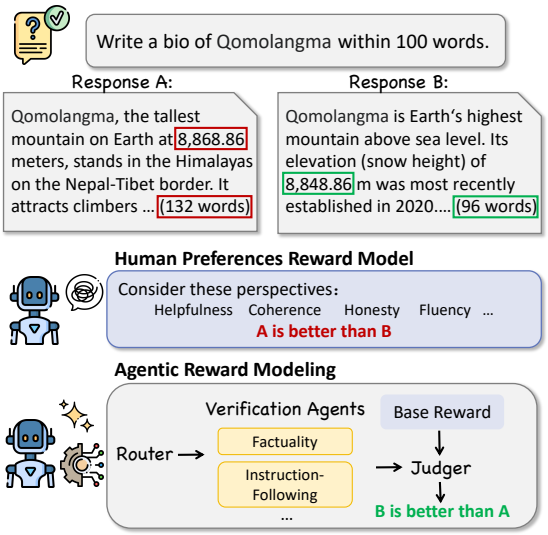


Figure 1: An illustration of *agentic reward modeling*.

Based on the above considerations, we propose *agentic reward modeling*, a reward system that combines reward models with verifiable correctness signals from different aspects to provide more reliable rewards. For example in Figure 1, a verification agent that specifically provides correctness signals, such as rule-based rewards (Mu et al., 2024), can be used to assess factual accuracy or verify adherence to instruction constraints. By integrating verifiable correctness rewards with human preferences, the reward system selects the superior response *B*. Agentic reward modeling enhances reliability through multi-dimensional correctness signals, enables flex-

ible integration of diverse verification agents, and improves the interpretability of the final reward.

In this paper, we empirically implement a reward agent, named REWARDAGENT, which integrates the conventional human preference-based reward models with correctness signals from two key aspects: (1) factuality, which assesses the factual correctness of the claimed facts in the response, and (2) instruction-following, which evaluates whether the response adheres to the hard constraints in the instruction (Zhou et al., 2023), such as length constraints, which significantly impact user experience in real-world applications (Sun et al., 2024b; Qi et al., 2024). The architecture of REWARDAGENT is shown in Figure 1, consisting of three main modules: (1) *Router*, which analyzes the instruction to determine the appropriate verification agents to invoke. (2) *Verification agents*, which evaluate the correctness of response in different aspects, including factuality and instruction-following. Specifically, for factuality, we design a verification agent that efficiently evaluates factual correctness compared to the previous factuality evaluation framework (Min et al., 2023) through a process including pairwise comparison, query generation, evidence generation, and verification, where evidence generation can utilize either a search engine or the model's parametric knowledge. For instruction-following, we design a verification agent that extracts hard constraints, generates constraint checker code, and executes the code for verification, where the constraint checker is the Python code script to verify whether a given response satisfies a specific hard constraint. (3) *Judger*, which integrates the correctness signals from the verification agents and human preference scores from the reward models to provide an overall reward score. We adopt ArmoRM (Wang et al., 2024a) as the reward model for computing human preference scores in REWARDAGENT. We use GPT-4o mini (OpenAI, 2024a) and Llama3-8B Instruct (Dubey et al., 2024) as the backbone LLMs for all the modules and implement $\text{REWARDAGENT}_{\text{MINI}}$ and $\text{REWARDAGENT}_{\text{LLAMA}}$, respectively, except that in $\text{REWARDAGENT}_{\text{LLAMA}}$, the LLM backbone of the instruction-following agent is Qwen2.5-Coder 7B (Hui et al., 2024).

We conduct extensive experiments to validate the effectiveness of REWARDAGENT. First, we conduct an evaluation on several reward model benchmarks, including RM-Bench (Liu et al., 2024b) and JudgeBench (Tan et al., 2024), as they contain response pairs that involve factual correctness, and IF-Bench, which is newly constructed for instruction-following and contains 444 instances, each of which includes an instruction with several hard constraints, a chosen response that satisfies all constraints, and a rejected response that violates some constraints. REWARDAGENT significantly outperforms other advanced reward models on these benchmarks. We further apply reward models in real-world downstream tasks, including inference-time best-of-n search and constructing training preference pairs. We evaluate best-of-n search on factuality question answering dataset TriviaQA (Joshi et al., 2017) and instruction-following datasets, IFEval (Zhou et al., 2023) and CELLO (He et al., 2024). We adopt Llama3-8B Instruct and GPT-4o (OpenAI, 2024b) as policy models to generate 32 responses for each instruction with 1.0 sampling temperature. REWARDAGENT significantly outperforms the base reward model AromRM in the best-of-n search, suggesting its ability to select superior responses and unleash inference scaling laws. Finally, we apply REWARDAGENT to construct training preference pairs and train an LLM using DPO (Rafailov et al., 2024). Specifically, we construct training data from two sources: UltraFeedback (Cui et al., 2024) and on-policy data. We adopt Zephyr-7B (Tunstall et al., 2023) as the policy model and train it using DPO. The LLM trained on REWARDAGENT-constructed data consistently outperforms those trained on AromRM annotations on several NLP benchmarks, which further demonstrates the effectiveness of REWARDAGENT. We encourage the community to explore more verifiable correctness signals to develop reliable reward systems for LLM development and alignment.

2 Preliminaries

In the LLM domain, a reward model is typically a regression model that takes an instruction and a response as input and outputs a reward score (Ouyang et al., 2022), which can be formulated as $r_{\text{RM}}(x, y)$, where x denotes an instruction and y represents a response. Reward models are typically trained on a large set of preference pairs based on the Bradley-Terry (BT) model (Bradley and Terry, 1952).

However, due to the subjectivity and complexity of human preferences and the capacity limitations of the BT model (Munos et al., 2023; Swamy et al., 2024; Sun et al., 2024a), reward models often exhibit subjective bias, such as favoring longer and detailed outputs (Saito et al., 2023), while ne-

glecting verifiable correctness signals like factuality (Liu et al., 2024b; Tan et al., 2024). On the other hand, training LLMs with verifiable correctness signals has shown strong potential (Lambert et al., 2024a; Guo et al., 2025). Based on these considerations, we propose *agentic reward modeling*, a reward system that integrates reward models with verifiable correctness signals from different aspects to provide more reliable rewards. Agentic reward modeling can be formulated as follows:

$$r(x, y) = \underbrace{\lambda \cdot r_{\text{RM}}(x, y)}_{\text{base reward}} + \sum_{i \in A_x} \underbrace{w_i \cdot a_i(x, y)}_{\text{correctness signals}} \quad (1)$$

λ denotes the weight of the base reward model. a_i denotes a specific verification agent that provides verifiable correctness signals, such as rule-based rewards (Mu et al., 2024). w_i denotes the corresponding weight for each verification agent, which can be set as a hyper-parameter or adaptive to the instruction. A_x is an index subset of the complete set of verification agents A and is determined based on the instruction x. Equation 1 provides the fundamental concept of agentic reward modeling, which can be implemented in various ways to construct a reward agent and our implementation is in § 3.

3 REWARDAGENT

In this work, we empirically implement a reward agent, named REWARDAGENT, which integrates the base human preference reward model with verifiable correctness signals from two key aspects: factuality, which assesses the correctness of claimed facts, and instruction-following, which evaluates whether the response satisfies the hard constraints specified in the instruction (Zhou et al., 2023). Both aspects significantly impact reliability and user experience in real-world applications and are challenging to evaluate effectively with existing reward models (Liu et al., 2024b). This section introduces the overall model architecture (§ 3.1) and the specific modules (§§ 3.2 to 3.4) of REWARDAGENT.

3.1 Model Architecture

Following the concept in Euqation 1, the overall architecture of REWARDAGENT is illustrated in Figure 2, which consists of three main modules: (1) *Router*, which analyzes the instruction and determines which agents to invoke, corresponding to A_x in Equation 1. As different instructions may require evaluations of different aspects of responses, dynamically selecting verification agents helps reduce inference costs and mitigate potential cumulative errors. (2) *Verification agents*, which evaluate different aspects of response correctness. In our implementation, we design two agents for assessing factuality and instruction-following, both based on LLMs augmented with additional tools. (3) *Judger*, which integrates the scores from the verification agents and human preferences from the base reward model to produce a final reward, corresponding to determining λ and w_i in Equation 1. We will provide detailed descriptions in the following sections.

3.2 Router

Given an instruction, the router analyzes its requirements to the response to select the appropriate verification agents. The router is powered by an existing LLM backbone. Specifically, we first manually provide a concise description for each verification agent, explaining its functionality and specifying the conditions for its usage. Then, we input the instruction with all agent descriptions into the LLM, prompting it to select appropriate verification agents for correctness assessment. More implementation details are placed in appendix A.

3.3 Verification Agents

Factuality Previous studies have proposed various methods to evaluate the factuality of responses, such as FactScore (Min et al., 2023), which can be directly used as a verification agent. However, these methods typically require extensive search engine queries to verify the correctness of each atomic fact, which is costly and inefficient for reward scoring. Intuitively, pairwise scoring based on only the differences between two responses can effectively reduce search engine queries and time costs. Therefore, we propose a pairwise factuality verification agent for efficiently evaluating the factual correctness of response pairs. The agent is illustrated in Figure 2, which consists of four main components: (1) Difference proposal, which identifies key differences in claimed facts between two given responses. (2) Query generation, which constructs queries based on the identified differences to retrieve evidence for distinguishing these differences. (3) Evidence generation, which uses the generated queries to retrieve supporting evidence using either external search engines or parametric knowledge in LLMs. (4) Verification, which assigns an integer score from 0 to 1 to each response, using the collected evidence and original responses

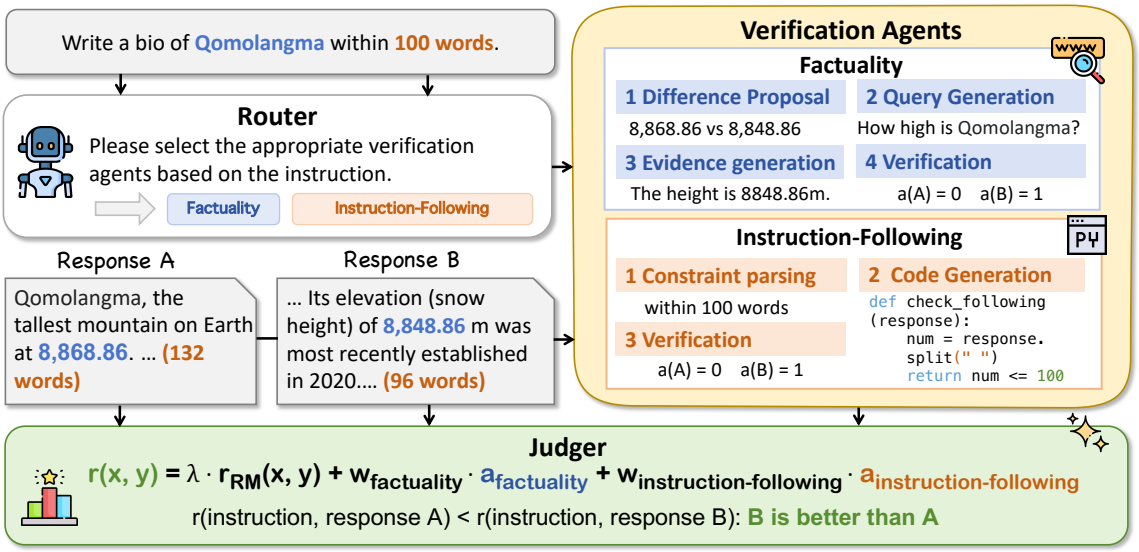


Figure 2: The framework of REWARDAGENT, including three modules: Router, Verification Agents, and Judger.

as inputs. The verification agent can effectively capture subtle factuality differences (Jiang et al., 2023) between responses while significantly reducing inference-time costs by verifying only their differences rather than all claimed facts. All modules are implemented using an LLM backbone. The implementation details are placed in appendix A.

Instruction-Following The evaluation of the instruction following primarily assesses the adherence to hard constraints (Zhou et al., 2023) specified in the instruction, such as length constraints. Typically, instruction-following constraints can be categorized into soft and hard constraints, where the former focuses on semantic aspects, such as language style, while the latter focuses on surface-form constraints, such as format, which can be objectively evaluated. For instruction-following, our verification agent focuses on hard constraints, which are difficult to evaluate with existing reward models but can be efficiently verified using external tools, such as Python code scripts. The agent is shown in Figure 2, including three components: (1) Constraint parsing, which extracts hard constraints from the instruction. (2) Code generation and refinement, which generates Python scripts used to check the adherence to the extracted constraints. The generated code takes the response as input and returns either 0 or 1, where 1 indicates that the constraint is satisfied, and 0 otherwise. We also incorporate a refinement step like Madaan et al. (2024) to correct invalid or syntactically incorrect code. Specifically, we execute the generated Python code using a Python interpreter, and if an error occurs, the error information and original code are fed back into the model to generate a refined code script. (3) Verification, which executes the generated code in the Python interpreter to obtain a binary score (0 or 1). The final score is the average of all hard constraint scores. All the modules are implemented using LLMs. The specific prompts and implementation details are provided in appendix A.

3.4 Judger

The judger integrates reward scores from verification agents and human preferences from base reward models. In our implementation, we use a weighted sum as the judger, where λ and w_i are all set to 1.0, to compute the final reward score in Equation 1. One can also adopt different λ and w_i for better applicability in different scenarios. Additionally, the judger can dynamically adjust λ and w_i based on the instruction like gating network (Wang et al., 2024a), we leave it as future work.

4 Experiments

This section presents experiments on several reward model benchmarks, including experimental setup (§ 4.1), results (§ 4.2), and analyses (§ 4.3).

4.1 Experimental Setup

REWARDAGENT Implementation We adopt the advanced and lightweight ArmoRM (Wang et al., 2024a) as the base reward model to compute human preference scores. As REWARDAGENT is agnostic to reward models, one can also adopt other advanced reward models. We use GPT-4o mini (Ope-

Model	RM-Bench		JudgeBench	IFBench			Overall
	Normal	Hard		Simple	Normal	Hard	
ArmoRM-Llama3-8B-v0.1	76.7	34.6	51.9	72.3	66.2	59.5	56.5
INF-ORM-Llama3.1-70B	77.5	25.1	59.1	78.7	69.2	53.8	55.7
Skywork-Reward-Llama-3.1-8B-v0.2	78.0	31.8	57.8	78.7	69.2	59.8	58.1
Skywork-Reward-Gemma-2-27B	82.7	35.1	55.8	**87.2**	68.4	56.1	59.2
internlm2-7b-reward	72.6	19.9	56.2	74.5	61.7	55.7	52.0
internlm2-20b-reward	74.4	26.1	61.7	74.5	68.4	58.7	56.4
GPT-4o	71.4	27.9	64.6	85.1	66.2	54.4	56.3
GPT-4o mini	60.5	15.0	51.9	70.2	59.4	51.9	45.9
o3-mini	76.0	38.6	66.6	81.9	76.3	64.6	62.8
Llama3-8B Instruct	9.3	20.2	2.6	12.8	12.8	13.6	11.3
DeepSeek-R1	83.7	50.1	**74.4**	72.3	74.4	64.0	69.1
DeepSeek-R1-Distill-Llama-8B	42.1	56.8	47.7	53.2	55.6	54.2	50.3
REWARDAGENT$_{\text{LLAMA}}$	79.3	53.5	52.9	70.2	63.9	67.8	63.2
w/ search engine	76.0	49.9	55.2	74.5	69.2	67.8	62.5
REWARDAGENT$_{\text{MINI}}$	**86.0**	**60.2**	68.2	78.7	69.2	**78.0**	**72.5**
w/ search engine	84.2	59.7	60.7	68.1	**80.5**	76.1	70.3

Table 1: Experimental results (%) of all investigated baselines and REWARDAGENT. The overall score is the average of RM-Bench, JudgeBench, and the micro-averaged score of three subsets of IFBench. By default, REWARDAGENT relies on its parametric knowledge, and "w/ search engine" denotes using Google API as an external source.

nAI, 2024a) as the LLM backbone for implementing all modules and developing REWARDAGENT$_{\text{MINI}}$. We also employ the open-source LLM Llama3-8B Instruct (Dubey et al., 2024) as the backbone and develop REWARDAGENT$_{\text{LLAMA}}$, except for the instruction-following verification agent, which requires strong coding capabilities and is instead powered by Qwen2.5-Coder 7B (Hui et al., 2024). We adopt two knowledge sources for the factuality verification agent: an external search engine using Google API and the LLM's parametric parameters. More details are placed in appendix A.

Evaluation Benchmarks Reward model benchmarks typically involve an instruction and a response pair and require selecting the better response as the chosen one. We use RM-Bench (Liu et al., 2024b), JudgeBench (Tan et al., 2024), and a new benchmark IFBENCH as evaluation benchmarks, as both RM-Bench and JudgeBench include response pairs involving factual correctness. We select the chat subset of RM-Bench as the evaluation set, using both the normal and hard settings. For JudgeBench, we use the knowledge subset as the evaluation set. We further construct a new benchmark IFBENCH to evaluate reward models on selecting responses that better follow constraints in instructions as there is no existing relevant benchmark. Specifically, we first construct instructions with several implicit constraints, integrating the constraint information with the primary task objective through paraphrasing. The constraints include both hard constraints, such as length, format, and keywords, as well as soft constraints, such as content and style. We then use GPT-4o to generate 8 responses for each instruction with a sampling temperature of 1.0. For each instruction, we create a response pair, selecting the one that satisfies all constraints as the chosen response and otherwise rejected. Based on the number of unsatisfied constraints (UC) in the rejected response, we split IFBENCH instances into three subsets: simple (#UC$\geq$3), normal (#UC=2), and hard (#UC=1), containing 47, 133, and 264 instances respectively. We report the micro-averaged accuracy across the three subsets as the final metric for IFBENCH. More evaluation details on these benchmarks are provided in appendix B.

Baselines We mainly investigate two categories of baselines: (1) typical reward models, which are specifically trained for reward modeling and typically implemented as regression models to score each response and select the one with the highest reward score as the chosen response. We investigate several advanced and representative reward models, including ArmoRM (Wang et al., 2024a), INF-ORM-Llama3.1-70B (Infly, 2024), Skywork-Reward (Liu et al., 2024a), internlm2 reward (Cai et al., 2024). (2) LLMs as generative reward models, where large language models serve as generative reward models to score responses or perform pairwise comparisons to select the best response (Lambert et al., 2024b). We evaluate propri-

etary models, including GPT-4o (OpenAI, 2024b), GPT-4o mini (OpenAI, 2024a), o3-mini (OpenAI, 2025), and open-source LLMs, including Llama3-8B Instruct (Dubey et al., 2024), DeepSeek-R1, and R1 distilled Llama3-8B model (Guo et al., 2025). We evaluate all the baselines using the code repository provided by Lambert et al. (2024b).

4.2 Experimental Results

Table 1 presents the experimental results, and we can observe that: (1) Existing reward models fall short in selecting more factual responses or better adhering to hard constraints in instructions, which may limit their reliability in real-world applications. (2) REWARDAGENT significantly outperforms the base reward model AromRM and the corresponding LLM backbone GPT-4o mini and Llama3-8B Instruct. It demonstrates that designing an appropriate reward agentic workflow can effectively enhance reward model performance. (3) Even when using Llama3-8B Instruct as the LLM backbone, REWARDAGENT$_{\text{LLAMA}}$ outperforms reward models with much more parameters and more advanced proprietary LLMs such as GPT-4o, which suggests that REWARDAGENT is more cost-efficient without requiring additional reward modeling training data or more parameters to achieve advanced performance. (4) Using a search engine as an external knowledge source for factuality slightly reduces performance in RM-Bench and JudgeBench. One possible reason is that the retrieved information may contain noise or irrelevant information (Chen et al., 2024a). We leave the detailed analysis and design of retrieval-augmented agents for future work. (5) REWARDAGENT achieves significant improvements on IFBench, particularly in the hard subset. It suggests that while not perfectly solved, existing LLMs can effectively analyze hard constraints and generate verification code, which can help the training of advanced LLMs (Lambert et al., 2024a).

In conclusion, incorporating additional verification agents for specific scenarios (Mu et al., 2024; Lambert et al., 2024a), particularly those with verifiable correctness, can develop more reliable and advanced reward systems, presenting a promising direction for future reward model development.

4.3 Analysis

We first conduct an ablation study on the verification agents in REWARDAGENT. Specifically, we investigate three settings: *– factuality verifier*, *– if verifier*, and *– both*, where the corresponding verification agents are reduced to **a single step**: using an additional LLM backbone to directly score the response, which is equivalent to the simple ensemble of the reward model ArmoRM with the corresponding LLM as a generative reward model (Coste et al., 2024). The ablation results are shown in Table 2. We can observe that removing the well-designed verification agent leads to a significant performance decrease. It demonstrates the importance of well-designed verification agents, and we encourage the community to develop more advanced verification agents for a more reliable REWARDAGENT.

Model	RM-Bench	JudgeBench	IFBench
REWARDAGENT$_{\text{MINI}}$	73.1	68.2	75.5
– factuality verifier	54.0	52.9	73.6
– if verifier	74.7	66.2	60.4
– both	55.4	58.8	58.8
Oracle setting	76.7	70.1	77.5
REWARDAGENT$_{\text{LLAMA}}$	66.4	52.9	66.9
– factuality verifier	51.9	51.6	65.8
– if verifier	58.0	57.5	57.2
– both	44.8	55.5	57.2
Oracle setting	79.5	73.1	68.5

Table 2: Experimental results (%) of ablation study and the oracle setting. *– factuality verifier* and *– if verifier* refer to the reduction of the corresponding verification agent into a single LLM scorer. The results are the micro-averaged scores of all the corresponding subsets.

We also observe the oracle setting of REWARDAGENT that invokes the most appropriate verification agents, that is, invoking the factuality agent on RM-Bench and JudgeBench, and the instruction-following verification agent on IFBench. The experimental results are shown in Table 2, and we observe that both REWARDAGENT$_{\text{MINI}}$ and REWARDAGENT$_{\text{LLAMA}}$ perform significantly better in the oracle setting. This further demonstrates the effectiveness of the verification agents and suggests that the planner in REWARDAGENT still has a large room for improvement and we leave developing a more advanced planner for future work.

5 Applications

This section explores applying REWARDAGENT to inference-time search (§ 5.1) and the training of LLMs (§ 5.2) to further validate its effectiveness.

5.1 Best-of-N Search

One important application of reward models is to conduct the inference-time search to find a better response (Brown et al., 2024; Zhang et al., 2024a),

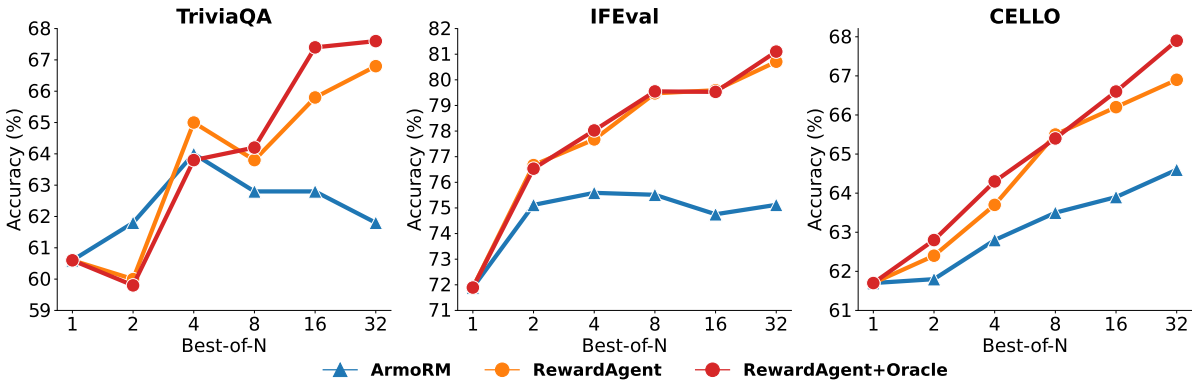


Figure 3: Best-of-n results (%) on TriviaQA, IFEval, and CELLO using the base reward model ArmoRM and REWARDAGENT to search. "+Oracle" denotes using the oracle setting of REWARDAGENT as mentioned in § 4.3.

DPO Training Data	MMLU	MMLU-Pro	TriviaQA	TruthfulQA	IFEval	CELLO	MT-Bench
–	58.9	28.8	54.8	39.5	43.3	51.5	5.2
Original UF	58.7	29.3	54.0	42.0	56.8	62.0	6.0
ArmoRM-UF	58.1	29.9	52.5	45.0	58.6	60.8	6.0
REWARDAGENT$_{\text{LLAMA}}$-UF	59.1	30.5	55.1	44.1	**59.4**	60.1	5.8
ArmoRM-OP	58.4	30.4	51.6	44.4	52.7	58.1	6.0
REWARDAGENT$_{\text{LLAMA}}$-OP	**59.5**	**31.3**	**55.3**	**48.5**	58.2	**65.7**	**6.1**

Table 3: Experimental results (%) of LLMs trained with DPO on different training data. "ArmoRM-UF" denotes using ArmoRM to construct preference pairs from UltraFeedback. "UF" and "OP" are short for UltraFeedback and on-policy data, respectively. "Original UF" refers to using the original GPT-4 annotated preference pairs from UltraFeedback to train the LLM. "–" denotes the original LLM zephyr-7b-sft-full without further DPO training.

which unleashes the inference-time scaling laws of LLMs (Snell et al., 2024; Wu et al., 2024). Therefore, we explore applying REWARDAGENT to the best-of-n search on downstream tasks. Specifically, we evaluate the best-of-n performance searched by REWARDAGENT on factuality question answering and constrained instruction following tasks.

Experimental Setup We conduct the best-of-n experiments on the factuality question answering dataset TriviaQA (Joshi et al., 2017), and the instruction-following datasets IFEval (Zhou et al., 2023) and CELLO (He et al., 2024). We use Llama3-8B Instruct and GPT-4o as the policy models to generate 32 responses for each instruction with 1.0 sampling temperature. We perform best-of-n search using the base reward model ArmoRM (Wang et al., 2024a), REWARDAGENT$_{\text{MINI}}$, and the oracle setting of REWARDAGENT$_{\text{MINI}}$. The oracle setting refers to invoking the factuality verification agent on TriviaQA, and the instruction-following verification agent on IFEval and CELLO.

Experimental Results The results of the best-of-n experiments using Llama3-8B Instruct as the policy model are shown in Figure 3. We can observe that REWARDAGENT significantly improves the best-of-n performance compared to using the base reward model ArmoRM, and the oracle setting further improves the results. It further validates the effectiveness of REWARDAGENT. The results using GPT-4o as the policy model are provided in appendix B, demonstrating the same trends and conclusions. We encourage the community to design more verification agents to unleash the inference scaling laws of LLMs across different scenarios.

5.2 DPO Training

Reward models are primarily used to train LLMs using RL (Ouyang et al., 2022) or DPO (Rafailov et al., 2024). Considering RL training is resource-intensive, we explore employing REWARDAGENT to construct preference pairs for DPO training to validate its effectiveness in real-world applications.

Experimental Setup We construct two training datasets based on: (1) UltraFeedback (Cui et al., 2024), where each instruction contains 4 responses sampled from various LLMs. (2) on-policy, which contains 20,000 instructions sampled from UltraFeedback and each instruction contains 8 responses sampled from the policy model itself with 1.0 sam-

pling temperature. We use reward models to score each response, taking the highest-scored response as the chosen one and the lowest as the rejected one to construct training pairs. We adopt the zephyr-7b-sft-full (Tunstall et al., 2023) model as the policy model to conduct DPO training because it is trained only using SFT (Ouyang et al., 2022). We evaluate the DPO-trained LLMs on various NLP benchmarks, including MMLU (Hendrycks et al., 2020), MMLU-Pro (Wang et al., 2024b), TriviaQA (Joshi et al., 2017), TruthfulQA (Lin et al., 2022), IFEval (Zhou et al., 2023), CELLO (He et al., 2024), and MT-Bench (Zheng et al., 2023). More experimental details are provided in appendix B.

Experimental Results The experimental results are shown in Table 3. We can observe that LLMs trained with data constructed by REWARDAGENT generally outperform those trained with ArmoRM, especially on the factuality question answering and instruction-following datasets. The improvement is more significant in on-policy data. Furthermore, models trained with REWARDAGENT-annotated data consistently outperform those trained on original UltraFeedback that is constructed with GPT-4. Notably, $\text{REWARDAGENT}_{\text{LLAMA}}$ uses open-source Llama3-8B Instruct and Qwen2.5-Coder 7B as the LLM backbones, at a much lower cost than GPT-4. The results further validate the effectiveness and applicability of REWARDAGENT. We believe using a more powerful LLM backbone in REWARDAGENT can achieve more advanced results and encourage the community to explore more advanced reward agents for better performance and reliability.

6 Related Work

Reward models are typically employed to score responses and are crucial to the success of modern LLMs. Since the emergence of RLHF (Ouyang et al., 2022), numerous studies have focused on developing more advanced reward models to help train LLMs. The approaches mainly include designing model architectures (Wang et al., 2024a; Dorka, 2024; Chen, 2025) and utilizing more high-quality data or new training objectives (Infly, 2024; Yuan et al., 2024; Park et al., 2024; Liu et al., 2024a; Cai et al., 2024; Cao et al., 2024; Lou et al., 2024; Li et al., 2024; Wang et al., 2024c). There are also various studies exploring using LLMs as generative reward models (Zheng et al., 2023; Mahan et al., 2024; Shiwen et al., 2024; Cao et al., 2024; Tan et al., 2024; Yu et al., 2024; Alexandru et al., 2025). Reward models are typically used for inference-time scaling laws (Irvine et al., 2023; Wu et al., 2024; Snell et al., 2024; Brown et al., 2024; Xin et al., 2024) or for training, such as RL(Ouyang et al., 2022) or DPO (Rafailov et al., 2024).

Despite the success of reward models, they primarily focus on human preferences, which may be susceptible to subjective biases or reward hacking (Saito et al., 2023; Singhal et al., 2023; Gao et al., 2023; Zhang et al., 2024b; Chen et al., 2024c). A notable limitation is *verbosity bias* (Saito et al., 2023), where reward models tend to favor longer responses (Singhal et al., 2023; Liu et al., 2024b). Additionally, some studies have shown that reward models may overlook correctness signals, such as factuality (Lin et al., 2024; Liu et al., 2024b; Tan et al., 2024). These limitations affect the reliability of reward models, thereby impacting the performance of the trained LLMs (Singhal et al., 2023).

Recently, several studies have shown that rule-based reward models or verifiable reward signals achieve impressive results in specific domains such as math (Guo et al., 2025), safety (Mu et al., 2024), instruction-following (Lambert et al., 2024a), medical (Chen et al., 2024b), and finance (Qian et al., 2025). The simplicity and advanced performance of rule-based reward models demonstrate significant potential for training LLMs, but it is still non-trivial to generalize to general domains. In this paper, we explore combining human preferences from reward models with verifiable correctness signals to develop more reliable reward systems. We believe that combining human preferences with verifiable correctness signals is a promising direction and encourage further research efforts in this area.

7 Conclusion

In this paper, we propose *agentic reward modeling*, a reward system that integrates the human preferences from conventional reward models with verifiable correctness signals to provide more reliable rewards. We empirically implement a reward agent, named REWARDAGENT, which consists of a router, well-designed verification agents for factuality and instruction-following, and a judger. We conduct extensive experiments on reward modeling benchmarks, best-of-n search, and DPO training. REWARDAGENT significantly outperforms other reward models and LLMs as generative reward models. We encourage more research efforts to develop more advanced and reliable reward systems.

Limitations

The main limitations of this work lie in the implementation of REWARDAGENT: (1) The verification agents are far from providing perfect rewards, as the average score on reward modeling benchmarks only reaches 72.5%. This suggests that achieving perfect rewards is challenging and requires further research efforts. (2) We only implement verification agents for factuality and instruction-following, which we believe are current weaknesses in reward models (Liu et al., 2024b) and important factors affecting LLM applications and user experiences. We encourage the community to explore more verifiable correctness signals. In conclusion, we believe the contribution of *agentic reward modeling* concept is substantial, and we look forward to developing more advanced reward systems in the future.

Ethical Considerations

We discuss the ethical considerations here: (1) Intellectual property. We have strictly adhered to the licenses of all utilized artifacts, including datasets, models, and code repositories. We will open-source REWARDAGENT, code, and IFBench under the MIT license[1]. (2) Intended use and potential risk control. We propose *agentic reward modeling*, a reward system that integrates human preferences with correctness signals. We implement a reward agent named REWARDAGENT to provide more reliable rewards. We believe that all data used is well anonymized. Our model does not introduce additional ethical concerns but may provide incorrect rewards due to performance limitations. Users should not conduct reward hacking (Skalse et al., 2022) and should carefully check important information. (3) AI assistance. We have used ChatGPT to refine some sentences.

References

Andrei Alexandru, Antonia Calvi, Henry Broomfield, Jackson Golden, Kyle Dai, Mathias Leys, Maurice Burger, Max Bartolo, Roman Engeler, Sashank Pisupati, Toby Drane, and Young Sun Park. 2025. Atla selene mini: A general purpose evaluation model. *Preprint*, arXiv:2501.17195.

Ralph Allan Bradley and Milton E Terry. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. 2024. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*.

Maosong Cao, Alexander Lam, Haodong Duan, Hongwei Liu, Songyang Zhang, and Kai Chen. 2024. Compassjudger-1: All-in-one judge model helps model evaluation and evolution. *arXiv preprint arXiv:2410.16256*.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024a. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17754–17762.

Junying Chen, Zhenyang Cai, Ke Ji, Xidong Wang, Wanlong Liu, Rongsheng Wang, Jianye Hou, and Benyou Wang. 2024b. Huatuogpt-o1, towards medical complex reasoning with llms. *arXiv preprint arXiv:2412.18925*.

Lichang Chen, Chen Zhu, Davit Soselia, Jiuhai Chen, Tianyi Zhou, Tom Goldstein, Heng Huang, Mohammad Shoeybi, and Bryan Catanzaro. 2024c. Odin: Disentangled reward mitigates hacking in rlhf. *arXiv preprint arXiv:2402.07319*.

Shikai Chen. 2025. Ldl-reward-gemma-2-27b-v0.1. Accessed: 2025-02-15.

Thomas Coste, Usman Anwar, Robert Kirk, and David Krueger. 2024. Reward model ensembles help mitigate overoptimization. In *Proceedings of ICLE*.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, et al. 2024. Ultrafeedback: Boosting language models with scaled ai feedback. In *Forty-first International Conference on Machine Learning*.

Nicolai Dorka. 2024. Quantile regression for distributional reward models in rlhf. *arXiv preprint arXiv:2409.10164*.

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

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

[1] https://opensource.org/license/mit

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Qianyu He, Jie Zeng, Wenhao Huang, Lina Chen, Jin Xiao, Qianxi He, Xunzhe Zhou, Jiaqing Liang, and Yanghua Xiao. 2024. Can large language models understand real-world complex instructions? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18188–18196.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. In *Proceedings of ICLR*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Infly. 2024. Inf-orm-llama3.1-70b. https://huggingface.co/infly/INF-ORM-Llama3.1-70B. Accessed: 2025-02-04.

Robert Irvine, Douglas Boubert, Vyas Raina, Adian Liusie, Ziyi Zhu, Vineet Mudupalli, Aliaksei Korshuk, Zongyi Liu, Fritz Cremer, Valentin Assassi, et al. 2023. Rewarding chatbots for real-world engagement with millions of users. *arXiv preprint arXiv:2303.06135*.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of ACL*, pages 14165–14178.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

Andreas Köpf, Yannic Kilcher, Dimitri Von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. 2023. Openassistant conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems*, 36:47669–47681.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. 2024a. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

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

Lei Li, Yekun Chai, Shuohuan Wang, Yu Sun, Hao Tian, Ningyu Zhang, and Hua Wu. 2024. Tool-augmented reward modeling. In *Proceedings of ICLR*.

Sheng-Chieh Lin, Luyu Gao, Barlas Oguz, Wenhan Xiong, Jimmy Lin, Wen-tau Yih, and Xilun Chen. 2024. Flame: Factuality-aware alignment for large language models. *arXiv preprint arXiv:2405.01525*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of ACL*, pages 3214–3252.

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

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. 2024b. Rm-bench: Benchmarking reward models of language models with subtlety and style. *arXiv preprint arXiv:2410.16184*.

Xingzhou Lou, Dong Yan, Wei Shen, Yuzi Yan, Jian Xie, and Junge Zhang. 2024. Uncertainty-aware reward model: Teaching reward models to know what is unknown. *arXiv preprint arXiv:2410.00847*.

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

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

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of EMNLP*, pages 12076–12100.

Tong Mu, Alec Helyar, Johannes Heidecke, Joshua Achiam, Andrea Vallone, Ian Kivlichan, Molly Lin, Alex Beutel, John Schulman, and Lilian Weng. 2024. Rule based rewards for language model safety. *arXiv preprint arXiv:2411.01111*.

Rémi Munos, Michal Valko, Daniele Calandriello, Mohammad Gheshlaghi Azar, Mark Rowland, Zhaohan Daniel Guo, Yunhao Tang, Matthieu Geist, Thomas Mesnard, Andrea Michi, et al. 2023. Nash learning from human feedback. *arXiv preprint arXiv:2312.00886*.

OpenAI. 2024a. Gpt-4o mini: Advancing cost-efficient intelligence. Accessed: 2025-02-04.

OpenAI. 2024b. Hello gpt-4o. Accessed: 2025-02-04.

OpenAI. 2025. Openai o3 mini. Accessed: 2025-02-15.

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

Junsoo Park, Seungyeon Jwa, Ren Meiying, Daeyoung Kim, and Sanghyuk Choi. 2024. Offsetbias: Leveraging debiased data for tuning evaluators. In *Findings of EMNLP*, pages 1043–1067.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2024. Constraint back-translation improves complex instruction following of large language models. *arXiv preprint arXiv:2410.24175*.

Lingfei Qian, Weipeng Zhou, Yan Wang, Xueqing Peng, Jimin Huang, and Qianqian Xie. 2025. Fino1: On the transferability of reasoning enhanced llms to finance. *arXiv preprint arXiv:2502.08127*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Keita Saito, Akifumi Wachi, Koki Wataoka, and Youhei Akimoto. 2023. Verbosity bias in preference labeling by large language models. *arXiv preprint arXiv:2310.10076*.

Tu Shiwen, Zhao Liang, Chris Yuhao Liu, Liang Zeng, and Yang Liu. 2024. Skywork critic model series. https://huggingface.co/Skywork.

Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. 2023. A long way to go: Investigating length correlations in rlhf. *arXiv preprint arXiv:2310.03716*.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. Defining and characterizing reward gaming. *Advances in Neural Information Processing Systems*, 35:9460–9471.

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

Hao Sun, Yunyi Shen, and Jean-Francois Ton. 2024a. Rethinking bradley-terry models in preference-based reward modeling: Foundations, theory, and alternatives. *arXiv preprint arXiv:2411.04991*.

Haoran Sun, Lixin Liu, Junjie Li, Fengyu Wang, Baohua Dong, Ran Lin, and Ruohui Huang. 2024b. Conifer: Improving complex constrained instruction-following ability of large language models. *arXiv preprint arXiv:2404.02823*.

Gokul Swamy, Christoph Dann, Rahul Kidambi, Zhiwei Steven Wu, and Alekh Agarwal. 2024. A minimaximalist approach to reinforcement learning from human feedback. *arXiv preprint arXiv:2401.04056*.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. 2024. Judgebench: A benchmark for evaluating llm-based judges. *arXiv preprint arXiv:2410.12784*.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, et al. 2023. Zephyr: Direct distillation of lm alignment. *arXiv preprint arXiv:2310.16944*.

Haoxiang Wang, Wei Xiong, Tengyang Xie, Han Zhao, and Tong Zhang. 2024a. Interpretable preferences via multi-objective reward modeling and mixture-of-experts. *arXiv preprint arXiv:2406.12845*.

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

Zhilin Wang, Yi Dong, Olivier Delalleau, Jiaqi Zeng, Gerald Shen, Daniel Egert, Jimmy J Zhang, Makesh Narsimhan Sreedhar, and Oleksii Kuchaiev. 2024c. Helpsteer2: Open-source dataset for training top-performing reward models. *arXiv preprint arXiv:2406.08673*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*.

Yue Yu, Zhengxing Chen, Aston Zhang, Liang Tan, Chenguang Zhu, Richard Yuanzhe Pang, Yundi Qian, Xuewei Wang, Suchin Gururangan, Chao Zhang, et al. 2024. Self-generated critiques boost reward modeling for language models. *arXiv preprint arXiv:2411.16646*.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. 2024. Advancing llm reasoning generalists with preference trees. *Preprint*, arXiv:2404.02078.

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

Xuanchang Zhang, Wei Xiong, Lichang Chen, Tianyi Zhou, Heng Huang, and Tong Zhang. 2024b. From lists to emojis: How format bias affects model alignment. *arXiv preprint arXiv:2409.11704*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Appendices

A REWARDAGENT Details

Tables 4 to 6 present the LLM prompts used for the implementation of REWARDAGENT. We employed Serper[2] to implement our external search engine and we utilize the `gpt-4o-mini-2024-07-18` model in the $\text{REWARDAGENT}_{\text{MINI}}$ version.

B Experimental Details

In this section, we provide a detailed description of the evaluation process, divided into three parts: the construction and distribution details of IFBENCH B.1, the evaluation dataset settings B.2, and additional experimental results B.3.

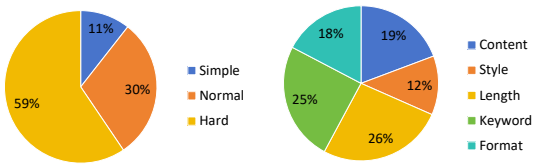
B.1 IFBENCH Details

IFBENCH is a benchmark designed to evaluate reward models for multi-constraint instruction-following. The dataset comprises 444 carefully curated instances, each containing: an instruction with 3 to 5 multi-constraints, a chosen response satisfying all constraints, and a rejected response violating specific constraints. All instances were constructed using `gpt-4o-2024-11-20` version through the following systematic pipeline.

Instruction Construction We sampled 500 initial instructions from the Open Assistant (Köpf et al., 2023). To ensure clarity and simplicity, we constrained the initial instruction length to 5 to 20 words. Subsequently, we employed GPT-4o to generate five distinct categories of constraints for each initial instruction. It then autonomously selected 3 to 5 constraints and paraphrased them into 1 to 2 sentences. The paraphrased constraints were integrated into the initial instruction. Finally, we use GPT-4o to evaluate the final instructions and filter out those with internal contradictions, resulting in a final set of 444 instructions.

- **Content Constraints:** Specify conditions governing response, including topic focus, detail depth, and content scope limitations.
- **Style Constraints:** Control linguistic characteristics such as tone, sentiment polarity, empathetic expression, and humor.
- **Length Constraints:** Dictate structural requirements including word counts, paragraph composition, and specific opening phrases.
- **Keyword Constraints:** Enforce lexical constraints through keyword inclusion, prohibited terms, or character-level specifications.
- **Format Constraints:** Define presentation standards that include specific formats such as JSON, Markdown, or Python, along with section organization and punctuation rules.

Response Construction For each instruction, we generated 8 candidate responses using GPT-4o with temperature 1.0 to maximize diversity. The chosen response was selected as the unique candidate satisfying all constraints through automated verification. Rejected responses were systematically selected to ensure balanced distributions of unsatisfied constraint (UC) categories and counts. As shown in Figure 4, instances are stratified by difficulty: simple (#UC≥3), normal (#UC=2), and hard (#UC=1), with detailed information of UC category distributions. Specifically, (a) shows the distribution by the number of unsatisfied constraints in the rejected responses, where the sum of all parts equals the total number of instances. (b) presents the distribution by the categories of all unsatisfied constraints, where the sum of all parts equals the total number of unsatisfied constraints.



(a) (b)

Figure 4: Proportion (%) of data in IFBENCH based on the number of unsatisfied constraints per instance and the categories of all unsatisfied constraints.

B.2 Evaluation Details

Best-of-N For the TriviaQA, we sample 500 instances from the validation split in `rc.nocontext` version. The model is prompted to generate direct answers, and we report the exact match accuracies. For the IFEval, we report the average accuracy across the strict prompt, strict instruction, loose prompt, and loose instruction settings. For the CELLO, we report the average score based on the official evaluation script. All three tasks are conducted under a zero-shot setting.

[2] https://serper.dev/

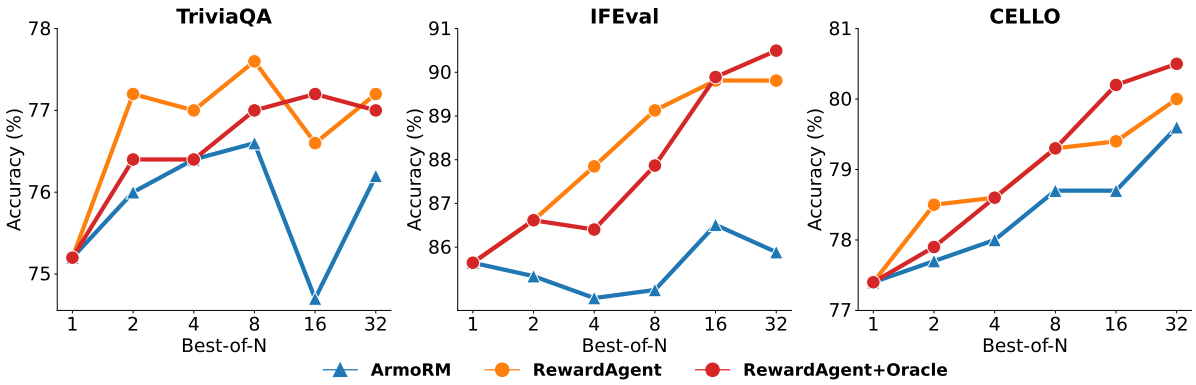


Figure 5: Best-of-n results (%) on TriviaQA, IFEval, and CELLO using the base reward model ArmoRM and REWARDAGENT to search. "+Oracle" denotes using the oracle setting of REWARDAGENT as mentioned in § 4.3.

DPO Training For MT-Bench and CELLO, we employ FastChat[3] and the official evaluation script respectively, to conduct the evaluations and report the average scores. For the other tasks, we use the `lm-evaluation-harness`[4] for evaluation. Specifically, we adopt a 5-shot setting for the MMLU and MMLU-Pro tasks, while using a zero-shot setting for TriviaQA and TruthfulQA. Notably, for TruthfulQA, we use the `truthfulqa_gen` setting.

B.3 More Results on Best-of-N

We conduct best-of-n search experiments using `gpt-4o-2024-11-20` as the policy model, with the results presented in Figure 5. The results demonstrate that REWARDAGENT significantly improves best-of-n performance compared to the base reward model ArmoRM, even when applied to a more powerful policy model than REWARDAGENT.

[3] https://github.com/lm-sys/FastChat/tree/main/fastchat/llm_judge

[4] https://github.com/EleutherAI/lm-evaluation-harness

Given the following instruction, determine whether the following check in needed.

[Instruction]
{instruction}

[Checks]
{ “name”: “constraint check”, “desp”: “A ‘constraint check’ is required if the instruction contains any additional constraints or requirements on the output, such as length, keywords, format, number of sections, frequency, order, etc.”, “identifier”: “[[A]]” },
{ “name”: “factuality check”, “desp”: “A ‘factuality check’ is required if the generated response to the instruction potentially contains claims about factual information or world knowledge.”, “identifier”: “[[B]]” }

If the instruction requires some checks, please output the corresponding identifiers (such as [[A]], [[B]]).
Please do not output other identifiers if the corresponding checkers not needed.

Table 4: Our prompt for the router, where the {instruction} part varies based on the input.

Prompt For Difference Proposal
[Answers]
{formatted_answers}

[Your Task]
Given the above responses, please identify and summarize one key points of contradiction or inconsistency between the claims.

[Requirements]
1. Return a Python list containing only the most significant differences between the two answers.
2. Do not include any additional explanations, only output the list.
3. If there are no inconsistencies, return an empty list.

Prompt For Query Generation
[Original question that caused the inconsistency]
{instruction}

[Inconsistencies]
{inconsistencies}

[Your Task]
To resolve the inconsistencies, We need to query search engine. For each contradiction, please generate a corresponding query that can be used to retrieve knowledge to resolve the contradiction.

[Requirements]
1. Each query should be specific and targeted, aiming to verify or disprove the conflicting points.
2. Provide the queries in a clear and concise manner, returning a Python list of queries corrresponding to the inconsistencies.
3. Do not provide any additional explanations, only output the list.

Prompt For Verification
Evaluate which of the two answers is more factual based on the supporting information.
[Support knowledge sources]:
{supports}

[Original Answers]:
{formatted_answers}

[Remeber]
For each answer, provide a score between 1 and 10, where 10 represents the highest factual accuracy. Your output should only consist of the following:
Answer A: [[score]] (Wrap the score of A with [[and]])
Answer B: «score» (Wrap the score of B with « and »)
Please also provide a compact explanation.

Table 5: Our prompt for assessing factuality in verification agents, with the {formatted_answers}, {supports}, {inconsistencies}, {instruction} and {supports} parts varying based on the input.

Prompt For Constraint Parsing
You are an expert in natural language processing and constraint checking. Your task is to analyze a given instruction and identify which constraints need to be checked.

The 'instruction' contains a specific task query along with several explicitly stated constraints. Based on the instructions, you need to return a list of checker names that should be applied to the constraints.

Task Example:
Instruction: Write a 300+ word summary of the Wikipedia page "https://en.wikipedia.org/wiki/Raymond_III,_Count_of_Tripol". Do not use any commas and highlight at least 3 sections that have titles in markdown format, for example, *highlighted section part 1*, *highlighted section part 2*, *highlighted section part 3*.
Response:
NumberOfWordsChecker: 300+ word
HighlightSectionChecker: highlight at least 3 sections that have titles in markdown format
ForbiddenWordsChecker: Do not use any commas

Task Instruction:
{instruction}

Your task:
- Generate the appropriate checker names with corresponding descriptions from the original instruction description.
- Return the checker names with their descriptions separated by '\n'
- Focus only on the constraints explicitly mentioned in the instruction (e.g., length, format, specific exclusions).
- Do **not** generate checkers for the task query itself or its quality.
- Do **not** infer or output constraints that are implicitly included in the instruction (e.g., general style or unstated rules).
- Each checker should be responsible for checking only one constraint.

Prompt For Code Generation
You are tasked with implementing a Python function 'check_following' that determines whether a given 'response' satisfies a constraint defined by a checker. The function should return 'True' if the constraint is satisfied, and 'False' otherwise.

[Instruction to check]:
{instruction}

[Specific Checker and Description]:
{checker_name}

Requirements:
- The function accepts only one parameter: 'response' which is a Python string.
- The function must return a boolean value ('True' or 'False') based on whether the 'response' adheres to the constraint described by the checker.
- The function must not include any I/O operations, such as 'input()' or 'ArgumentParser'.
- The Python code for each checker should be designed to be generalizable, e.g., using regular expressions or other suitable techniques.
- Only return the exact Python code, with no additional explanations.

Table 6: Our prompt for assessing instruction-following in verification agents, with the {instruction} and {checker_name} parts varying based on the input.