ULTRALOGIC: ENHANCING LLM REASONING THROUGH LARGE-SCALE DATA SYNTHESIS AND BIPOLAR FLOAT REWARD

Anonymous authors
Paper under double-blind review

ABSTRACT

While Large Language Models (LLMs) have demonstrated significant potential in natural language processing, complex general-purpose reasoning—requiring multi-step logic, planning, and verification—remains a critical bottleneck. Although Reinforcement Learning with Verifiable Rewards (RLVR) has succeeded in specific domains, the field lacks large-scale, high-quality, and difficulty-calibrated data for general reasoning. To address this, we propose ULTRALOGIC, a framework that decouples the logical core of a problem from its natural language expression through a Code-based Solving methodology to automate high-quality data production. The framework comprises hundreds of unique task types and an automated calibration pipeline across ten difficulty levels, which is dynamically adjustable to remain effective and future-proof as base model capabilities evolve. Through feedback-guided task refinement and difficulty recalibration, it supports a data-centric approach to recursive self-improvement (RSI), producing verifiable training data matched to evolving model capabilities. Furthermore, to mitigate binary reward sparsity and the Non-negative Reward Trap, we introduce the Bipolar Float Reward (BFR) mechanism, utilizing graded penalties to effectively distinguish perfect responses from those with logical flaws. Our experiments demonstrate that BFR, combined with a difficulty matching strategy, significantly improves training efficiency, guiding models toward consistent logical policies.

1 INTRODUCTION

In recent years, Large Language Models (LLMs) have achieved revolutionary breakthroughs in numerous areas of Natural Language Processing (NLP) (Achiam et al., 2023; DeepSeek-AI et al., 2025; Yang et al., 2025). However, complex reasoning—particularly general-purpose reasoning that requires multi-step logic, planning, and verification—remains a critical bottleneck in advancing model intelligence to higher levels (Liu et al., 2025b; Wu et al., 2025). To overcome this limitation, both academia and industry have turned their attention to the post-training stage, especially methods based on Reinforcement Learning (RL) (Ouyang et al., 2022; Rafailov et al., 2023). These models have demonstrated astonishing improvements in reasoning abilities in domains like mathematics and code by leveraging Reinforcement Learning with Verifiable Rewards (RLVR) (Shao et al., 2024). The core of this paradigm lies in the fact that task answers in these domains (e.g., whether code passes unit tests or a mathematical answer is correct) have explicit, automatically verifiable feedback, providing a clear reward signal for reinforcement learning.

However, when extending this paradigm from specific domains to broader, more general-purpose reasoning tasks, a fundamental problem becomes particularly prominent: the insufficient supply of high-quality training data (Liu et al., 2025a; Lu et al., 2024; Liu et al., 2025b). The success of RLVR currently relies heavily on existing high-quality competition datasets (Hendrycks et al., 2021; Rein et al., 2024), but the field of general reasoning lacks similar large-scale data resources. Existing datasets are not only limited in task diversity (Liu et al., 2025a; Li et al., 2025a), failing to cover a wide range of reasoning scenarios, but they also generally lack a clear and controllable difficulty calibration system (Kwan et al., 2025; Wang et al., 2025). This makes the difficulty distribution of the training data hard to manage, which in turn affects the model's learning efficiency and stability.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To systematically address this data supply problem, we propose and implement ULTRALOGIC: a data-centric framework supporting recursive self-improvement (RSI) through feedback-guided refinement of verifiable training tasks. Built upon a Code-based Solving methodology, the framework combines large-scale data generation with difficulty recalibration as model capabilities evolve. This framework operates through a process that combines human definition with automated production. First, in a collaborative process between domain experts and prompt-driven LLMs, two critical Python functions are written for each novel reasoning task type: an input function to generate the slot-filling data for predefined problem templates, and a solution function to provide a deterministic ground-truth answer based on that data. This design grounds instance generation and answer computation in shared task logic, providing a foundation for programmatic verification and difficulty control. Subsequently, an automated pipeline takes over, performing large-scale generalization on these "seed tasks" using Programmatic Expansion (PE) techniques (Mishra et al., 2022). This pipeline not only generates a vast number of problem variants but also systematically creates task instances spanning 10 difficulty levels by precisely controlling the parameters of the input function. These difficulty levels are then objectively calibrated against the measured success rates of a flagship model. Experiments show that by training on our synthesized data alone, using a standard binary reward mechanism (Shao et al., 2024), the model exhibits a significant improvement in general reasoning capabilities compared to baselines.

Building on this foundation, we further explore potential methods for improving training efficiency, which constitutes our second exploratory contribution. We hypothesize that although the binary reward mechanism is clear and effective, providing the learning process with a more information-dense and graded reward signal could potentially enhance both training efficiency and final performance. To this end, we design and implement a novel Bipolar Float Reward (BFR) mechanism, which introduces a penalty-driven optimization signal to the reinforcement learning process. This mechanism is capable of quantifying "partially correct" outputs based on task-specific characteristics, using metrics such as accuracy or F1-Score (Paulus et al., 2018). Our experiments demonstrate that this approach significantly outperforms both binary and standard non-negative float rewards, facilitating faster convergence and superior final performance by effectively penalizing imperfect reasoning paths.

The contributions of this paper are twofold. First, we introduce ULTRALOGIC, a general reasoning data framework that combines task diversity, programmatic generation, and fine-grained difficulty calibration. By decoupling the logical core of a problem from its natural language expression, the framework enables large-scale synthesis of verifiable reasoning data and supports training matched to model capacity through a ten-level difficulty hierarchy. Second, we propose Bipolar Float Reward (BFR), which combines partial-correctness feedback with a full-correctness bonus to provide fine-grained verifiable rewards for complex reasoning tasks. Our experiments demonstrate that ULTRALOGIC improves general reasoning and further validate the performance advantages of BFR over binary and standard float rewards. Our dataset and code are available at: https://anonymous.4open.science/r/UltraLogic.

2 RELATED WORK

Data Synthesis for Logic and Reasoning. Programmatic frameworks construct verifiable reasoning data using task-specific generators and verifiers. SynLogic (Liu et al., 2025a) covers 35 logical tasks and controls difficulty through task parameters and model-based filtering. Reasoning Gym (Stojanovski et al., 2025) provides over 100 reasoning environments with configurable complexity and investigates curriculum RLVR. LLM-based approaches include ScaleQuest for mathematical question synthesis (Ding et al., 2025), Chimera for STEM problems and solution trajectories (Zhu et al., 2026), and MathSmith for reinforcement-optimized mathematical problem generation (Zhan et al., 2026). ULTRALOGIC combines decoupled input/solution logic, bilingual templates, a model-calibrated ten-level difficulty ladder, and BFR for structured final answers. A detailed comparison is provided in Section G.

Fine-Grained Reward Mechanisms. To alleviate the sparsity of binary rewards, Process Reward Models (PRMs) (Lightman et al., 2024) provide fine-grained feedback on intermediate reasoning steps (Ma et al., 2023), while methods such as OpenPRM (Zhang et al., 2025) and DG-PRM (Yin et al., 2025) further explore the automated construction of process supervision signals. ULTRALOGIC's BFR uses the verifiable structure of final answers to quantify correctness through task-specific partial

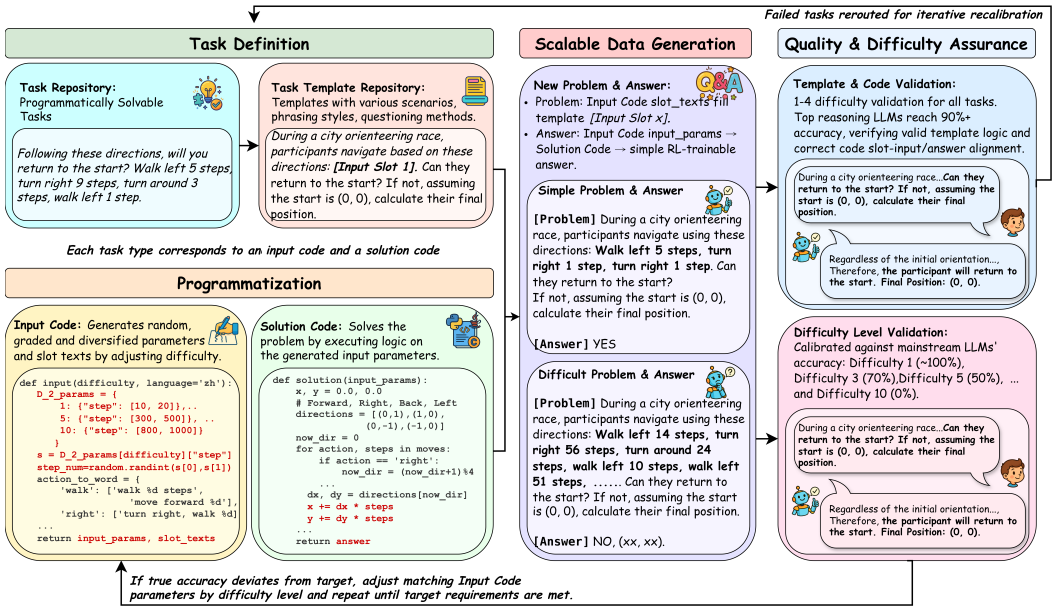

Figure 1: The overall architecture of the ULTRALOGIC data framework. Feedback-driven task refinement and difficulty calibration form an iterative data-generation loop that supports recursive self-improvement.

matching criteria and provides an additional reward for fully correct answers. This mechanism offers fine-grained outcome supervision for verifiable reasoning tasks without step-level reasoning annotations or additional reward model training.

Role of Difficulty in Reinforcement Learning. Difficulty profoundly impacts LLM training efficiency. Methods like OpenSIR (Kwan et al., 2025) and MorphoBench (Wang et al., 2025) dynamically adjust difficulty for training and evaluation. Consequently, curriculum learning frameworks like E3-RL4LLMs (Liao et al., 2025) and SEELE (Li et al., 2025b) actively maintain "high-efficiency regions." ULTRALOGIC addresses this via an automated Levels 1–10 calibrated ladder. Furthermore, we identify a "Difficulty Matching Phenomenon," demonstrating that RL peaks within the "Zone of Proximal Development" where difficulty precisely aligns with model capacity.

3 THE ULTRALOGIC DATA FRAMEWORK

To address the shortage of high-quality general reasoning data, ULTRALOGIC automates scalable data production with diverse tasks and controllable difficulty. This section describes its architecture, synthesis pipeline, and difficulty calibration.

3.1 OVERALL ARCHITECTURE

ULTRALOGIC utilizes a "Code-based Solving Framework" that decouples a problem's *logical core* from its *natural language expression*. This allows independent, programmatic control over both intrinsic complexity and external presentation, enabling massive diverse data generation.

As illustrated in Figure 1, the framework comprises three static components: an Original Task Repository, a Diverse Task Template Repository, and a core Data Synthesis Pipeline (consisting of Input Code, Solution Code, and a Difficulty Control Module). The workflow selects a task, matches it with a template, and programmatically generates a concrete problem instance featuring a deterministic answer and an objective difficulty rating.

3.2 ORIGINAL TASK REPOSITORY

The diversity of ULTRALOGIC is grounded in a repository of reusable reasoning tasks. We provide generators, metadata, and task-specific scoring functions for 100 tasks. Our training experiments use 50 task types, with 10,000 samples in each difficulty-stratified dataset. To ensure a systematic and novel coverage of reasoning skills, we developed a three-dimensional orthogonal classification system. Each task is uniquely situated along three dimensions: Task Domain (defining the problem context), Core Reasoning Ability (characterizing the cognitive focal point), and Difficulty Setup (dissecting the source of complexity). We constructed a massive question bank from these types and performed rigorous data cleaning to exclude tasks similar to existing benchmarks. Detailed taxonomies and category definitions are provided in Section B.1. The detailed task category cases are provided in Section B.2.

3.3 DIVERSE TASK TEMPLATE REPOSITORY

To express logical cores as human-readable problems, an LLM first assesses each task's suitability for programmatic generation. For suitable tasks, an automated LLM workflow builds Chinese and English templates for bilingual training.

The workflow has two stages. First, given an original reasoning problem, an LLM identifies variable parameters (e.g., numbers, names, or operation sequences) and invariant logic, then creates a base template with slots such as *initial state* and *operation sequence* (Section D.1). We also incorporate logical structures inspired by competitive programming platforms such as LeetCode; these tasks require no LLM-based deconstruction and offer verified solution code and deterministic ground-truth answers. Second, an LLM expands each base template into multiple (e.g., 10) narrative variants, such as logistics scenarios, spy stories, or sci-fi settings, while preserving its logic and slots (Section D.2). The variants are automatically aggregated into a standardized template library and iteratively revised by the downstream quality assurance module if textual or logical flaws are detected.

3.4 DATA SYNTHESIS PIPELINE

3.4.1 INPUT CODE GENERATION

We define the core input code, implemented as an `input` function. The creation of this function is an automated process designed for scalability: the task logic and structural parameters identified during the template deconstruction phase serve as the specification; a prompt-driven LLM then generates the full Python function based on this specification (the detailed prompt is provided in Section D.3).

The primary responsibility of the final `input(difficulty, language)` function is to programmatically generate the core parameters (or "slot-filling data") for populating the template slots. It takes a difficulty level `difficulty` and a target language `language` as input and contains a set of rules for generating parameters related to the intrinsic logic of the task. Its output is not a complete problem, but rather a structured set of data that constitutes the unique logical core of that problem instance. Crucially, since these parameters are generated stochastically within defined constraints, the framework can theoretically produce an infinite supply of unique reasoning problems for any given task type.

3.4.2 SOLUTION CODE GENERATION

For each task, an LLM generates a `solution(params, language)` function that deterministically computes the ground-truth answer and is checked and revised through quality control (Section D.4). Sharing `params` with instance generation keeps problem descriptions and solutions consistent, integrating repeatable answer computation into large-scale programmatic data production.

3.4.3 DATA VALIDATION AND QUALITY ASSURANCE

Before large-scale production, we automatically validate task types and their templates to check consistency between programmatic logic and natural language expression. Specifically, we generate representative samples covering different template variants at lower difficulty tiers (Levels 1–4) and use a flagship model to evaluate their solvability. For tasks that fall below the predefined success

threshold (above 90%), the system initiates a ReAct-style feedback loop, diagnoses problems in failed samples, and guides the LLM to revise the templates, `input` code, or `solution` code. This process brings template expression, instance generation, and answer computation into a unified quality control stage, screening for and correcting logical and linguistic issues before difficulty calibration and large-scale production.

3.4.4 DIFFICULTY CONTROL MODULE

The Difficulty Control Module constructs a unified 1–10 difficulty ladder through closed-loop calibration, linking task complexity to the actual problem-solving performance of calibration models. First, we define target success rates $P(d)$ for specific levels (e.g., approximately 100%, 70%, 50%, 30%, and 0% for levels 1, 3, 5, 7, and 10, respectively). The system then generates test samples via the `input` and `solution` functions and uses multiple open-source flagship models to calculate the actual average success rate P_{actual}. For configurations that deviate from the target, the system follows a ReAct-style iteration (Yao et al., 2022) to adjust task complexity parameters, such as the number of reasoning steps and constraints, until the actual success rate falls within the target tolerance.

The resulting difficulty levels are grounded in the success rates of calibration models under specified evaluation conditions and implemented in data production through fixed generation parameters. These configurations support batch generation at a specified difficulty. From a data-centric RSI perspective, model feedback drives repeated refinement of the task-generation process through task revision and difficulty recalibration. Reusing this process as model capabilities evolve provides verifiable data for subsequent learning. The reported experiments use fixed datasets produced by this process, evaluating its data and reward components. The prompt guiding the automated adjustment is provided in Section D.5.

A complete metadata example of a task instance is provided in Section B.3.

4 BIPOLAR FLOAT REWARDS

To mitigate the sparsity and poor discrimination of binary rewards, we introduce the Bipolar Float Reward (BFR) mechanism. This chapter details the BFR's design, utilizing dense graded feedback to overcome training bottlenecks in multi-step reasoning.

4.1 BASELINE TRAINING PARADIGM AND THE LIMITATIONS OF BINARY REWARDS

We adopt Reinforcement Learning with Verifiable Rewards (RLVR) as our core training paradigm and choose Group Relative Policy Optimization (GRPO) (Shao et al., 2024) as our primary policy optimization algorithm. In our baseline experiments, we initially used a standard binary reward function: the reward is 1 only if the final answer generated by the model is identical to the ground truth, and 0 otherwise.

However, for complex multi-step reasoning tasks, binary rewards are overly sparse and lack discrimination. A near-correct answer with minor errors receives the same zero reward as a completely wrong response. Failing to reflect the "degree of correctness" wastes valuable exploratory actions and reduces learning efficiency.

4.2 DESIGN AND ITERATION OF THE GRADED FLOAT REWARD

To overcome the limitations of binary rewards, we hypothesize that a more information-dense, graded reward signal that reflects the "degree of correctness" could provide the model with richer and more fine-grained learning guidance, thereby accelerating convergence and improving final performance. Based on this motivation, we designed and implemented a novel Graded Float Reward mechanism.

4.2.1 INITIAL EXPLORATION: GRADED FLOAT REWARD IN THE [0, 1] RANGE

We first extend rewards from the discrete $\{0, 1\}$ set to the continuous $[0, 1]$ interval to capture partial correctness. Specifically, we organize answers into verifiable structured outputs and score them using Accuracy, F1-score, Similarity, or Absolute Difference Rate according to task characteristics

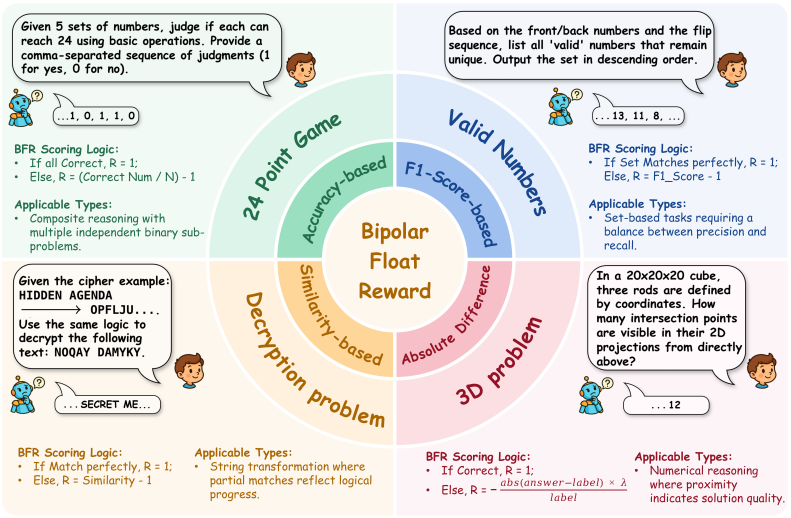

Figure 2: Four scoring methods of the BFR mechanism and illustrative examples. Each task type is matched with the most appropriate scoring method to effectively quantify the correctness of model responses based on its unique characteristics, mapping partially correct answers into graded negative penalties.

(see Figure 2 for detailed logic and examples). This design distinguishes different degrees of error, providing fine-grained feedback for answers that receive the same score under binary rewards. However, under standard Float rewards, the reward gap between a fully correct answer and a near-correct answer shrinks as the latter's score approaches 1. For reasoning tasks that require complete satisfaction of logical constraints, we therefore introduce a separate bonus for fully correct answers, combining partial-correctness feedback with the objective of complete solutions.

4.2.2 Bipolar Float Reward: Graded Feedback and Full-Correctness Incentives

BFR combines partial correctness with a full-correctness bonus. Let $S \in [0, 1]$ denote the correctness score assigned by a task-specific scoring function. The correctness reward is 1 for a fully correct answer and $S - 1$ otherwise. This mapping preserves fine-grained differences among imperfect answers and introduces a distinct reward cliff between fully correct and imperfect answers. We optimize the policy using GRPO, whose group-normalized advantage $\hat{A}_{i,g}$ enters the following surrogate objective:

$$\mathcal{J}_{GRPO}(\theta) = \frac{1}{G}\sum_{i=1}^{G}\Big[\min\left(r_i(\theta)\hat{A}_{i,g}, \text{clip}(r_i(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{i,g}\right) - \beta\mathbb{D}_{KL}\Big]$$
$$r_i(\theta) = \frac{\pi_\theta(a_i|s)}{\pi_{\theta_{\text{old}}}(a_i|s)}, \qquad \hat{A}_{i,g} = \frac{R_i - \text{mean}(\{R_1, \ldots, R_G\})}{\text{std}(\{R_1, \ldots, R_G\})} \tag{1}$$

where $r_i(\theta)$ denotes the importance sampling ratio, ϵ is the clipping hyperparameter, and $\beta\mathbb{D}_{KL}$ represents the KL divergence penalty against a reference model.

Reward Cliff and Full-Correctness Bonus. The effect of BFR can be expressed directly through the following equivalent form:

$$R_{\text{BFR}}(S) + 1 = S + \mathbf{1}[S = 1]. \tag{2}$$

Since group normalization is invariant to a common translation, BFR yields the same standardized advantages as a correctness score augmented with a full-correctness bonus. Compared with standard Float rewards, the reward gap between a fully correct answer and an answer with score $S < 1$ increases from $1 - S$ to $2 - S$, maintaining a clear reward separation between near-correct and fully correct answers.

Table 1: Comparison of supervision signals and computational requirements between PRMs and BFR.

Dimension	PRMs	BFR (Ours)
Supervision signal	Scores intermediate reasoning steps.	Scores final answers by fine-grained correctness and rewards fully correct answers.
Scorer construction	Builds or adopts a process reward model.	Configures task-specific deterministic verification functions; no additional reward model training.
Scoring overhead during training	Process reward model inference.	Verification function computation; no reward model inference.

In sampled groups containing fully correct answers and imperfect answers with different correctness scores, this mapping redistributes relative advantages within the group, combining fine-grained answer evaluation with reinforcement of complete solutions. BFR thus provides two complementary signals: graded feedback that distinguishes partially correct answers, and an additional incentive for fully correct answers. Section F further analyzes the relationship between reward scaling and the allocation of group-relative advantages. All these rewards are computed by task-specific verification functions, without training an additional reward model (see Table 1).

5 Experimental Setup

To evaluate the UltraLogic framework and BFR mechanism, we conduct ablation studies using Qwen3-8B and 14B models, utilizing their unified reasoning architecture in default thinking mode[1]. All UltraLogic runs use only synthetic data generated by the framework.

5.1 Training Configuration and Benchmarks

Training. All training utilizes the GRPO algorithm with consistent hyperparameters: $lr = 1e-6$, rollout $= 16$, max_response_length $= 32,768$, $temperature = 1.0$, and $top_p = 1.0$. The experimental rewards combine the correctness reward defined in Section 4 with formatting checks and a 0.1-point language-consistency bonus. A formatting failure receives the minimum reward under the corresponding reward scheme (-1 or 0); otherwise, the total reward is the correctness reward plus the language bonus, with a maximum of 1.1. The final results in Section 6 were obtained after training for two epochs.

Evaluation. We evaluate models on five benchmarks: AIME (2024 & 2025)[2], HMMT 2025[3], BBH (Suzgun et al., 2022), BBEH (Kazemi et al., 2025), and ARC-AGI[4]. Crucially, benchmarks such as AIME and ARC-AGI present a significant domain shift from our textual logical training data, thereby validating the models' robust cross-domain transfer and out-of-distribution (OOD) reasoning capabilities. Evaluation employs a sampling strategy ($T = 0.6$, `top_p`=0.95, `top_k`=20, `min_p`=0) with a 32,768 token limit. We report accuracy averaged over 64 samples (avg@64), with each sampled final answer evaluated individually for correctness.

5.2 Ablation Study Design

Difficulty Matching. This study examines model-dependent preferences for training difficulty. We train Qwen3-8B and Qwen3-14B on three fixed difficulty-stratified datasets spanning levels 1–4, levels 4–7, and levels 7–10. Each dataset contains 10,000 samples in total across 50 task types. All runs in this study use the standard $\{0, 1\}$ binary correctness reward.

[1] https://huggingface.co/collections/Qwen/qwen3

[2] https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions

[3] https://www.hmmt.org/www/archive/problems

[4] https://arcprize.org/arc-agi/1/

Table 2: Comprehensive evaluation of ULTRALOGIC across different dimensions. Panel A compares our dynamic difficulty subsets against LogicPro. Panel B explores the scaling law of difficulty on a larger model capacity. Panel C demonstrates the superiority of the Bipolar Float Reward over traditional reward mechanisms. Bold and underline indicate the best and second-best results within their respective panels.

Setting / Model	Difficulty / Reward	AIME 24	AIME 25	HMMT 25	BBH	BBEH	ARC-AGI
Panel A: Difficulty Matching & Baseline Comparison (Base: Qwen3-8B, Reward: Binary)							
Base Model	N/A	75.2	66.1	47.1	88.2	29.2	4.0
LogicPro	Easy	75.1	63.3	47.6	88.5	29.6	4.1
LogicPro	Medium	73.3	64.1	46.7	88.5	29.1	3.9
LogicPro	Hard	75.1	65.8	47.3	88.6	29.4	4.0
ULTRALOGIC (Ours)	Levels 1–4	**81.7**	**69.1**	**52.3**	**90.2**	**31.1**	**4.6**
ULTRALOGIC (Ours)	Levels 4–7	<u>77.6</u>	<u>67.6</u>	<u>50.9</u>	<u>88.7</u>	<u>29.6</u>	<u>4.2</u>
ULTRALOGIC (Ours)	Levels 7–10	76.5	65.1	50.0	88.3	29.0	4.0
Panel B: Difficulty Matching Scaling (Base: Qwen3-14B, Reward: Binary)							
Base Model	N/A	76.7	70.3	54.3	89.6	32.2	6.3
ULTRALOGIC (Ours)	Levels 1–4	78.8	72.5	58.6	<u>90.9</u>	33.7	6.1
ULTRALOGIC (Ours)	Levels 4–7	**82.4**	**75.8**	**63.1**	**92.1**	**36.7**	**7.9**
ULTRALOGIC (Ours)	Levels 7–10	<u>80.8</u>	<u>73.8</u>	<u>59.9</u>	<u>90.9</u>	<u>34.7</u>	<u>6.8</u>
Panel C: Reward Mechanism Ablation (Base: Qwen3-8B, Difficulty: Levels 1–4)							
ULTRALOGIC	Binary Reward	<u>81.7</u>	<u>69.1</u>	52.3	90.2	<u>31.1</u>	<u>4.6</u>
ULTRALOGIC	Graded Float	76.9	66.3	<u>53.0</u>	<u>90.4</u>	31.0	4.3
ULTRALOGIC (Ours)	**Bipolar Float (BFR)**	**82.6**	**71.3**	**56.6**	**91.1**	**32.5**	**4.7**

Bipolar Float Reward. To validate BFR, we compare three reward schemes: Binary ($\{0, 1\}$), Graded Float ($[0, 1]$), and BFR ($[-1, 0) \cup \{1\}$). Each experiment uses 50 tasks (10k samples), with Qwen3-8B trained on only Levels 1–4 data.

6 RESULTS AND ANALYSIS

Using the experimental setup above, we analyze how difficulty matching and Bipolar Float Reward affect ULTRALOGIC's reasoning performance through two ablation studies. Detailed results are provided in Section C.

6.1 DIFFICULTY MATCHING: THE SCALING LAW OF DIFFICULTY

We identified the Difficulty Matching Phenomenon in the Qwen3-8B and 14B: model capacity dictates sensitivity to training data difficulty, exhibiting a strong positive correlation. The optimal training difficulty shifts upward as model size increases, indicating that matching training difficulty to model capacity is important for realizing the full training benefits of synthetic reasoning data.

The difference is reflected in the benchmark results in Table 2. On HMMT25, Qwen3-8B improves from 47.1 to 52.3 with Levels 1–4 data, compared with 50.9 and 50.0 for Levels 4–7 and Levels 7–10. For Qwen3-14B, Levels 4–7 achieves 63.1, compared with 58.6 for Levels 1–4 and 59.9 for Levels 7–10. The preferred difficulty band also achieves the highest reported accuracy on the other benchmarks for each model. These findings support model-dependent difficulty selection across the two evaluated model scales and three difficulty bands.

Regarding training stability (details in Section C.1), the most compatible difficulty yields both higher performance and smoother training curves. Conversely, overly simple problems provide insufficient learning signals, while excessively difficult problems introduce noise that can precipitate training collapse. This highlights the necessity of RL within the "Zone of Proximal Development", maximizing gradient effectiveness by aligning task difficulty with the model's cognitive capacity.

6.2 Effectiveness of Bipolar Float Reward

We ablated BFR against Binary ($\{0, 1\}$) and standard Graded Float ($[0, 1]$) rewards on Qwen3-8B. As shown in Table 2 (Panel C), BFR achieves the highest final accuracy on all six benchmarks. Relative to Binary, BFR improves AIME24, AIME25, and HMMT25 by 0.9, 2.2, and 4.3 percentage points, respectively. Moreover, after one epoch, BFR matches or exceeds Binary's two-epoch results on five of the six benchmarks, showing that BFR can achieve comparable or better performance in fewer training epochs. Detailed training curves and checkpoint results are provided in Section C.2.

These results support combining partial-correctness feedback with an additional incentive for complete solutions. As shown in Section 4.2.2, BFR yields the same standardized advantages as $S + \mathbf{1}[S = 1]$ under group normalization. Its effect is therefore governed by the relative reward structure: the bonus increases the separation between fully correct and imperfect answers, while graded scores preserve distinctions among imperfect answers. In groups containing fully correct answers and multiple levels of partial correctness, this mapping redistributes relative advantages. The benchmark and checkpoint comparisons provide empirical evidence for the effectiveness of this reward design in the evaluated setting.

Comparison with Existing Baselines. To further contextualize our results, we compare UltraLogic with the programmatic reasoning framework LogicPro, with both methods using Qwen3-8B, binary rewards, 10,000 training examples per setting, and two training epochs. As summarized in Table 2 (Panel A), UltraLogic generally outperforms this baseline across general reasoning benchmarks. A comprehensive discussion regarding the methodological differences with other recent works is provided in Section G. Notably, training on LogicPro yields almost no improvement over the base model because as model capabilities advance, the required difficulty for effective reinforcement learning escalates; consequently, static past data becomes overly simplistic, rendering training obsolete and ineffective. In contrast, UltraLogic continuously evolves task difficulty through iterative adaptation, ensuring long-term viability.

6.3 Additional Empirical Observations

Beyond the primary ablation studies, we identify two key empirical findings regarding the stability of the Reinforcement Learning (RL) process:

Data Quality Sensitivity. In our early GRPO baseline experiments with standard binary rewards, training collapse occurred when only 1–3 of the 50 task types contained errors in their solution logic or templates. This observation highlights the sensitivity of binary-reward training to data quality and the importance of the validation gate described in Section 3.4.3 for stable training.

Architectural Choice. Initial Mixture-of-Experts (MoE) trials frequently diverged during GRPO. To ensure stable gradients and smooth convergence, we adopted Dense models (e.g., Qwen3-8B and 14B), which handled UltraLogic's complex reasoning signals significantly more robustly.

7 Conclusion

We present the UltraLogic data framework and the Bipolar Float Reward (BFR) mechanism. UltraLogic employs an automated pipeline—combining seed tasks with programmatic expansion to enable large-scale synthesis of diverse, verifiable, and difficulty-stratified training datasets. Our experiments identify a difficulty matching phenomenon: training efficiency is maximized when task difficulty aligns with the model's zone of proximal development. Furthermore, feedback-driven task refinement and adaptive calibration position UltraLogic as a data-centric framework for RSI, providing a reusable mechanism for refreshing verifiable training tasks as model capabilities evolve. The BFR mechanism addresses the inherent information sparsity and the Non-negative reward trap found in traditional reward structures. This approach enhances the precision of reward signals, thereby improving both training accuracy and optimization efficiency.

AI USE STATEMENT

We used large language models to assist with task-template construction, bilingual template expansion, and the generation of task-specific input and solution code. LLMs also supported feedback and revision in the data-validation and difficulty-calibration workflows described in the paper. Problem instances and reference answers were produced by the task-specific generators and solvers. During manuscript preparation and revision, AI tools assisted with literature retrieval and comparison, translation and editing, checking the mathematical exposition of the reward mechanism, interpreting the reported results, code and template inspection, and LaTeX formatting. Human verification of task logic, templates, and reward configurations is described in the appendix on human annotation. The authors take responsibility for the final manuscript, methods, and reported results.

REPRODUCIBILITY STATEMENT

To make our results reproducible, we release an anonymized archive of the UltraLogic toolkit as supplementary material, comprising the metadata-driven data-synthesis pipeline (the entry-point driver plus one standalone generator per task, for all 100 tasks), all task metadata, every Bipolar Float Reward (BFR) verifier, and a small set of worked example samples per task (bilingual, Chinese–English) that illustrate the output format. The archive ships the synthesis *code* and these worked examples rather than the full training corpus: the complete datasets are regenerated by running the released pipeline, whose per-task sample count is a single configuration parameter. The archive is dual-licensed (Apache-2.0 for code, CC BY 4.0 for data) and documents all third-party dependencies with verified versions together with the tested Python version. The synthesis procedure—the task schema, the template/input/reference-solution definitions, the ten-level difficulty scale (Levels 1–10), and the bilingual generation protocol—is specified in Section 3, and in particular in the data-synthesis pipeline of Section 3.4, with the complete data taxonomy in Appendix B and every prompt used for task mining, template generalization, and input/solution-code generation in Appendix D; the released entry-point script regenerates the full bilingual sample set end-to-end from the task metadata. The full task inventory—per-task domain, difficulty source, and verifier routing for all released tasks—is tabulated in Appendix B and in the accompanying `TASK_SOURCES.md`, while the full training corpus used in our experiments (50 tasks, 10,000 samples) is *not* shipped in the archive but is regenerated by running the released synthesis code with the corresponding sample-count setting, as documented in Section 5.2. The BFR scoring function and its bipolar $[-1, 0) \cup \{1\}$ range are defined in Section 4; the exact scoring code for every common and task-specialized verifier ships in the archive, a complete worked verifier example is given in Appendix B.3, and the mathematical analysis of the reward-scaling factor is provided in Appendix F. Scoring is deterministic given a reference and a candidate answer, and each task provides a worked example question and reference answer (the `question_example` / `answer_example` metadata fields) for verification. All experimental hyperparameters, model versions, decoding settings, and the compute environment are documented in Section 5.1 (Training Configuration and Benchmarks) and Appendix C (Detailed Experiment Results). Together, these materials allow the datasets to be regenerated following the same documented protocol, the reward signal to be reproduced exactly, and the reported experiments to be independently replicated.

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.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang,

Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Yuyang Ding, Xinyu Shi, Xiaobo Liang, Juntao Li, Zhaopeng Tu, Qiaoming Zhu, and Min Zhang. Unleashing LLM reasoning capability via scalable question synthesis from scratch. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13414–13438, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.658. URL https://aclanthology.org/2025.acl-long.658/.

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. *NeurIPS*, 2021.

Jin Jiang, Yuchen Yan, Yang Liu, Jianing Wang, Shuai Peng, Xunliang Cai, Yixin Cao, Mengdi Zhang, and Liangcai Gao. LogicPro: Improving complex logical reasoning via program-guided learning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26200–26218, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1270. URL https://aclanthology.org/2025.acl-long.1270/.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V Le, and Orhan Firat. BIG-bench extra hard. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26473–26501, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1285. URL https://aclanthology.org/2025.acl-long.1285/.

Wai-Chung Kwan, Joshua Ong Jun Leang, Pavlos Vougiouklis, Jeff Z Pan, Marco Valentino, and Pasquale Minervini. Opensir: Open-ended self-improving reasoner. *arXiv preprint arXiv:2511.00602*, 2025.

Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, Yingying Zhang, Fei Yin, Jiahua Dong, Zhiwei Li, Bao-Long Bi, Ling-Rui Mei, Junfeng Fang, Xiao Liang, Zhijiang Guo, Le Song, and Cheng-Lin Liu. From system 1 to system 2: A survey of reasoning large language models, 2025a. URL https://arxiv.org/abs/2502.17419.

Ziheng Li, Zexu Sun, Jinman Zhao, Erxue Min, Yongcheng Zeng, Hui Wu, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Xu Chen, and Zhi-Hong Deng. Staying in the sweet spot: Responsive reasoning

evolution via capability-adaptive hint scaffolding, 2025b. URL https://arxiv.org/abs/2509.06923.

Mengqi Liao, Xiangyu Xi, Chen Ruinian, Jia Leng, Yangen Hu, Ke Zeng, Shuai Liu, and Huaiyu Wan. Enhancing efficiency and exploration in reinforcement learning for LLMs. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1451–1463, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.75. URL https://aclanthology.org/2025.emnlp-main.75/.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Junteng Liu, Yuanxiang Fan, Zhuo Jiang, Han Ding, Yongyi Hu, Chi Zhang, Yiqi Shi, Shitong Weng, Aili Chen, Shiqi Chen, Yunan Huang, Mozhi Zhang, Pengyu Zhao, Junjie Yan, and Junxian He. Synlogic: Synthesizing verifiable reasoning data at scale for learning logical reasoning and beyond, 2025a. URL https://arxiv.org/abs/2505.19641.

Xianyang Liu, Yilin Liu, Shuai Wang, Hao Cheng, Andrew Estornell, Yuzhi Zhao, and Jiaheng Wei. Agenticmath: Enhancing llm reasoning via agentic-based math data generation, 2025b. URL https://arxiv.org/abs/2510.19361.

Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. MathGenie: Generating synthetic data with question back-translation for enhancing mathematical reasoning of LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2732–2747, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.151. URL https://aclanthology.org/2024.acl-long.151/.

Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. Let's reward step by step: Step-level reward model as the navigators for reasoning, 2023. URL https://arxiv.org/abs/2310.10080.

Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. LILA: A unified benchmark for mathematical reasoning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5807–5832, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.392. URL https://aclanthology.org/2022.emnlp-main.392/.

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.

Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkACllQgA-.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

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. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning Gym: Reasoning environments for reinforcement learning with verifiable rewards. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-1913. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/528d56195a2c77c808494c86fa7c77ad-Paper-Datasets_and_Benchmarks_Track.pdf.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Xukai Wang, Xuanbo Liu, Mingrui Chen, Haitian Zhong, Xuanlin Yang, Bohan Zeng, Jinbo Hu, Hao Liang, Junbo Niu, Xuchen Li, et al. Morphobench: A benchmark with difficulty adaptive to model reasoning. *arXiv preprint arXiv:2510.14265*, 2025.

Junru Wu, Tianhao Shen, Linxi Su, and Deyi Xiong. C²RBench: A Chinese complex reasoning benchmark for large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 21031–21050, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1083. URL https://aclanthology.org/2025.findings-acl.1083/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Zhangyue Yin, Qiushi Sun, Zhiyuan Zeng, Qinyuan Cheng, Xipeng Qiu, and Xuan-Jing Huang. Dynamic and generalizable process reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4203–4233, 2025.

Shaoxiong Zhan, Yanlin Lai, Ziyu Lu, Dahua Lin, Ziqing Yang, and Fei Tan. MathSmith: Towards extremely hard mathematical reasoning by forging synthetic problems with a reinforced policy. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(41):34602–34610, 2026. doi: 10.1609/aaai.v40i41.40760.

Kaiyan Zhang, Jiayuan Zhang, Haoxin Li, Xuekai Zhu, Ermo Hua, Xingtai Lv, Ning Ding, Biqing Qi, and Bowen Zhou. OpenPRM: Building open-domain process-based reward models with preference trees. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=fGIqGfmgkW.

Xinyu Zhu, Yihao Feng, Yanchao Sun, Xianzhi Du, Pingzhi Li, Olli Saarikivi, Yun Zhu, and Yu Meng. Chimera: Compact synthetic data for generalizable llm reasoning. *arXiv preprint arXiv:2603.00889*, 2026.

A LIMITATIONS

Dependence on Human Verification. ULTRALOGIC combines task-level quality control with large-scale data production. Human verification covers task logic, template expression, and reward configuration, while finalized task definitions are used for programmatic data generation. As a core contribution of this work, the public dataset also undergoes human review before release to strengthen checks on problem statements and answer consistency. This workflow focuses human review on reusable task definitions and release quality control, while supporting large-scale production of training samples through an automated pipeline.

Static Reward Mapping. BFR currently uses a common linear graded mapping across tasks and a fixed full-correctness bonus. This design makes the reward mechanism simple and easy to apply to a variety of verifiable tasks. Future work can further combine reward mappings with task structure and training stages to explore adaptive bonuses and nonlinear correctness mappings.

B DETAILED DATA SETTINGS

This appendix provides comprehensive details regarding the ULTRALOGIC dataset configuration.

B.1 THREE-DIMENSIONAL ORTHOGONAL CATEGORIES

The complete taxonomy, including all sub-categories and their respective descriptions within the ULTRALOGIC Task Repository, is presented in Table 3.

Table 3: The three-dimensional orthogonal classification system of the ULTRALOGIC Task Repository. This system guides task creation and screening to ensure coverage of a broad spectrum of reasoning challenges, where each task type is uniquely situated and synthesized by the intersection of sub-categories selected from three independent dimensions.

Categories	Sub-categories	Description
Task Domain	Symbolic Manipulation	Manipulate letters and special symbols, such as symbol or regular matching.
	Numerical Manipulation	Manipulate numbers, such as numerical conversion or numerical decryption.
	Textual Manipulation	Manipulate word or text, such as password puzzles.
	Object Manipulation (Real-World)	Tasks integrated with real-world scenarios, such as case investigation.
	Planning & Scheduling	Tasks involving the planning and organization of complex tasks, such as elevator scheduling, cargo transportation.
	Classic Games	Classic games such as Sudoku, 24-point, or Monopoly.
	Spatial: Geometry	Describe complex geometric or positional relationships using plain text.
	Spatial: Pathfinding	Pathfinding problem, finding the optimal path.
	Spatial: Visual-to-Text	Numbers, letters or symbols are used to represent graphical or geometric positional relationships, such as a two-dimensional map.
	Cross-Topic	Various combinations of the aforementioned tasks.
	Others	-
Core Ability	Constraint Satisfaction	The problem will contain numerous conditions and constraint rules. It is essential to pay attention to ensuring that solutions do not contradict the conditions or violate the rules

Categories	Sub-categories	Description
	Algorithmic Thinking	Solution requires the application of algorithmic concepts such as dynamic programming, the knapsack problem or depth-first search.
	Info Extraction & Integration	The problem may contain a significant amount of irrelevant information, which requires identifying the effective information relevant to solving the problem.
	Item Connection & Mapping	Identify the mapping relationships between multiple items, such as the relationships between spatial connections and cipher mappings
	Instruction Following	The problem may require performing multiple operations on an item, such as rotating a Rubik's cube multiple times. When solving the problem, it is essential to follow to the specified sequence of operations.
	Others	-
Difficulty Source	Complex Rules	The difficulty of the problem lies in the numerous and complex rules involved, such as in complex board games.
	Complex Conditions	The problem presents multiple conditional pathways for selection, requiring precise comprehension of the conditions to enable correct choices at each step and ultimately solve the problem, as exemplified in game-level progression scenarios
	Large Search Space	When solving the problem requires identifying the optimal solution, it is necessary to explore numerous approaches to achieve this, such as the maze problem.
	Tedious Solution Steps	The problem may contain lengthy procedural steps or complex logical structures, requiring tedious solution steps to resolve.
	Computational Complexity	Problem-solving involves complex mathematical computations
	Intrinsic LLM Weaknesses	Some weaknesses inherent to large models, such as errors in numeric and alphabetic characters caused by token encoding issues
	Others	-

B.2 TASK CATEGORY CASES

Table 4: Representative task examples across different domains, core abilities, and difficulty sources in the ULTRALOGIC framework.

Task examples	Task Domain Core Ability Difficulty Source
Observe the chat records of a certain study group. There are 7 members who sent messages one after another, and the content is: Wright: There are exactly 6 people telling the truth. Turner: There are at least 6 people telling the lie. Ross: There are at least 4 people telling the lie. Torres: There are at least 3 people telling the lie. Harris: There are exactly 3 people telling the lie. Brooks: There are at least 2 people telling the lie. Garcia: There are at least 1 people telling the truth. Question: Among these 7 members who sent messages, who are the ones telling the truth?	Object Manipulation (Real-World) Constraint Satisfaction Large Search Space
In an ancient and mysterious maze, an adventurer wants to find the exit. The maze is a two-dimensional layout, described as follows: □□□■□ ■□□■■ ■■□■■ ■■□□■ ■□■□□ The entrance of the maze is in the top-left corner, and the exit is in the bottom-right corner. There are some exploration paths, where each exploration action represents moving 1 unit in the specified direction. If the corresponding direction is blocked by a maze wall, this step is skipped. The paths are as follows: A:right, down, right, down, right, down, right B:right, down, right, down, down, right, down, down, right C:right, down, down, right, down, right D:right, down, down, right, down, down, down, right, down, right Please list all the path IDs that allow the adventurer to get from the start point to the endpoint and exit the maze	Spatial: Pathfinding Instruction Following Tedious Solution Steps
The Royal Antiquities Museum has enlisted a legendary cipher expert to reassign symbolic seals for its treasured artifacts. The original seals consist of interleaved uppercase letters, lowercase letters, and positive/negative integers. The specific seal to be decoded is: 6889SnKBUNXl. Decoding rules are as follows: 1. Eliminate all lowercase letters. 2. Retain uppercase letters in their original sequence. 3. Sum all integers (defined as consecutive digits) to form the numerical suffix of the new seal. What is the length of the newly assigned seal after applying these rules?	Symbolic Manipulation Instruction Following Intrinsic LLM Weaknesses
In the art summer camp, the student Lily is creating an abstract painting. The canvas is a grid composed of small squares, and each square must be painted with a specific color. Lily's rule is that each time she can only use one color to paint a complete rectangular area, and the same color cannot be reused. Now given the target color distribution of each position on the canvas: row 1: color: 1, 1, 1 row 2: color: 1, 1, 3 row 3: color: 1, 2, 1 can Lily complete the painting according to the rules?	Spatial: Geometry Constraint Satisfaction Large Search Space

Task examples	Task Domain Core Ability Difficulty Source
In a particular location, scorching weather triggered a series of cascading effects. Determine the valid causal chain based on the following conditions. **Events**: B17: High-speed closure A1: Heavy rain C9: The driver failed to avoid the space station data **Conditions**: (1) If there is no "heavy rain", "highway closure" is unlikely to occur. (2) Typically, [highway closures] will occur shortly after [heavy rain]. (3) Without "highway closure", it is unlikely for "drivers to fail to avoid" accidents. (4) Typically, after [high-speed closure], [driver's failure to avoid] will occur shortly thereafter. **Question**: What is the established causal chain?	Planning & Scheduling Info Extraction & Integration Complex Conditions

B.3 A COMPLETE TASK CASE

Table 5: Complete Task Case: True and False Game (Full Metadata)

Field	Content
Task ID	`true_and_false_game`
Problem Case	Observe the chat records of a certain study group. There are 7 members who sent messages one after another, and the content is: Wright: There are exactly 6 people telling the truth. Turner: There are at least 6 people telling the lie. Ross: There are at least 4 people telling the lie. Torres: There are at least 3 people telling the lie. Harris: There are exactly 3 people telling the lie. Brooks: There are at least 2 people telling the lie. Garcia: There are at least 1 people telling the truth. Question: Among these 7 members who sent messages, who are the ones telling the truth?
Categories	Object Manipulation (Real-World) & Constraint Satisfaction & Large Search Space
En Templates	1. "At the discussion meeting of the new product project team of a certain company, the participants took turns expressing their views. The specific contents of the speeches are as follows: [slot_2]. It is known that there are a total of [slot_1] participants in this discussion. Question: Which among the [slot_1] participants are telling the truth? Please list the names separated by commas, and in the order of their speeches, for example: Amelia, Bella, Taylor." 2. "In a class-themed group discussion, [slot_1] members expressed their opinions in sequence, with the specific content being: [slot_2]. So who among these [slot_1] members is telling the truth? Please separate the names with commas and list them in the order they spoke, for example: Charlie, Lily, Alexander." 3. "During the weekend, members of the photography enthusiasts group gathered together to share their shooting experiences, and each person expressed their own insights on photography one by one. The specific content is as follows: [slot_2]. It is known that a total of [slot_1] people participated in this exchange. Question: Among these [slot_1] people, who are the ones telling the truth? Please list the names separated by commas in the order they spoke, for example: Lee, Tom, Jerry" 4. "Observe the chat records of a certain study group. There are [slot_1] members who sent messages one after another, and the content is: [slot_2]. Question: Among these [slot_1] members who sent messages, who are the ones telling the truth? Please list the names separated by commas in the order of speaking, for example: William, James, John" 5. "In the debate competition organized by the school, there are [slot_1] debaters on each side who take turns presenting their viewpoints. The specific debate speeches are as follows: [slot_2]. Question: Among these [slot_1] debaters, who told the truth? Please list the names separated by a comma and in the order of speaking, for example: Michael, David, Liam." 6. "At the family gathering, family members discussed weekend travel plans. Everyone took turns sharing their thoughts, specifically: [slot_2]. It is known that there are [slot_1] family members participating in the discussion. The question is: Who among these [slot_1] people is telling the truth? Please list the names in the order they spoke, separated by commas, e.g., Noah, Oliver, Mary." 7. "The community volunteer group held a meeting to plan next weekend's activity. The group members expressed their opinions on the activity plan one by one, and the specific speeches are as follows: [slot_2]. It is known that there are [slot_1] volunteers participating in this meeting. Question: Among these [slot_1] people, who are telling the truth? Please list their names in the speaking order separated by commas, for example: John, Jennifer, Jessica" 8. "Members of a certain book club gathered to discuss a popular novel. Each person shared their understanding of the novel's ending in turn, as follows: [slot_2]. It is known that there are [slot_1] participants in this book club. The question is: among these [slot_1] members, who is telling the truth? Please list the names separated by commas and in the order they spoke, for example: Emma, Harry, Sophia"

Field	**Content** (Continued)
	9. "At the company's quarterly goal kick-off meeting, team members successively presented their feasibility analyses for achieving the goals. The specific speeches were: [slot_2]. It is known that there are a total of [slot_1] project team members attending. Question: Among these [slot_1] people, who are telling the truth? Please list the names separated by commas and in the order of speaking, for example: Aaron, Michael, Mary"
	10. "On a popular forum, there are [slot_1] users discussing 'whether new energy vehicles should be promoted'. The original poster initiated the topic, and then other users responded one by one to express their views. The specific content is as follows: [slot_2]. Question: Among the [slot_1] users participating in the discussion, who is telling the truth? Please list the names separated by commas in the order of speaking, for example: David, Jennifer, Abel"

Input Code

```
def input(difficulty: int = 1, language: str = 'en'):
  import random, re; common_surnames = []
  if language == 'en':
     common_surnames = ['Smith', 'Johnson', 'Williams', 'Brown', 'Jones', 'Garcia', 'Miller',
'Davis', 'Rodriguez', 'Martinez', 'Hernandez', 'Lopez', 'Gonzalez', 'Wilson', 'Anderson',
'Thomas', 'Taylor', 'Moore', 'Jackson', 'Martin', 'Lee', 'Perez', 'Thompson', 'White',
'Harris', 'Sanchez', 'Clark', 'Ramirez', 'Lewis', 'Robinson', 'Walker', 'Young', 'Allen',
'King', 'Wright', 'Scott', 'Torres', 'Nguyen', 'Hill', 'Flores', 'Green', 'Adams', 'Nelson',
'Baker', 'Hall', 'Rivera', 'Campbell', 'Mitchell', 'Carter', 'Roberts', 'Gomez', 'Phillips',
'Evans', 'Turner', 'Diaz', 'Parker', 'Cruz', 'Edwards', 'Collins', 'Reyes', 'Stewart', 'Morris',
'Morales', 'Murphy', 'Cook', 'Rogers', 'Gutierrez', 'Ortiz', 'Morgan', 'Cooper', 'Peterson',
'Bailey', 'Reed', 'Kelly', 'Howard', 'Ramos', 'Kim', 'Cox', 'Ward', 'Richardson', 'Watson',
'Brooks', 'Chavez', 'Wood', 'James', 'Bennett', 'Gray', 'Mendoza', 'Ruiz', 'Hughes', 'Price',
'Alvarez', 'Castillo', 'Sanders', 'Patel', 'Myers', 'Long', 'Ross', 'Foster', 'Jimenez']
  def generate_unique_solution_problem():
    while True:
      names, statements = generate_random_truth_falsehood_problem(difficulty)
      solutions = solve_truth_statements(names, statements)
      if len(solutions) == 1:
        true_speakers = [name for name, truth in solutions[0].items() if truth == 'Truth']
        return names, statements, solutions[0], len(true_speakers), true_speakers
  def generate_random_truth_falsehood_problem(difficulty):
   difficulty_params = {1: {"people_count": 7}, 2: {"people_count": 9}, 3: {"people_count": 11},
4: {"people_count": 12}, 5: {"people_count": 13}, 6: {"people_count": 14}, 7: {"people_count":
15}, 8: {"people_count": 16}, 9: {"people_count": 18}, 10: {"people_count": 20}}
    count = difficulty_params[difficulty]["people_count"]
    surnames = random.sample(common_surnames, count); names = [f"{s}" for s in surnames]
    statements = []; used = set()
    while len(statements) < count:
      mode = random.choices(["at least", "at most", "exactly"], [4, 1, 3])[0]
      target = random.choice(["truth", "lie"]); num = random.randint(1, count)
      stmt = f"There are {mode} {num} people telling the {target}."
      if stmt not in used: used.add(stmt); statements.append(stmt)
    return names, statements
  def solve_truth_statements(names, stmts):
    n = len(names); result = []; pattern = r"There are (at least|at most|exactly) (\d+) people
telling the (truth|lie)"
    from itertools import product
    for possibility in product([True, False], repeat=n):
      true_count = sum(possibility); false_count = n - true_count; checks = []
      for stmt in stmts:
        match = re.match(pattern, stmt); m_str, num, k_str = match.groups()
        curr = true_count if k_str == 'truth' else false_count; num = int(num)
        if m_str == 'at least': checks.append(curr >= num)
        elif m_str == 'at most': checks.append(curr <= num)
        else: checks.append(curr == num)
      valid = True
      for i in range(n):
        if possibility[i] != checks[i]: valid = False; break
     if valid: result.append({names[j]: 'Truth' if possibility[j] else 'Lie' for j in range(n)})
    return result
  names, stmts, assign, t_cnt, t_spks = generate_unique_solution_problem()
  process = " ".join([f"{n}: {s}" for n, s in zip(names, stmts)])
```

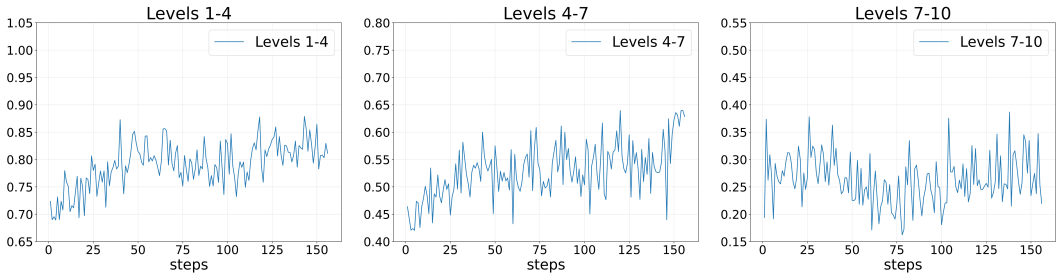

Figure 3: The critic/score/mean metrics of Qwen3-8B during the GRPO process on the different training sets of varying difficulty levels, including Easy, Medium, and Hard.

Field	**Content** (Continued)
	`return ', '.join(t_spks), [len(names), process]`
Solution Code	`def solution(params, language: str = 'en'): return params`
BFR Type	F1-score

BFR Code	`def f1_score_reward(model_answer, ground_truth):`
	`  gt_set = set(ground_truth.split(', '))`
	`  model_set = set(model_answer.split(', '))`
	`  if not gt_set or not model_set: return -1.0`
	`  inter = len(gt_set.intersection(model_set))`
	`  p = inter / len(model_set); r = inter / len(gt_set)`
	`  f1 = 2 * (p * r) / (p + r) if (p + r) > 0 else 0`
	`  return 1.0 if f1 == 1.0 else f1 - 1.0`

C Detailed Experiment Results

This appendix details the experiment results. Based on the verl[5] framework, we conducted GRPO training. The training of Qwen3-8B took approximately 120 hours for 2 epochs, while the training of Qwen3-14B consumed around 300 hours for the same number of epochs[6].

C.1 Difficulty Matching Ablation Study

Figures 3 and 4 illustrate the critic/score/mean metrics during the GRPO process over 2 training epochs for Qwen3-8B and Qwen3-14B on the different training datasets with varying difficulty levels, including Easy, Medium, and Hard.

As shown in Figure 3, Qwen3-8B exhibits the most consistent reward growth on Levels 1–4 data. While Levels 4–7 shows initial gains followed by late-stage oscillations, the Levels 7–10 subset is characterized by severe instability and lack of overall improvement, indicating a mismatch with the model's capacity.

For Qwen3-14B (Figure 4), the Levels 4–7 subset yields the optimal and steadiest upward trajectory. In contrast, the Levels 1–4 data leads to premature convergence with minimal gains, while Levels 7–10 introduces noticeable oscillations, particularly in the second epoch.

Based on the experimental phenomena observed above, it can be generally concluded that when the model is trained on relatively simple data, its performance can significantly improve. However, overly simple data can lead to rapid convergence, resulting in insignificant performance enhancement. Conversely, when the model is trained on more difficult data, its performance exhibits significant fluctuations, which may even lead to training collapse. Therefore, in GRPO training, selecting the most suitable data for the model's adaptability to difficulty is particularly crucial, and this point is highly consistent with the performance on the evaluation set shown in Table 6.

[5] https://github.com/volcengine/verl

[6] https://huggingface.co/collections/Qwen/qwen3

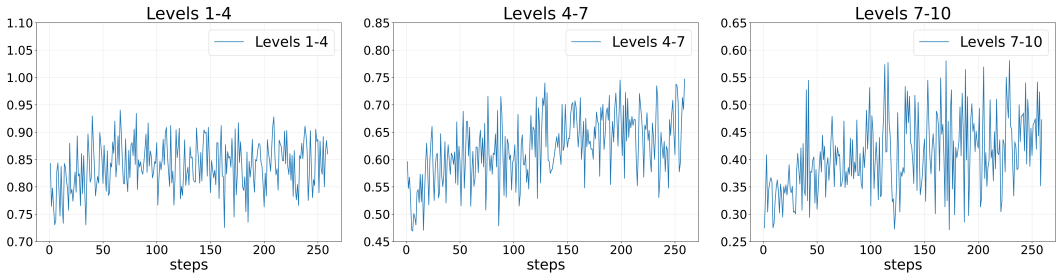

Figure 4: The critic/score/mean metrics of Qwen3-14B during the GRPO process on the different training sets of varying difficulty levels, including Easy, Medium, and Hard.

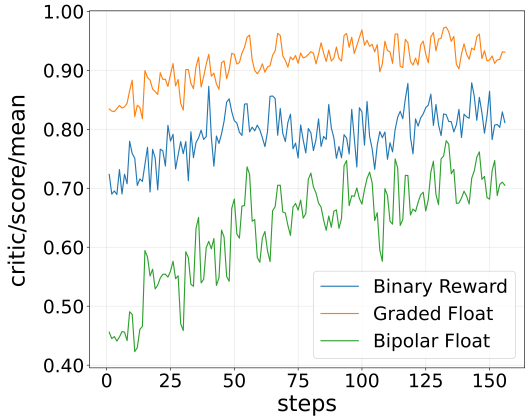

Figure 5: The critic/score/mean metrics of Qwen3-8B during the GRPO process using different reward mechanisms.

Table 6: Ablation study on model scale and task difficulty.

Model	AIME24	AIME25	HMMT25	BBH	BBEH	ARC-AGI
Qwen3-8B	75.2	66.1	47.1	88.2	29.2	4.0
Levels 1–4_1epoch	78.3	67.5	50.3	89.3	30.4	4.3
Levels 1–4_2epoch	**81.7**	**69.1**	**52.3**	**90.2**	**31.1**	**4.6**
Levels 4–7_1epoch	77.5	66.7	50.1	88.5	30.2	4.1
Levels 4–7_2epoch	77.6	67.6	50.9	88.7	29.6	4.2
Levels 7–10_1epoch	75.8	64.5	48.4	88.0	28.8	3.9
Levels 7–10_2epoch	76.5	65.1	50.0	88.3	29.0	4.0
Qwen3-14B	76.7	70.3	54.3	89.6	32.2	6.3
Levels 1–4_1epoch	78.3	71.6	56.9	90.9	33.6	6.3
Levels 1–4_2epoch	78.8	72.5	58.6	90.9	33.7	6.1
Levels 4–7_1epoch	81.6	74.2	61.5	91.4	35.6	7.6
Levels 4–7_2epoch	**82.4**	**75.8**	**63.1**	**92.1**	**36.7**	**7.9**
Levels 7–10_1epoch	80.0	72.9	59.5	90.8	34.1	6.9
Levels 7–10_2epoch	80.8	73.8	59.9	90.9	34.7	6.8

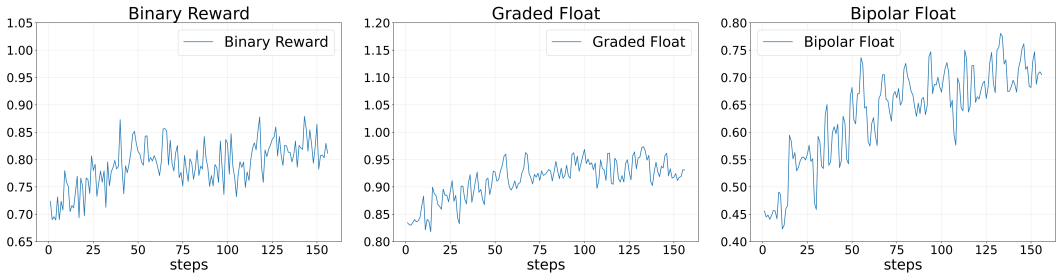

Figure 6: The critic/score/mean metrics of Qwen3-8B during the GRPO process by using different reward mechanisms, including Binary Reward, Graded Float Reward, and Bipolar Float Reward.

C.2 Bipolar Float Reward Ablation Study

Table 7 further shows the earlier performance gains achieved by BFR. After one epoch, BFR reaches 81.8, 69.2, and 54.9 on AIME24, AIME25, and HMMT25, respectively, exceeding Binary's two-epoch results of 81.7, 69.1, and 52.3. It also matches Binary's two-epoch results on BBEH and ARC-AGI (31.1 and 4.6). These checkpoint comparisons show that BFR achieves comparable or better performance in fewer training epochs across multiple reasoning benchmarks. The corresponding training trajectories are shown in Figures 5 and 6.

Table 7: Ablation study on reward mechanisms. Comparison of Binary, Graded Float, and Bipolar Float rewards.

Reward	AIME24	AIME25	HMMT25	BBH	BBEH	ARC-AGI
Qwen3-8B	75.2	66.1	47.1	88.2	29.2	4.0
Binary Reward-1epoch	78.3	67.5	50.3	89.3	30.4	4.3
Binary Reward-2epoch	81.7	69.1	52.3	90.2	31.1	4.6
Graded Float-1epoch	76.4	66.2	53.6	88.9	30.2	4.2
Graded Float-2epoch	76.9	66.3	53.0	90.4	31.0	4.3
Bipolar Float-1epoch	81.8	69.2	54.9	89.7	31.1	4.6
Bipolar Float-2epoch	**82.6**	**71.3**	**56.6**	**91.1**	**32.5**	**4.7**

D PROMPTS

This appendix presents the specific prompt templates used in the ULTRALOGIC framework.

D.1 THE PROMPT OF TASK MINING AND TEMPLATE CREATION

This prompt acts as an initial gatekeeper and architect, determining whether a reasoning problem can be programmatically generated and, if so, transforming it into a natural language logical reasoning template with placeholders (slots).

Task Mining and Template Creation Prompt

You are an intelligent assistant for task mining and question template generation. Your task is to judge whether a given reasoning problem can be solved by code and whether its input can be programmatically generated. If feasible, rewrite it into a daily life logical reasoning template.
Requirements:

1. Feasibility Check: If the problem cannot be solved by code or the inputs cannot be generated, reply "Unable to generate a code-solvable template."
2. Scenario Rewriting: Convert the problem into a natural daily life reasoning question by introducing backgrounds, names, and locations.
3. Slot Replacement: Replace specific input parameters with [Slot 1], [Slot 2], etc. Ensure the required answer logic remains identical to the original problem.
4. Technical Constraint: Do not include any code snippets or technical terms (e.g., arrays, hash maps, binary trees).
5. Output Format: If feasible, output the result in JSON format with three keys: `rewriting_plan`, `new_question_template`, and `slot_description`.

Example:
`[Reasoning Problem]` Walking in the park, if you follow these routes..
`[Output]` Unable to generate a code-solvable template.
or

```
{
    "rewriting_plan": "xxx",
    "new_question_template": "xxx",
    "slot_description": "xxx"
}
```

Input:
`[Reasoning Problem]` {reasoning_problem}
`[Output]`

D.2 THE PROMPT OF QUESTION TEMPLATE GENERALIZATION

This prompt is used to expand the diversity of the dataset by rewriting a single logical template into ten different scenarios while maintaining the underlying logic and slot structure.

Question Template Generalization Prompt

You are a rewriting generalizer. Your task is to rewrite a given question template into ten different versions.

1. Randomly introduce backgrounds, change scenarios, names, and word orders to ensure diversity.
2. Maintain the same question type and core logic; do not change the intent of the test.
3. Ensure that the slots (e.g., [Slot 1], [Slot 2]) remain unchanged in the new templates.
4. The similarity between the ten generated templates must be low; do not simply swap names.
5. Output in JSONArray format with two keys: `rewriting_plan`, `new_question_template`.

6. Maintain the original language of the template.

Example:
[Original Template] Walking in the park, if you follow these routes... [Slot 1]
[Output JSONArray] (Ten variations including maze exploration, robot navigation, etc.)

```
[
    {
        "rewriting_plan": "xxx",
         "new_question_template": "xxx"
    },
    ...
]
```

Task:
Rewrite the following template into ten variations:
[Original Template] {question_template}
[Output JSONArray]

D.3 The Prompt of Input Code Generation

The following prompt is used to guide the model to generate Python scripts. these scripts are responsible for producing randomized, difficulty-controlled input text that fits into the predefined question templates.

Input Code Generation Prompt

You are a code generator. Please generate input generation code based on a given original programming problem, logic reasoning template, slot description, and input examples. Requirements are as follows:

1. The function of the code: According to the parameters in the original problem, generate natural language text for various slots based on the examples. This ensures the template forms a complete, logical, and easy-to-understand reasoning problem.
2. Input parameter: `difficulty` (1–10). It controls the complexity so that different levels of difficulty are covered and the span between levels is significant. In the input code, there will be a mapping of difficulty mapping parameters, for example:

```
difficulty_params = {
    1: {"num": 3}, 2: {"num": 5},
    3: {"num": 8}, 4: {"num": 10},
    5: {"num": 12}, 6: {"num": 15},
    7: {"num": 18}, 8: {"num": 22},
    9: {"num": 26}, 10: {"num": 30}
}
```

3. Input parameter: `language` (en, zh). It controls the language of the slot-filling text, whether it is in Chinese or English.
4. The code should be concise, clear, and handle edge cases.
5. The generated text must be randomized. Do not output raw data formats like arrays, lists, or JSON; convert them into natural language descriptions.
6. The return value should contain two elements: the raw parameter values (as in the example) and a list of strings for the slots.
7. Ensure generated values are logically consistent (e.g., no "893:00" in time contexts) and the distribution of answers is as uniform as possible.

Example:
[Question Template] There are [Slot 1] members who sent messages one after another...
[Slot Desc] [Slot 1] is number of people...
[Input Code] (Generated Python function `input(difficulty, language='en')`)
Task:
Generate the code for the following:
[Template] {question_template}

`[Slot Desc]` {slot_description}
`[Input Code]`

D.4 The Prompt of Solution code Generation

This prompt guides the model to generate the core solver script. This Python code takes the randomized slot values as input and computes the definitive ground-truth answer according to the problem's logic.

Solution Code Generation Prompt

You are a Python expert. Your task is to write a solver function for a logical reasoning problem based on its original programming logic and template.
Requirements:
1. Core Function: `def solution(input_params, language)` that takes raw parameter values returned by input code as input_parameters and returns the correct answer.
2. Logic Accuracy: The code must strictly implement the reasoning logic described in the `[Programming Problem]`.
3. Code Style: Ensure the code is concise, efficient, and includes necessary comments for complex steps.
4. Output Format: The function should return a single value (string, int, or float) that can be directly compared with model outputs.

Example:
`[Question Template]` There are [Slot 1] members who sent messages one after another...
`[Slot Desc]` [Slot 1] is number of people...
`[Complete Question]` There are 5 members who sent messages one after another...
`[Input Parameters]` [5]
`[Solution Code]` (Generated Python function `solution(input_params, language='en')`)
Task:
Generate the Python `solution` function for this problem.
`[Question Template]` {question_template}
`[Slot Desc]` {slot_desc}
`[Complete Question]` {complete_question}
`[Input Parameters]` {input_parameters}
`[Solution Code]`

D.5 The Prompt of Dynamic Parameter Adjustment

This prompt is used to determine and refine the mapping between difficulty levels ($1 \sim 10$) and internal complexity parameters. It ensures that structural complexity scales appropriately and can be adjusted based on the actual success rate of models.

Dynamic Parameter Adjustment Prompt

You are a mathematical modeling assistant for logic puzzles. Your task is to define or adjust a mapping between a difficulty scale (1 to 10) and the structural parameters of a reasoning problem.
Instructions:
1. Identify core variables (e.g., N people, M steps).
2. Create or update the `difficulty_params` dictionary for all levels from 1 to 10.
3. Global Calibration: If measured success rates for anchor levels (e.g., 1, 5, and 10) are provided, evaluate the overall difficulty trend.
4. Monotonic Consistency: Ensure that parameters increase monotonically and linearly across the 1–10 range. If a middle level (e.g., Level 5) is adjusted, intermediate levels must be shifted accordingly to maintain a smooth gradient and avoid "difficulty inversion".

Example (Global Adjustment):
`[Question Template]` There are [Slot 1] members who sent messages one after another...

[Slot Desc] [Slot 1] is number of people...
[Current Mapping]

```
difficulty_params = {
    1: {"num": 3}, 2: {"num": 6},
    3: {"num": 9}, 4: {"num": 12},
    5: {"num": 15}, 6: {"num": 18},
    7: {"num": 21}, 8: {"num": 24},
    9: {"num": 27}, 10: {"num": 30}
}
```

[Feedback] L1:100%, L5:30% (Target 50%), L10:0%.
[Adjustment Logic] The model struggles as num exceeds 12. To maintain linear growth, we compress the parameters for levels 2–9.
[Updated Mapping]

```
difficulty_params = {
    1: {"num": 3}, 2: {"num": 5},
    3: {"num": 8}, 4: {"num": 10},
    5: {"num": 12}, 6: {"num": 15},
    7: {"num": 18}, 8: {"num": 22},
    9: {"num": 26}, 10: {"num": 30}
}
```

Task:
Suggest/Refine the complete 1–10 parameter mapping based on the following feedback:
[Question Template] {question_template}
[Slot Desc] {slot_desc}
[Current Mapping] {currect_mapping}
[Feedback] {feedback}
[Adjustment Logic]
[Updated Mapping]

E Details of Human Annotation

A total of 12 annotators participated in task logic verification, template refinement, and reward mechanism configuration. All were internal members of our research team and held a bachelor's degree or higher in a computer-related discipline. Human verification focuses on reusable task definitions: annotators check the consistency between problem constraints and input/solution logic, slot completeness and logical equivalence across templates, and the task-specific scoring rules, and revise identified issues. Verified task definitions support programmatic generation of large numbers of samples without requiring ground-truth answers to be manually constructed for every instance. The public dataset also undergoes human review before release to check problem statements and answer consistency. The detailed annotation guidelines are provided below.

Guidelines for Human Annotators

(1) Question Quality Check
- Ambiguity Analysis: Check for logic loopholes, multiple possible interpretations, missing/redundant information, unstated implicit assumptions, misleading statements, or vague phrasing (e.g., unclear temporal/spatial relationships).
- Judgment Criteria: A task is valid only if it has a unique reasonable answer, presents all necessary information for solving, and contains no presets that violate common sense.

(2) Category Labeling: Annotate each task according to the descriptions of the *Three-dimensional Orthogonal Classification System* (Task Domain, Core Ability, and Difficulty Source).

(3) Template Verification
- Ensure slots are compatible with the new templates and no critical information is lost.
- Verify that the 10 generated templates are visually distinct and that the core reasoning focal point has not been altered by rephrasing or scenario changes.

(4) Difficulty and Correctness Verification: Validate the task through multiple sampling of LLMs. Ensure that responses are complete and scoring is accurate, ruling out "noise" such as low success rates caused by API call failures.
(5) Input and Solution Code Verification
- Manual Correction: If full automation fails, annotators must manually intervene to ensure code correctness based on the principle of accurate parameter generation and solution logic.
- Difficulty Adjustment Principles:
  i. *Complexity Scaling*: Increase distractors (recommended ratio of 3:7 for core vs. distractor info), design multi-step chains (3–5 logical jumps), or hide key clues in secondary descriptions.
  ii. *Advanced Techniques*: Utilize nested structures (e.g., contradictory testimonies), psychological misdirection (exploiting cognitive biases), or domain-specific knowledge (e.g., cryptography).
  iii. *Optimization*: Maintain strict logical rigor and ensure all clues are presented fairly and are traceable.

(6) Bipolar Float Reward (BFR) Annotation
- Response Restructuring: Convert single answers into structured combinations that reflect the reasoning process.
- Scoring Configuration: Select from standard types: Accuracy, F1-Score, Similarity, or Absolute Difference.
- Robustness: The scoring code must handle minor output instabilities, such as redundant spaces or varied punctuation (bilingual compatibility).
- Determinism: Functions must yield identical results for the same input with no randomness. Returns must be a float in $[-1, 0) \cup \{1\}$.
- Difficulty Preservation: Modifications should focus on output format (e.g., "list all IDs" instead of "count the IDs") without changing the problem's underlying constraints.

F Mathematical Analysis of the Bipolar Float Reward Scaling Factor

In this section, we provide a mathematical derivation and sensitivity analysis to justify the specific scaling factor $c = 1.0$ (mapping partial rewards to $[-1, 0)$) utilized in our Bipolar Float Reward (BFR) mechanism within the GRPO framework.

F.1 Background and Notation

In GRPO, for a given prompt, we sample G responses. The reward for each response is r_i, and the advantage is obtained via Z-score standardization within the group:

$$\begin{aligned} \hat{A}_i &= \frac{r_i - \mu_r}{\sigma_r}, \quad \mu_r = \frac{1}{G}\sum_{j=1}^{G} r_j \\ \sigma_r &= \sqrt{\frac{1}{G}\sum_{j=1}^{G}(r_j - \mu_r)^2} \end{aligned} \tag{3}$$

Our BFR mapping scheme (assuming $S \in [0, 1]$ is the original raw correctness score) is defined with a scaling factor $c > 0$:

$$R_c(S) = \begin{cases} 1 & \text{if } S = 1 \\ -c \cdot (1 - S) & \text{if } S < 1 \end{cases} \tag{4}$$

The core question is: what value of c allows the advantages to most effectively distinguish between responses of varying quality?

F.2 Conservation of Advantage Variance

Theorem 1: For any G raw rewards $r_1, \ldots, r_G$ (provided they are not all identical), the sum of squared differences of the Z-score standardized advantages is a conserved quantity:

$$\sum_{i<j}(\hat{A}_i - \hat{A}_j)^2 = G^2 \tag{5}$$

Proof: Based on the properties of Z-scores, we know $\sum \hat{A}_i = 0$ and $\sum \hat{A}_i^2 = G$. Using the algebraic identity:

$$\begin{aligned} \sum_{i<j}(\hat{A}_i - \hat{A}_j)^2 &= G\sum_i \hat{A}_i^2 - \Big(\sum_i \hat{A}_i\Big)^2 \\ &= G \cdot G - 0 = G^2 \end{aligned} \tag{6}$$

Corollary: This value is completely independent of the scale, translation, or multiplier of the raw rewards (including our scaling factor c). Therefore, maximizing the global variance of advantages is an invalid optimization objective. To select c, we must evaluate the specific advantage gap between perfect and flawed responses.

F.3 Advantage Gap Analysis

In GRPO, the most critical signal is ensuring that perfect responses receive significantly higher advantages than incorrect ones. If we assume a simplified group containing only n_+ perfect responses ($r = 1$) and n_- completely incorrect responses ($r = -c$), with $p = n_+/G$, the advantages are mathematically independent of c:

$$\hat{A}_+ = \sqrt{\frac{1-p}{p}}, \quad \hat{A}_- = -\sqrt{\frac{p}{1-p}} \tag{7}$$

The gap $\hat{A}_+ - \hat{A}_- = \frac{1}{\sqrt{p(1-p)}}$ is unaffected by c. This means the scaling factor c only influences the gradient dynamics when the group simultaneously contains *partially correct* responses.

The scaling factor c controls the relative weight of partial correctness and the full-correctness bonus. For $c > 0$, BFR can be rewritten as

$$R_c(S) + c = cS + \mathbf{1}[S = 1]. \tag{8}$$

Under the group normalization used here, this reward has the same standardized advantages as

$$S + \frac{1}{c}\mathbf{1}[S = 1]. \tag{9}$$

Thus, a smaller c increases the relative weight of the full-correctness bonus, while a larger c places relatively greater emphasis on differences in partial correctness.

This effect depends on the composition of the response group. When all answers are imperfect, the mapping leaves standardized advantages unchanged. When a group contains fully correct answers together with imperfect answers of different correctness levels, the full-correctness bonus changes the allocation of relative advantages. Consequently, c provides an explicit means of adjusting the balance between full-correctness incentives and fine-grained correctness feedback.

F.4 Monte Carlo Sensitivity Analysis

Table 8: Simulated GRPO advantages across different scaling factors (c). "Perf" indicates Perfect responses, and "P-W" refers to the gap between Perfect and completely Wrong responses.

c	$\hat{A}$ (Perf)	$\hat{A}$ (0.75)	$\hat{A}$ (0.25)	$\hat{A}$ (Wrong)	Gap (P-W)	Gap (P-0.75)
0.05	1.676	0.090	-0.451	-1.136	2.812	**1.586**
0.50	1.655	0.202	-0.500	-1.201	2.856	1.453
0.80	**1.634**	**0.256**	**-0.521**	**-1.228**	**2.862**	**1.378**
1.00	1.620	0.286	-0.532	-1.241	2.861	1.334
2.00	1.561	0.387	-0.565	-1.278	2.839	1.175
5.00	1.471	0.505	-0.598	-1.311	2.782	0.966

To analyze the effect of c on the allocation of group-relative advantages, we conducted Monte Carlo simulations with group size $G = 8$, 5000 sampled prompts, and 5 levels of partial correctness. As shown in Table 8, we observe that:

- The critical gap between Perfect and Completely Wrong responses reaches its maximum at $c \approx 0.8 \sim 1.0$.
- The gap between Perfect and Near-Perfect (0.75) monotonically decreases as c increases, favoring smaller values.
- The gradient signal *within* partial answers (e.g., separating 0.75 from 0.25) increases with c.

Conclusion: We adopt the unit-scale setting $c = 1.0$. In the equivalent reward representation, this setting assigns equal unit weights to the correctness score and the full-correctness bonus. It preserves graded differences among partially correct answers and provides an independent reward increment for fully correct answers, combining fine-grained feedback and the objective of complete solutions on a common reward scale.

G Detailed Comparison with Related Works

We select LogicPro (Jiang et al., 2025) as our numerical baseline because it constructs reasoning data from algorithmic problems, program execution, and intermediate variables, providing a program-assisted synthesis pipeline for comparison. Related frameworks differ in their task coverage, verification mechanisms, and difficulty-control strategies.

SynLogic (Liu et al., 2025a) provides programmatic generators and task-specific verifiers for 35 logical tasks. Its difficulty control combines generation parameters with model-based solvability

filtering, and its experiments include both logic-only training and mixtures with mathematics or code data. Reasoning Gym (Stojanovski et al., 2025) provides over 100 reasoning environments with adjustable complexity and investigates dynamic curriculum training. ULTRALOGIC emphasizes the combination of decoupled input/solution logic, bilingual template expansion, a model-calibrated ten-level difficulty ladder, and BFR for structured final answers.

ScaleQuest (Ding et al., 2025) trains mathematical question generators, filters questions for solvability, and uses a difficulty scorer trained from model failure rates to select challenging questions. It generates multiple candidate solutions and selects them using a reward model.

MathSmith (Zhan et al., 2026) combines nine difficulty strategies with reinforcement learning of the problem-generation policy. Its reward includes structural validity, teacher-model reasoning length as a complexity proxy, and answer consistency across sampled solutions.

Chimera (Zhu et al., 2026) synthesizes challenging STEM problems and solution trajectories. It uses cross-model LLM verification to assess problem validity and answer correctness; this differs from computing ground truth through a deterministic task solver.

Table 9 summarizes these frameworks by generation paradigm, primary domain, answer verification, and difficulty control.

Table 9: Comparison of reasoning synthesis frameworks by generation paradigm, primary domain, answer verification, and difficulty-control mechanism.

Method	**Generation Paradigm**	**Primary Domain**	**Answer Verification**	**Difficulty Control**
ScaleQuest	LLM question synthesis	Mathematics	LLM solvability filtering; reward-ranked solutions	Failure-rate-trained difficulty scorer
Chimera	LLM problems and trajectories	STEM	Cross-model LLM verification	Prompt constraints; post-hoc difficulty analysis
MathSmith	LLM generation with RL	Mathematics	Sampled-answer consistency	Nine strategies; reasoning-length reward
LogicPro	Program-assisted synthesis	Code & Logic	Program execution and intermediate variables	Input-complexity tiers
SynLogic	Programmatic	Logic	Task-specific verifiers	Generation parameters; model-based filtering
Reasoning Gym	Procedural environments	Math, algorithms, logic, games	Task-specific verifiers	Configurable complexity; dynamic curricula
ULTRALOGIC (Ours)	**Programmatic (Decoupled)**	**Logic & Code & Math**	**Deterministic solvers; task-specific scoring**	**Model-calibrated ten-level ladder**