PHYS4037

Machine Learning in Science

Investigating Structured Reasoning Capabilities in Small-Scale Language Models Trained with Group Relative Policy Optimization for High School Physics Problem Solving

Natchira Chongsawad

20649126

CONVENOR

Prof Simon Dye

School of Physics and Astronomy

University of Nottingham

United Kingdom

Contents

Abstract

Large language models show strong reasoning ability; however, step-by-step training on memory-constrained hardware remains costly. The study presents an approach using a small model with Qwen 2.5 3B Instruct as the base, fine tuning with LoRA to lower memory use, and applying Group Relative Policy Optimization without a critic together with a structured reasoning template and multidimensional rewards. The goal is to improve understanding and answer quality. The dataset contains high school physics problems in dialogue form. The reward system checks format tags and numerical accuracy with rules set for every question type. Multiple-choice items must match the chosen option and be correct, whereas numerical open-ended items are checked for unit and for value within an acceptable range. On 199 test items, overall accuracy rose from 0.45 to 0.82, with multiple-choice accuracy improving from 0.45 to 0.85 and open-ended accuracy from 0.46 to 0.76, while format adherence stayed around 0.98. Efficiency per resource matches or exceeds larger baselines. Further analysis indicates that an absolute percentage error penalty which is less strict than percentage squared error improves learning of numeric answers. Additionally, the language of the question matters because the fine-tuned model is more accurate on English numerical questions than on Chinese ones, even when the English set is translated from Chinese.

1 Introduction

Large language models have advanced rapidly, enabling document summarisation, translation, information extraction, and step-by-step problem solving [1]. A key reason for this progress is the Transformer, which uses self-attention for sequence modelling [2, 3]. For each token, the model constructs query, key, and value vectors [4]. It then computes similarity between queries and keys to obtain attention weights and uses these weights to aggregate the corresponding values [3]. Positional encodings provide order information, residual connections support stable optimisation, position-wise feed-forward layers increase model capacity [2], and multi-head attention enables parallel capture of multiple patterns [2]. Three main types are widely used [1]. Encoder–decoder architectures separate input processing and output generation [1, 5]. The encoder handles the input with bidirectional attention, and the decoder produces the output while attending to the encoded representation [5]. The design is well matched to translation and source-grounded summarisation [5]. Decoder-only models adopt a single stack that continues a prompt token by token, supporting instruction following and open-ended text generation [1]. Prefix-decoder models follow the decoder-only pattern but apply bidirectional attention to a visible prefix before generating the remaining tokens autoregressively [1]. Tasks with a clear lead-in context are well suited to the prefix-decoder approach [6].

In practice, large language models rely on several core techniques. First, self-supervised pre-training learns from large unlabelled text corpora [7, 8]. After pre-training, the model can be fine-tuned with labelled data for a specific task. Second, few-shot prompting guides a pre-trained model by giving a small number of examples in the prompt without changing the parameters [9]. This approach often provides a short and effective route to a solution [10]. Furthermore, Reinforcement Learning from Human Feedback aligns the model with user preferences [11]. The process starts with supervised fine-tuning on instruction–answer pairs, continues with training a reward model from human pairwise rankings to learn a scoring function, and ends with policy optimisation using a reinforcement algorithm such as PPO with a KL penalty to keep the policy close to a reference model [11, 12]. This pipeline appears in systems such as Llama-2-Chat [11].

Modern language models often require substantial memory and compute for both training and inference [1]. Larger parameter counts, longer context windows, and alignment stages increase VRAM usage, latency, energy use, and cost, which in turn limits reproducibility, fine-tuning, and systematic evaluation [13, 14, 15]. This study uses decoder-only architectures such as Llama [16] and Qwen [17] and applies reinforcement-learning post-training with Group-Relative Policy

Optimization (GRPO) [18] to encourage step-by-step explanations and unit-aware answers for high-school physics problems. The method can generate multiple candidate answers per prompt [19] when needed and avoids training a critic network or a separate reward model, reducing parameter load and VRAM use [16]. The aim is to fine-tune small models with verifiable, task-aligned training signals that support stepwise reasoning. The study examines how well small open models solve high-school science problems under tight resource and parameter budgets, with a focus on clarity of explanation, content accuracy, and practical usability.

2 Related Work

This chapter summarizes foundations for the experiments, including the Transformer architecture, differences between encoder–decoder and decoder-only, post-training with GRPO, details of small LLMs used in practice, and LoRA fine-tuning, to establish the theoretical and technical basis for the subsequent analysis.

2.1 Transformer

Effective reasoning in large language models often depends on the Transformer architecture [20, 21, 22]. Self-attention [5] captures long-range relationships across a sequence. A Transformer block in an encoder or a decoder begins by mapping tokens to embedding vectors and adding a positional signal to every token [5]. To form the input to a block, the token sequence $x_{1:n}$ is tokenised, converted to embedding vectors, and augmented with positional signals.

$$X = [e_1 + p_1;\ e_2 + p_2;\ \ldots;\ e_n + p_n] \in \mathbb{R}^{n \times d_{\text{model}}}, \qquad e_i = E[x_i]. \tag{1}$$

where $x_{1:n}$ denotes a token sequence of length n. E is the embedding matrix. $e_i \in \mathbb{R}^{d_{\text{model}}}$ is the embedding of token. x_i; $p_i \in \mathbb{R}^{d_{\text{model}}}$ is the positional vector. $X \in \mathbb{R}^{n \times d_{\text{model}}}$ is the block input and d_{model} is the model width. In theory, embeddings learn coordinates where dot products reflect semantic closeness.

To preserve sequence information and reduce order invariance in self-attention, alternating sinusoidal sin and cos positional signals scaled by d_{model} are applied, as in Equation (2). Absolute positional encoding [5] assigns a fixed vector to each position to mark order.

$$\text{PE}(\text{pos}, 2k) = \sin\left(\frac{\text{pos}}{10000^{\,2k/d_{\text{model}}}}\right), \qquad \text{PE}(\text{pos}, 2k+1) = \cos\left(\frac{\text{pos}}{10000^{\,2k/d_{\text{model}}}}\right). \tag{2}$$

Here, $\text{pos} \in \{1, \ldots, n\}$ is the position index. $k = 0, 1, \ldots, \lfloor d_{\text{model}}/2 \rfloor - 1$ is the paired channel index and d_{model} is the model width. Key phase shift terms $\sin(\alpha - \beta)$ and $\cos(\alpha - \beta)$ cause the dot product of vectors at different positions to depend systematically on the positional difference, preserving relative information even with absolute encoding.

Another option is rotary or relative positional encoding [5, 23], capturing position by rotating sub-vectors to produce position-aware representations.

$$\langle R_m q,\ R_n k \rangle = \langle q,\ R_{m-n} k \rangle. \tag{3}$$

In this equation, $q, k \in \mathbb{R}^{d_k}$ are a query and a key. R_m is a rotation operator tied to position m. and $\langle \cdot, \cdot \rangle$ is the dot product. The attention logit is directly determined by relative distance $m - n$.

After that, multi-head self-attention [5] computes Q, K, and V from X through matrix multiplications. Attention weights summarise global context for each position and produce updated representations.

$$Q = XW^Q, \qquad K = XW^K, \qquad V = XW^V. \tag{4}$$

Given per-head parameter matrices $W^Q, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, the projections produce $Q, K \in \mathbb{R}^{n \times d_k}$ and $V \in \mathbb{R}^{n \times d_v}$. Here, d_k is the key and query dimension per head, and d_v is the value dimension per head.

As d_k increases, the variance of $\langle q, k\rangle$ grows roughly in proportion to d_k. Scaling by $1/\sqrt{d_k}$ as in Equation (5) keeps the logits within a stable range near 1, reduces softmax saturation at high dimensionality [5], and supports training stability.

$$\mathrm{Var}\big[\langle q, k\rangle\big] \approx d_k \quad \Rightarrow \quad \mathrm{Var}\left[\frac{\langle q, k\rangle}{\sqrt{d_k}}\right] \approx 1. \tag{5}$$

where $q, k \in \mathbb{R}^{d_k}$ are vectors for a single position. $\mathrm{Var}[\cdot]$ denotes variance under the usual i.i.d. assumption. Thus, scaled dot-product [5] attention is

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right) V. \tag{6}$$

The mask M defines the attention pattern. In the encoder $M = 0$. In the decoder $M_{ij} = -\infty$ for $j > i$ to keep autoregressive generation. Compute attention for heads $h = 1, \ldots, H$, concatenate the outputs along the feature axis, then project with W^O to obtain $\mathrm{MHSA}(X)$ as in Equation (7) [5] to aggregate multi-scale views.

$$\mathrm{MHSA}(X) = \mathrm{Concat}_{h=1}^{H}\Big(\,\mathrm{Attn}\big(XW_Q^{(h)},\, XW_K^{(h)},\, XW_V^{(h)}\big)\Big)\, W^O. \tag{7}$$

where $X \in \mathbb{R}^{n \times d_{\text{model}}}$ is the block input, H is the number of heads, $W_Q^{(h)}, W_K^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W_V^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are per-head projections. For stability, the block applies a skip connection [24] and post-norm layer [25] normalisation to the sum $X + \mathrm{MHSA}(X)$.

$$S = \mathrm{LayerNorm}\big(X + \mathrm{MHSA}(X)\big). \tag{8}$$

According to Equation (8), $S \in \mathbb{R}^{n \times d_{\text{model}}}$ represents the residual sum with layer normalisation applied to each token. Processing then moves to the position-wise feed-forward network [5], an MLP shared across all positions. The network applies the two-layer mapping to add a non-linear transformation.

$$\mathrm{FFN}(S) = \phi\big(SW_1 + b_1\big)W_2 + b_2. \tag{9}$$

Equation (9) shows that $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ and $b_1 \in \mathbb{R}^{d_{\text{ff}}}$ are first layer parameters. $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ and $b_2 \in \mathbb{R}^{d_{\text{model}}}$ are second layer parameters. ϕ denotes the nonlinearity. d_{ff} is the hidden width.

Finally, apply a residual connection and layer normalisation to the FFN output to produce the block output H [5]. Using residuals and layer normalisation at both stages stabilises training, reduces vanishing gradients, and preserves context across the sequence.

$$H = \mathrm{LayerNorm}\big(S + \mathrm{FFN}(S)\big). \tag{10}$$

A Transformer block follows a steady loop. Stacking multiple blocks [5] enables the model to gradually build deeper and more powerful abstract representations for sequential data [26, 27].

In an encoder–decoder design cross-attention [5] transfers information from the encoder to the decoder. Training updates the parameters by reducing the language modelling loss, as shown in Equations (11) and (12).

$$\mathrm{CrossAttn}(Q_{\mathrm{dec}}, K_{\mathrm{enc}}, V_{\mathrm{enc}}) = \mathrm{softmax}\left(\frac{Q_{\mathrm{dec}} K_{\mathrm{enc}}^{\top}}{\sqrt{d_k}}\right) V_{\mathrm{enc}}. \tag{11}$$

where $Q_{\mathrm{dec}} \in \mathbb{R}^{n_{\mathrm{dec}} \times d_k}$ comes from decoder states. $K_{\mathrm{enc}} \in \mathbb{R}^{n_{\mathrm{enc}} \times d_k}$ and $V_{\mathrm{enc}} \in \mathbb{R}^{n_{\mathrm{enc}} \times d_v}$ come from the encoder. n_{enc} and n_{dec} are sequence lengths. Causal masking is usually not applied.

$$\mathcal{L}_{\mathrm{LM}} = -\sum_{t=1}^{n} \log p_\theta(x_t \mid x_{<t}). \tag{12}$$

where $x_{<t} = (x_1, \ldots, x_{t-1})$ is the left context. p_θ is the model distribution with parameters θ. n is the sequence length. The language modelling loss $\mathcal{L}_{\mathrm{LM}}$ [28] is the token-level cross-entropy to be minimised.

Decoder-only Transformers keep the block structure while removing the encoder and cross-attention. Autoregressive generation requires causal self-attention [5] with the mask $M_{ij} = -\infty$ for $j > i$ as in Equation (6). Training minimises the next-token cross-entropy in Equation (12). The configuration scales well for open-ended text generation and large datasets [29].

2.2 Group-Relative Policy Optimization

GRPO is a reinforcement learning method for language models [18]. The method follows a PPO-style scheme that compares multiple responses to the same prompt within a group [18]. For each prompt q, a set of G complete outputs $o_{1:G} \sim \pi_{\mathrm{old}}(\cdot \mid q)$ is sampled and rewards are defined as $r_i = r(q, o_i)$. The update raises the probability of higher-reward outputs and reduces that of lower-reward ones, without a value function [18, 30]. it is defined as

$$J_{\mathrm{GRPO}}(\theta) = \mathbb{E}_{q,\{o_i\}}\left[\frac{1}{G}\sum_{i=1}^{G}\Big(\min(\rho_i A_i,\ \mathrm{clip}(\rho_i, 1-\varepsilon, 1+\varepsilon)A_i) - \beta\, \mathrm{D}_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}})\Big)\right]. \tag{13}$$

$$\rho_i \coloneqq \frac{\pi_\theta(o_i \mid q)}{\pi_{\mathrm{old}}(o_i \mid q)}. \tag{14}$$

From Equation (13), ε and β are hyperparameters; A_i denotes the group-standardised advantage; ρ_i is the importance ratio; and π_{ref} is the fixed reference policy.

GRPO employs a sequence-level advantage computed by standardising rewards within each prompt-specific group [30].

$$A_i \coloneqq \frac{r_i - \mu_q}{\sigma_q}, \qquad \mu_q \coloneqq \frac{1}{G}\sum_{j=1}^{G} r_j, \quad \sigma_q \coloneqq \sqrt{\frac{1}{G}\sum_{j=1}^{G}(r_j - \mu_q)^2}\,. \tag{15}$$

The term μ_q serves as a per-prompt baseline [18]. Because samples are drawn from $\pi_{\theta_{\mathrm{old}}}$, subtracting μ_q leaves the expected policy gradient with respect to θ unchanged [31]. Normalising by σ_q stabilises the scale and reduces variance [30]. The KL penalty [18, 30] to π_{ref} is commonly approximated per sampled output by

$$\widehat{D}_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}}) \;\approx\; \frac{\pi_{\mathrm{ref}}(o_i \mid q)}{\pi_\theta(o_i \mid q)} \;-\; \log\frac{\pi_{\mathrm{ref}}(o_i \mid q)}{\pi_\theta(o_i \mid q)} \;-\; 1\,. \tag{16}$$

GRPO and Proximal Policy Optimisation with Reinforcement Learning from Human Feedback (PPO-RLHF) follow the PPO idea of reward based policy updates [14]. GRPO compares multiple responses from the same prompt without a critic, offers simple setup, remains robust to reward scale shifts, but requires many generations per question and lacks token level credit assignment [30, 32]. PPO RLHF uses a learned critic to evaluate the sequence over time, enables token level credit assignment and supports learning with long contexts or delayed rewards, but requires critic training and tuning and may introduce bias when the critic is inaccurate [14, 18].

2.3 Small-Scale Large Language Models

In this context, five small decoder only models are introduced. All experiments use instruct checkpoints of the models, and core specifications match the base models in parameter count, tokenizer, and context window.

2.3.1 Llama-3.2 (1B and 3B)

Llama-3.2 [33, 34, 35, 36, 37, 38] uses a decoder-only Transformer with Grouped-Query Attention (GQA) to improve inference efficiency. The standard release supports a 128k-token context window and targets resource-constrained deployment. Available variants are pretrained and instruct. The instruct variant applies supervised fine-tuning and alignment methods such as RLHF and Direct Preference Optimization (DPO). Pretraining uses up to about 9 trillion tokens. The 1B model has about 1.23 billion parameters. The 3B model keeps the same design and increases the size to about 3.21 billion parameters. Distribution uses the Llama-3.2 Community License.

2.3.2 Qwen-2.5 (3B and 7B)

Qwen-2.5 [39, 40, 41, 42] adopts a decoder-only Transformer with GQA, Rotary Positional Embedding (RoPE), Root Mean Square Layer Normalization (RMSNorm), Swish-Gated Linear Unit (SwiGLU). The tokenizer is a byte-level Byte Pair Encoding (BBPE) with about 151k tokens. The 3B model has about 3.09 billion parameters and a 32,768-token context window. Pretraining focuses on higher quality data in code and mathematics. Post-training uses supervised fine-tuning and reinforcement learning to strengthen instruction following and structured output. The 7B model has about 7.61 billion parameters with GQA configured as 28 query groups and 4 key-value groups. The long-context setup for 7B reaches 131,072 tokens. The series offers broader multilingual coverage and stronger coding and math reasoning than earlier Qwen versions.

2.3.3 Llama-3.1 (8B)

Llama-3.1-8B [43, 44] is an autoregressive decoder-only Transformer with GQA, RoPE and SwiGLU. The context window reaches 128k tokens. The instruct release uses supervised fine-tuning with RLHF or DPO to improve helpfulness and safety in chat use. Distribution uses the Llama-3.1 Community License.

2.4 Low-Rank Adaptation

LoRA [45] provides a parameter-efficient fine-tuning method for large language models under compute and memory limits. Base weights remain frozen and low-rank adapters approximate

the weight change. Common insertion points are the attention projections W_q and W_v with optional use in W_o or the feed-forward layer. A small rank r is typically used [45]. The core relations are presented below.

First, the low-rank update is defined in Eq. (17).

$$\Delta W = BA \tag{17}$$

where $W \in \mathbb{R}^{m \times n}$ denotes the frozen weight matrix, $B \in \mathbb{R}^{m \times r}$ and $A \in \mathbb{R}^{r \times n}$ are the trainable factors, and $r \ll \min(m, n)$.

Next, the adapted weight used during training and inference is formed as

$$W' = W + \frac{\alpha}{r} BA \tag{18}$$

where α is a scaling constant and only A and B are updated.

When applied to the attention projections the forward pass is modified as shown in Eq. (19) and Eq. (20).

$$q = \left(W_q + \frac{\alpha}{r} B_q A_q \right) h \tag{19}$$

$$v = \left(W_v + \frac{\alpha}{r} B_v A_v \right) h \tag{20}$$

Here, h denotes the input hidden state, W_q and W_v are the query and value projection matrices, and B_q, A_q, B_v, A_v are the LoRA adapters in the matching projections. The same form can be used for W_o or for the feed-forward layer.

Model training then minimises cross-entropy under the low-rank update as shown in Eq. (21).

$$\min_{A,B} \; \mathbb{E}_{(x,y^*)\sim\mathcal{D}} \Big[\mathcal{L}_{\mathrm{CE}} \big(f_{\theta,\, W + \frac{\alpha}{r} BA}(x),\, y^* \big) \Big] \tag{21}$$

where x is the input, y^* is the target, and $f_{\theta,\, W + \frac{\alpha}{r} BA}$ denotes the model with non-LoRA parameters θ kept fixed.

Finally, the number of trainable parameters [46] follows

$$\#\text{trainable} = r(m + n) \tag{22}$$

The reduction from mn to $r(m + n)$ explains the memory and compute savings under limited hardware.

3 Methodology

This method develops a small physics reasoning model that maintains depth of thought and balances speed, compute, memory, and problem solving power. Reinforcement learning is the core approach, with GRPO guiding updates to produce stable solutions, clear step by step reasoning, and accuracy of answers. The workflow has three stages, and the following sections will describe each stage in detail.

3.1 Base Model

In the initial phase, multiple pre-trained language models were surveyed to identify a suitable base model for fine-tuning. Selection criteria focused on answer correctness and efficient use of compute and memory.

Accuracy assessment. The AI-MO/NuminaMath-TIR dataset [47] was adopted as the evaluation set. The dataset contains both open-ended and multiple-choice mathematics questions together with step-by-step reasoning, which is appropriate for judging readiness for further development. Training used a GRPO-based pipeline [48], and the reward function is described in Appendix A.

Efficient GPU usage. The study was carried out on an MLiS2 system with two NVIDIA GeForce RTX 2080 Ti GPUs, each providing 11,264 MiB (about 11 GiB) of VRAM. The model design and training configuration were adapted to fit these memory limits.

Qwen 2.5 3B was selected as the final base model because it balanced accuracy and memory use on this hardware. Hyperparameters were adjusted to the machine capacity. The final settings and the results are reported in Tables 1 and 2.

Table 1: Training configuration and GPU memory use for Qwen 2.5 3B on two RTX 2080 Ti GPUs with about 11 GiB each. Items include per-GPU VRAM, hyperparameters, and total training days.

Setting	Value
Base model	Qwen 2.5 3B
GPU 1 usage (MiB)	10,720
GPU 2 usage (MiB)	8,796
`bnb_4bit`	False
LoRA r	8
LoRA α	32
Gradient accumulation steps	64
Per-device training batch size	2
Maximum prompt length (tokens)	256
Maximum completion length (tokens)	512
Epochs	30
Training duration (days)	4.127

According to Table 1, Qwen 2.5 3B required approximately 10.7 GiB on GPU 1 and 8.8 GiB on GPU 2 without 4-bit quantisation. The configuration supports a maximum prompt of 256 tokens and a maximum completion of 512 tokens, which is adequate for step-by-step reasoning. A training duration of about 4.13 days was adopted to improve numerical stability and enable longer outputs.

Table 2: Reward metrics for Qwen 2.5 3B under a GRPO fine-tuning pipeline on AI-MO/NuminaMath-TIR comparing base and fine tuned accuracy and format rewards with deltas.

Metric	Base	Fine-tuned	Δ
Accuracy reward	0.1717	0.2323	+0.0606
Format reward	0.2828	0.9293	+0.6465

As shown in Table 2, fine-tuning Qwen 2.5 3B produced consistent improvements in both reward metrics. The largest gain was observed in adherence to the required answer format, with a clear improvement in answer accuracy as well. These outcomes indicate strong suitability for further fine-tuning on physics-related questions.

3.2 Dataset

This project targets high-school physics question answering. Several open physics QA datasets on Hugging Face were combined and reformatted into JSON for fine-tuning. The collection covers free-response and multiple-choice items, including numerical calculation tasks and factual or conceptual questions.

Table 3: Reference table of Hugging Face datasets and question categories. Counts are provided for numerical open ended, long factual open ended, numerical multiple choice, and long factual multiple choice, with totals in the last row.

Dataset (Hugging Face)	Numerical open-ended	Long/factual open-ended	Numerical multiple-choice	Long/factual multiple-choice
`cais/mmlu` [49]	56	2	26	89
`derek-thomas/ScienceQA` [50]	0	0	451	562
`mrohith29/high-school-physics` [51]	260	1	0	139
`cristiano-sartori/conceptual_physics` [52]	26	0	0	209
`qgallouedec/physics-problems` [53]	247	0	0	0
`BaunRobotics/physics` [54]	28	2	0	0
`nathannarrik/physics_tutor` [55]	11	1	0	0
Total	**628**	**6**	**477**	**999**

This research uses seven open physics question–answer datasets released on Hugging Face. The collection includes the high-school physics part of MMLU [49] and the physics subset of ScienceQA [50], which contains questions from open educational resources such as those managed by IXL Learning. The dataset mrohith29/high-school-physics [51] is also included. Together, these datasets cover topics studied in school physics, such as kinematics, mechanics, electricity and circuits, thermodynamics, optics, electromagnetism, electronic devices, atomic and modern physics, periodic motion, and waves or oscillations.

The conceptual physics dataset [52] focuses on conceptual and factual multiple-choice questions. In contrast, the BaunRobotics/physics dataset [54] is almost all numerical and open-ended, with step-by-step solutions and no multiple-choice items. Their license rules are followed to avoid copyright problems.

Data preparation. This project applies a data cleaning process to these datasets. All records were changed into a four-column format: question, options, chain of thought (CoT), and answer. The dataset is designed with two main groups: open-ended questions and multiple-choice questions. Each item was then assigned to an output type based on its correct answer. If the

answer was a number with a physical unit (for example, 400 N, 3.2 m/s^2, 2×10^{-3} A), the item was treated as open-ended and the options column was left empty. If the answer was text the item stayed as multiple-choice and the full set of options was kept. For multiple-choice questions, the answer had to be written the same as one of the options. Using a single schema made it possible to combine both formats in one training pipeline. Table 4 presents examples after reformatting.

Table 4: Structure of the training record with formatting rules and examples for open ended and multiple choice items. Table reports the data type and example text for question, options, reasoning steps, and answer.

Column	Type	Open-end questions (example)	multiple-choice questions (example)
question	string	"A sound wave has a frequency of 1,500 Hz and travels at a speed of 340 m/s in air. What is its wavelength?"	"Compare the motion of three cars. Which car was moving at the lowest speed?"
options	list of strings	`[]`	`["a car that moved 140 miles west in 10 hours", "a car that moved 640 miles east in 10 hours", "a car that moved 355 miles east in 10 hours"]`
CoT	string	"$\lambda = v/f = 340$ m/s $\div$ 1500 Hz."	"Look at the distance each car moved and the time it took to move that distance. The direction each car moved does not affect its speed.\nNotice that each car moved for 10 hours. The car that moved 140 miles moved the shortest distance in that time. So, that car must have moved at the lowest speed."
answer	string	"0.227 m"	"a car that moved 140 miles west in 10 hours"

Subsequently, to ensure consistent, reliable evaluation by the reward function, duplicate entries were removed. Scientific notation and units were standardized. A space was inserted between numbers and units. Examples of the data transformation are shown in Tables 5–7.

Table 5: Standardizing numeric notation for dataset cleaning. Left column shows original forms and right column shows unified forms using caret for powers and asterisk for multiplication.

Before	**After**
10^2	`10^2`
$\times 10^{-7}$	`*10^-7`
`3e-7`	`3*10^-7`
`x 10^4`	`*10^4`

Table 5 illustrates that adopting a uniform style (10^k) prevents parsing errors across libraries and ensures reliable numeric comparison in reward calculations.

Table 6: Standardizing units and symbols for dataset cleaning. Table shows raw entries and normalized forms.

Before	After
`pi` / π written as plain text	π (typeset as `$ \pi $`)
`ohm`	Ω
`m/s2`, `m/s`^{2}	$\mathrm{m/s^2}$
`m`^{3}	$\mathrm{m^3}$
`O` (used for degrees)	$^\circ$
`minutes.`, `years.`	`minutes`, `years` (remove trailing periods)
`Joules`	`J`
`4.18 J/g°C`	4184 $\mathrm{J/(kg \cdot K)}$

Table 6 shows that different ways of writing the same unit may cause incorrect errors. When the units and spacing are made consistent, the reward function can check the number and the unit separately with confidence. Some entries must be manually revised or converted to reduce ambiguity, as indicated in Table 7.

Table 7: Samples of manual corrections and calculated values. Entries show common formatting adjustments and evaluation to a clear numeric form such as adding a space between value and unit and converting an inverse tangent to a degree value.

Before	After
`16 000 J`	`16000 J`
`-36m`	`-36 m` (add space between value and unit)
$\tan^{-1} 4$	75.96° (evaluate to a clear degree value)

In addition, some questions and answers were rephrased, and multi-target questions were split into separate items so that each question had only one clear answer. For example, a question that originally asked for both "net force" and "acceleration" was divided into two questions. Some records were removed, including cases outside the scope of physics problem solving. Edits and removals reduced ambiguity and label conflicts, improving reliability without reducing coverage. The cleaned dataset on Hugging Face, psamtam/high school physics dataset [56], contains 1,997 items.

After cleaning and standardising the dataset, the next step is to explore the structure and distributions to gain a clear overall picture of the data.

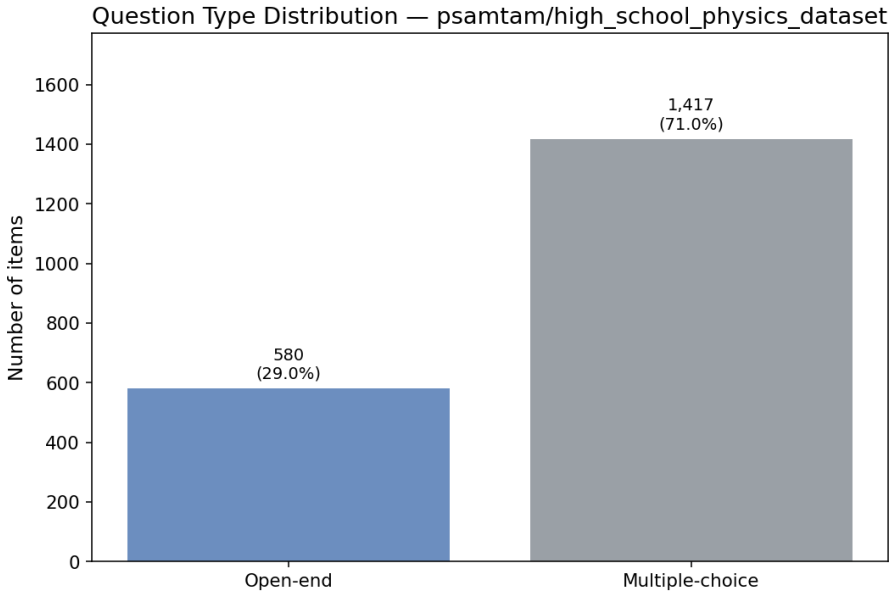

Figure 1: Question Type Distribution. Bar chart showing the proportion of open end and multiple choice questions in the dataset.

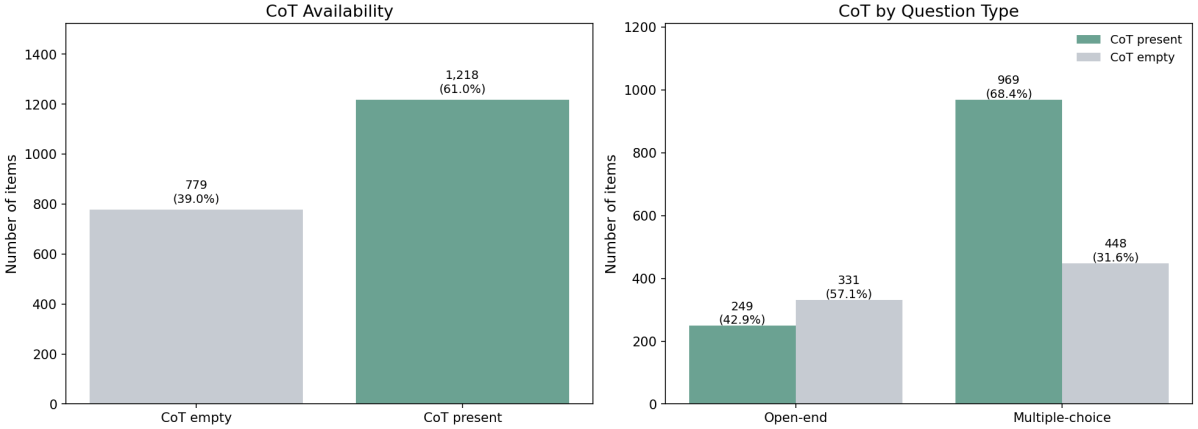

Figure 2: CoT Annotations and Coverage. Two panel figure. Left panel indicates overall availability of chain of thought annotations. Right panel presents the distribution of chain of thought across open end and multiple choice questions.

Figure 1 shows that the dataset is split into 580 open-ended questions (29.0%) and 1,417 multiple-choice questions (71.0%). The distribution is clearly unbalanced, with a strong tilt toward multiple-choice items. For Chain-of-Thought (CoT) annotations (Figure 2), 1,218 records contain reasoning steps (61.0%), while 779 records do not (39.0%). When divided by question type, only 249 of the 580 open-ended questions have CoT (42.9%), showing that more than half of the calculation problems are without reasoning traces (331 items; 57.1%). By contrast, 969 of the 1,417 multiple-choice questions include CoT (68.4%), which means that most reasoning examples are found in the MCQ portion of the dataset.

3.3 Fine-tuning through Reinforcement Learning with GRPO

This section explains the entire training process. It begins with prompt creation and then moves to the configuration of training arguments. After the training is complete, the model is saved

and uploaded to Hugging Face.

Constructing Prompts. The procedure began with adjustment of the input data. Dataset preparation started with inspection of the options field in each example. When options were absent, the original question remained unchanged. When options were present, the string "Here are the options: [option 1, option 2, ...]" was appended to the question so that the question and the options appeared on a single line. The question column was updated with the combined text. All examples were converted to a chat-style prompt with two roles: a system message defining assistant behaviour and a user message containing the revised question. The order followed a fixed pattern: system first, then user. Table 8 presents the final format.

Table 8: Instruction and conversation format used in training. Table shows the system prompt the user prompt structure and the target completion. Reasoning is enclosed by `<think>` tags and final answers are enclosed by `<answer>` tags. When options are given the wording follows the options. Otherwise answers use a number and a unit with a space between them.

Field	Content

SYSTEM_PROMPT	`SYSTEM_PROMPT = (`
	`"A conversation between User and Assistant. The user asks a question,`
	`and the Assistant solves it. The assistant "`
	`"first thinks about the reasoning process in the mind and then provides`
	`the user with the answer. The reasoning "`
	`"process and answer are enclosed within <think> </think> and <answer>`
	`</answer> tags, respectively, i.e., "`
	`"'<think> reasoning process here </think><answer> answer here </answer>'`
	`If the question provides options, use "`
	`"the exact wordings in the options. Otherwise, answer with a numerical`
	`number and units, with a space between them."`
	`)`

prompt	`prompt = [`
	`{"role": "system", "content": SYSTEM_PROMPT},`
	`{"role": "user", "content": example["problem"]},`
	`]`

target_completion	`target_completion = (`
	`f"<think>{example['rationale']}</think>"`
	`f"<answer>{example['final_answer']}</answer>"`
	`)`

Base Architecture and LoRA Parameters. The experiments begin with the Qwen2.5-3B-Instruct model, where a lightweight LoRA adapter is applied only to the attention projection layers (`q_proj` and `v_proj`). The setup specifies rank $r = 8$, scaling $\alpha = 32$, and dropout $p = 0.1$. This design reduces the number of trainable parameters and memory usage during reinforcement learning, while keeping the backbone model frozen.

Reward functions. The reward has two parts: a format score that checks structural validity, and an accuracy score that measures how well the content matches the task. When more than one reward function is used, their outputs are averaged. For each completion i,

$$r_i \;=\; \tfrac{1}{2}\big(r_i^{\text{fmt}} + r_i^{\text{acc}}\big). \tag{23}$$

Here r_i^{fmt} is a binary format reward and r_i^{acc} is the accuracy reward. Overall performance is reported as the mean reward across items.

1. **Format reward.** The output must contain the tags `<think>` ... `</think>` and `<answer>` ... `</answer>`. If this structure is missing or malformed, the format score is zero. To handle multi-line content inside the tags, the check allows newlines and whitespace. Formally,

$$r_i^{\text{fmt}} = \begin{cases} 1, & \text{if it matches } \texttt{<think>}\ldots\texttt{</think><answer>}\ldots\texttt{</answer>}, \\ 0, & \text{otherwise.} \end{cases} \tag{24}$$

2. **Accuracy reward.** Scoring depends on item type:

   (a) **Multiple-choice questions (MCQ).** Exact-match scoring: let $\hat{y}$ be the model prediction, y the ground-truth option, and $\mathcal{O}$ the set of provided options,

$$r_i^{\text{acc}} = \begin{cases} 1.0, & \hat{y} = y, \\ 0.1, & (\hat{y} \in \mathcal{O}) \ \wedge \ (\hat{y} \neq y), \\ 0, & \text{otherwise.} \end{cases} \tag{25}$$

   (b) **Open-ended numerical answers.** The score is the sum of unit validity and numeric proximity,

$$r_i^{\text{acc}} = r_i^{\text{unit}} + r_i^{\text{num}} \in [0, 1]. \tag{26}$$

   Unit validity. Normalize unit strings and verify dimensional equivalence between the predicted unit $\hat{u}$ and reference unit u^* (e.g., via a unit library):

$$r_i^{\text{unit}} = \begin{cases} 0.5, & \text{dim\_eq}\big(\text{norm}(\hat{u}),\, \text{norm}(u^*)\big) = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{27}$$

   Numeric proximity. Let the absolute percentage error (APE) be

$$\text{APE} = \frac{|\hat{y} - y|}{|y|}, \tag{28}$$

   and convert it to reward via a smooth, saturating penalty

$$\text{penalty}(|e|) = c\big(1 - e^{-|e|/\tau}\big), \qquad c = 0.5,\ \tau = 0.05. \tag{29}$$

   Then

$$r_i^{\text{num}} = \begin{cases} 0.5, & \text{if the values match when rounded to four significant figures,} \\ 0.5 - \big(0.5(1 - e^{-\text{APE}/0.05})\big), & \text{otherwise.} \end{cases} \tag{30}$$

To gain a clearer view of how the penalty is determined for numerical answers, it is necessary to examine how the score changes according to the distance between the predicted value and the actual value.

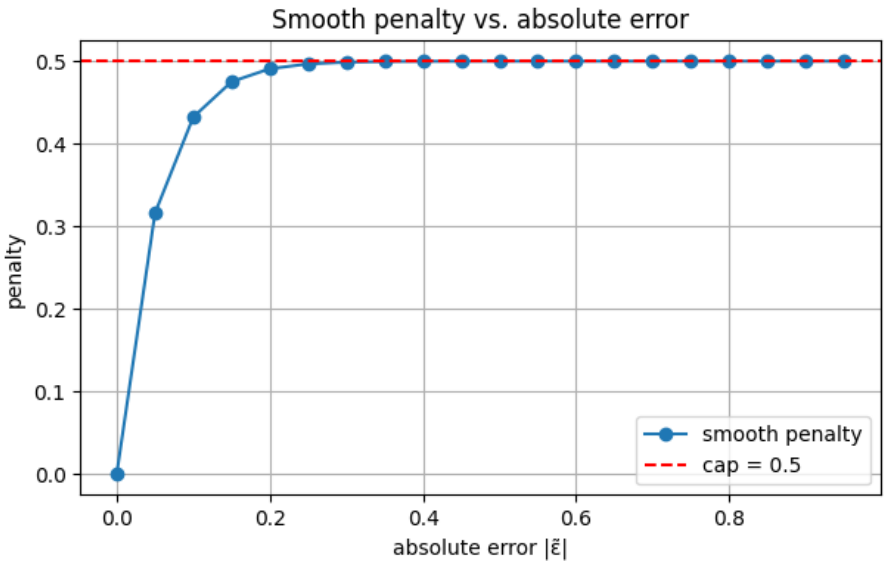

Figure 3: Smooth Penalty versus Absolute Error. Line plot showing a smooth penalty that increases with absolute error and then levels off at a fixed cap. The solid curve shows the smoothed function set by a temperature value. The dashed line marks the cap.

As shown in Figure 3, the smooth penalty rises gradually with the absolute error. In line with Equation (29), the penalty starts at 0 when the error is zero, increases to about 0.31 at a 5% error, and approaches the cap of 0.5 by around 20% error. This shape ensures that the derived value reward, defined explicitly as $\max(0.5 - \text{penalty}(|e|), 0)$ in Equation (30), remains bounded and interpretable. To avoid trivial penalties from formatting issues, both the model output and the reference value are rounded to four significant figures for the exact equality check. If they do not match, the smooth, saturating penalty is then computed from the unrounded magnitudes.

This design also minimizes confusion from unit notation, accepts small rounding differences, and grants meaningful partial credit. As a result, answers with correct units and values close to the target are rewarded more, while inconsistent units or large errors are penalized, ensuring fair and robust evaluation.

Training with Group-Relative Policy Optimization. Training pipeline applies GRPO in TRL. The configuration uses GRPOTrainer with the specified GRPOConfig and disables column pruning (remove_unused_columns=False) to allow reward functions to access ground-truth fields during training. Policy updates maximize a PPO-style clipped objective with KL regularization. The expectation is over prompts and groups of G samples drawn from the old policy.

Table 9: Training arguments configured with GRPOConfig. Table lists learning rate, gradient accumulation steps, number of epochs, per device train and eval batch sizes, bf16 use, maximum prompt and completion lengths, number of generations, and the unused columns setting.

Hyperparameter	**Value**
learning_rate	`1e-5`
remove_unused_columns	`False`
gradient_accumulation_steps	`64`
num_train_epochs	`30`
per_device_train_batch_size	`2`
per_device_eval_batch_size	`2`
bf16	`True`
max_completion_length	`512`
num_generations	`4`
max_prompt_length	`256`

Table 9 lists the GRPOConfig hyperparameters used in this setup. Hyperparameters were tuned with an accuracy sufficiency approach. The goal was minimal and stable settings that meet the target QA accuracy within limits on compute and memory. Each optimizer step aggregates 64 micro steps with a per-device micro batch size of 2, giving an effective per-device batch of $2 \times 64 = 128$ prompts. With group size $G = 4$, each update processes $128 \times 4 = 512$ completions. Prompt length is capped at 256 tokens and completion length at 512 tokens. The theoretical upper bound on newly generated tokens per update is $512 \times 512 = 262{,}144$ tokens, although practical usage is lower due to early stopping. Training runs for 30 epochs. Parameter-efficient tuning uses LoRA adapters with rank 8, alpha 32, and dropout 0.1, applied only to `q_proj` and `v_proj`. Adapters are saved for release.

4 Results

After 30 epochs, training logs and a held-out test set were reviewed to assess understanding of the target writing task, compliance with the specification, and output correctness. The `Qwen2.5-3B` model was trained with the GRPO method, and metrics were recorded on Hugging Face under `Ppear/Qwen2.5-3B-GRPO-Physics_expo` [57]. Three training metrics were tracked: mean total reward, mean accuracy reward, and mean format compliance reward, representing overall quality, correctness, and adherence to the required format. The trends of these metrics are shown in Figure 4.

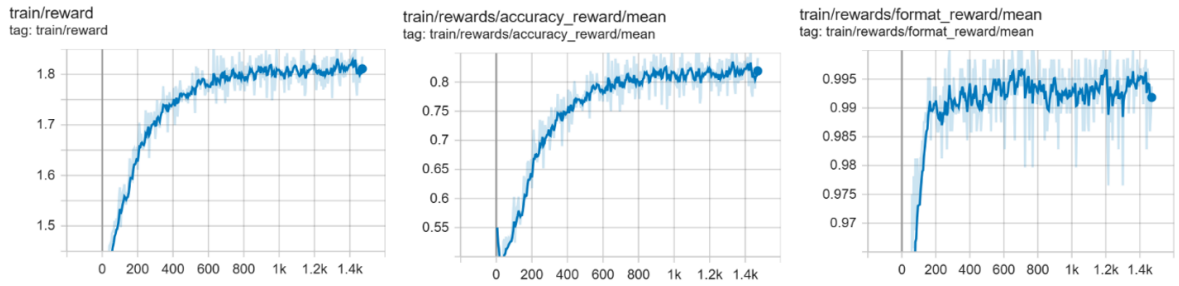

Figure 4: Training Rewards over Steps. Three panels show trends for total reward, accuracy reward, and format reward during GRPO fine tuning. Curves rise in the early phase and then stabilise with small fluctuations.

Figure 4 shows a rapid rise in the mean total reward from about 1.5 to above 1.8, then a plateau around 1,400 training steps. This indicates convergence and improved overall quality. The mean accuracy-focused reward increases from roughly 0.52 to around 0.82, with progress slowing after 600–800 steps, suggesting strong early gains that then stabilize. The mean format-compliance reward exceeds 0.99 within about 160 steps and remains between 0.99 and 0.996, indicating near-perfect adherence to the output rules. Taken together, the three curves show training that is effective, stable, and produces outputs that are accurate and compliant.

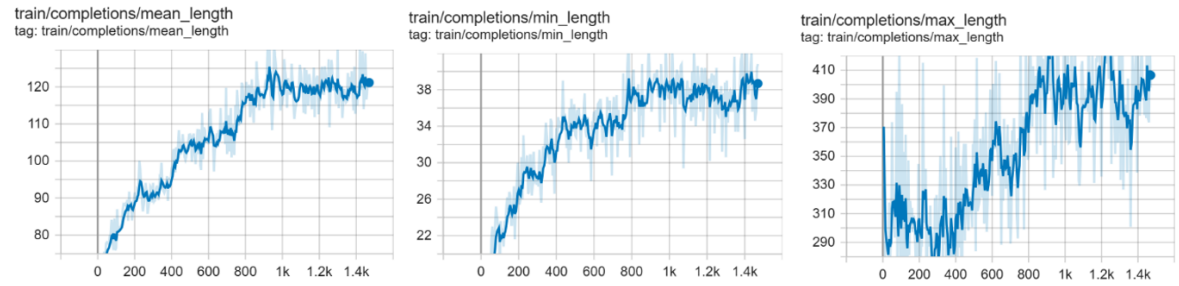

Figure 5: Completion Length over Training Steps. Three panels show mean minimum and maximum completion lengths during GRPO fine tuning. Lengths rise in early training then stabilise with small variations.

To complement the reward analysis, Figure 5 examines answer length. The length first increases and then stabilizes, with the average near 120 tokens. The minimum rises to about 38–40 tokens, meaning very short answers become less common. Some answers reach a maximum of about 400–410 tokens, which remains below the limit of 512 tokens. Since each update produces 512 completions, the theoretical maximum newly generated tokens per update is $512 \times 512 = 262{,}144$ tokens. With the observed average of about 120 tokens, actual usage per update is

around $512 \times 120 = 61{,}440$ tokens ($\approx 23\%$ of the limit). The system is therefore not length-bound, though the rising maximum should be monitored together with the clipped ratio to avoid truncation near the limit.

Evaluation on the physics dataset's test set validates performance on unseen data. The set comprises 199 items: 56 open-ended questions and 143 multiple-choice questions. Figure 6 displays a comparison between the base model (original Qwen) and the GRPO fine-tuned model.

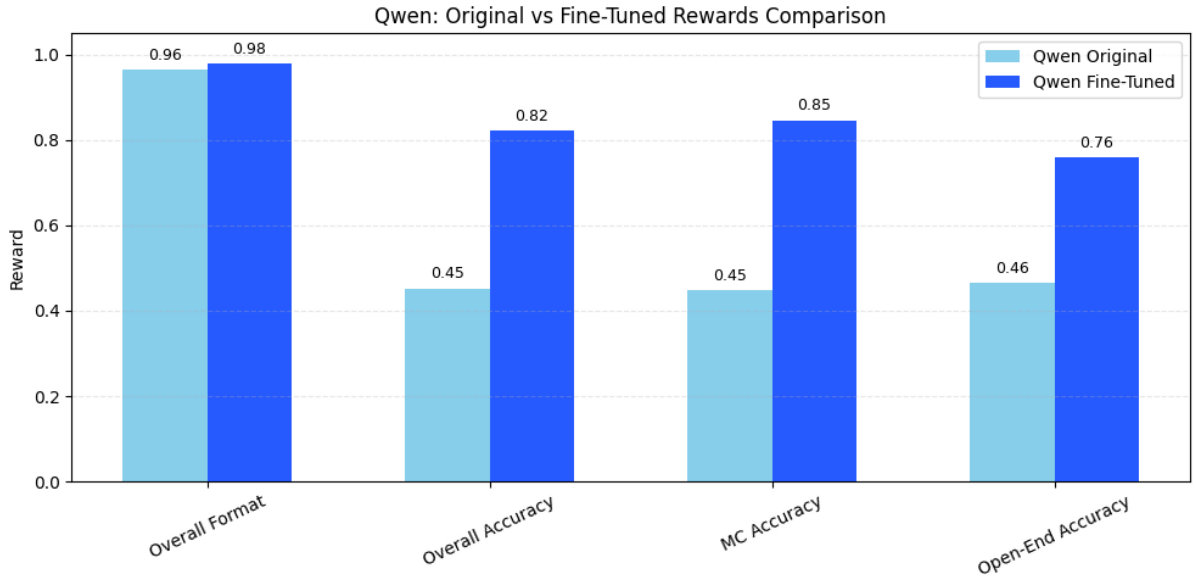

Figure 6: Base vs GRPO Fine Tuned on Physics Test Set. Bar chart comparing rewards for overall format, overall accuracy, multiple choice accuracy, and open end accuracy.

Figure 6 summarises differences between the base model and the GRPO fine-tuned model on the physics test set, with all scores measured on a 0–1 scale. Format compliance improves from 0.96 to 0.98, showing near-perfect consistency with the required structure. Overall accuracy rises from 0.45 to 0.82. For multiple-choice questions, accuracy increases from 0.45 to 0.85, while open-ended accuracy climbs from 0.46 to 0.76. These results indicate that GRPO fine-tuning greatly enhances accuracy across both question types while keeping format compliance at an excellent level, highlighting strong generalization to new test data.

A more detailed analysis considers the open-ended numerical questions in the test set. Figure 7 shows the comparison between the base model and the model fine-tuned with GRPO for this category.

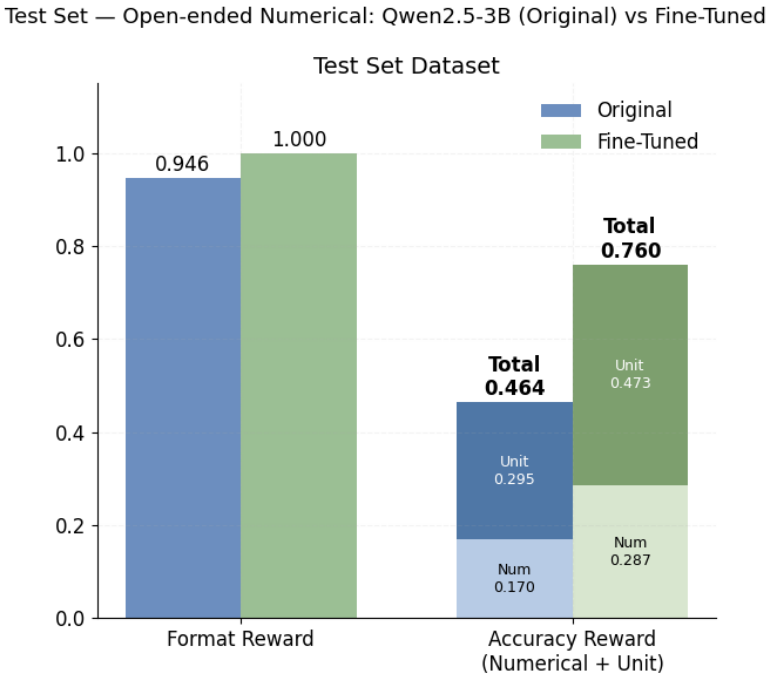

Figure 7: Open End Numerical Questions on the Test Set. Bar chart comparing the base model and the GRPO fine tuned model. The first group shows format reward. The second group shows accuracy reward split into numeric value and unit components.

Figure 7 reports test-set results for open-ended numerical questions. The format reward increases from 0.946 in the base model to 1.000 in the GRPO fine-tuned model, indicating complete template adherence. The total accuracy reward rises from 0.464 to 0.760. Decomposing the accuracy shows gains in both components: the unit score increases from 0.295 to 0.473, while the numerical score increases from 0.170 to 0.287. The unit component yields the larger absolute improvement, whereas the numerical component exhibits the larger proportional change. Overall performance indicates substantially higher accuracy with perfect formatting on this subset.

A scatter plot is employed to assess the discrepancy between ground-truth values and the model's predictions. By plotting each prediction against its reference value, the figure shows how closely the outputs match the correct numbers.

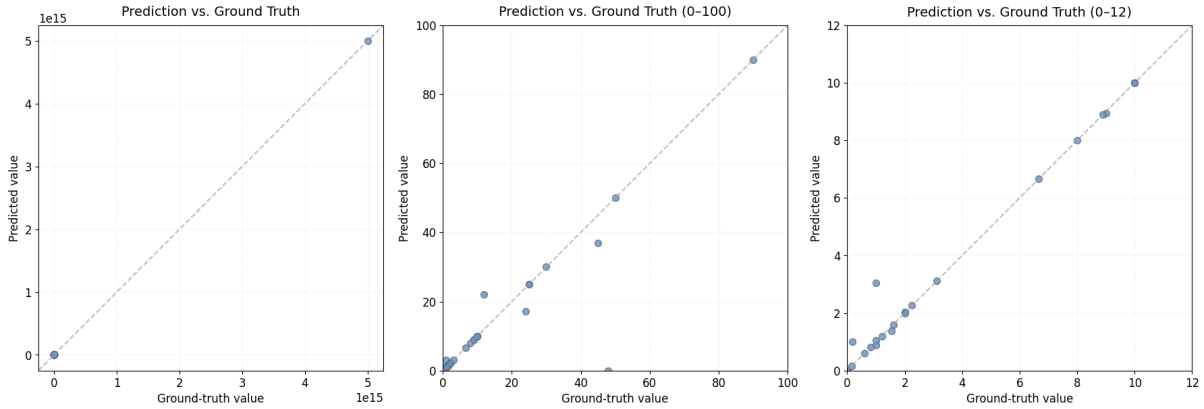

Figure 8: Prediction versus Ground Truth for Numerical Accuracy. Scatter plots for the fine tuned model across different value ranges. Points close to the diagonal line indicate strong agreement. Wider spread reflects larger error.

Figure 8 presents scatter plots for numerical accuracy. The dashed reference line $y = x$ marks perfect agreement between predicted and true values. Three panels are included because most ground-truth values fall in the 0–100 range. The left panel shows the full scale, covering values up to 10^{15}. The middle panel focuses on 0–100, the densest region. The right panel covers 0–12 to assess small numbers. Points cluster near the diagonal across panels, indicating strong agreement with ground truth. The full-scale view remains stable at extreme magnitudes with no clear systematic bias. In the 0–100 window, the scatter shows moderate spread with minor prediction deviations remaining. The 0–12 view forms a tight band along the diagonal, indicating high accuracy at low values. Overall numerical accuracy across ranges is strong.

To present model performance clearly, the comparison covers language models in the 1B to 8B range, including models for other languages. For fairness, all models use `max_prompt_length=256` for the input context. The fine tuned model uses `max_new_tokens=512`, which is enough for tasks that need short final answers such as a single option or a numeric value with a unit. The other baseline models use a wider ceiling of 32,768 to prevent truncation when long reasoning appears, so the final answer is not cut off. Scoring reads only the final answer and ignores extra reasoning text. A larger decoding ceiling does not increase the score. With equal input context across models, the evaluation remains directly comparable.

Table 10: Evaluation on the physics test set using the top 199 samples on a zero to one scale. Results include format reward accuracy reward multiple choice accuracy and open end accuracy for each model.

Model	Format Reward	Accuracy Reward	MC Accuracy Reward	Open-End Accuracy Reward (Total)
Qwen3-4B	0.0000	0.7644	0.7441	0.8162
gemma-3-4b-it	0.5477	0.6488	0.6259	0.7072
Qwen2.5-7B-Instruct	0.4070	0.5279	0.4706	0.6741
DeepSeek-R1-Distill-Qwen-1.5B	0.0000	0.2399	0.1629	0.4363
Qwen3-8B	0.0000	0.6690	0.6720	0.6612
DeepSeek-R1-Distill-Qwen-7B	0.0000	0.3868	0.3364	0.5158
DeepSeek-R1-Distill-Llama-8B	0.0000	0.4168	0.3217	0.6596
Phi-4-mini-instruct	0.1809	0.1800	0.0881	0.4148
Fined-tuning Qwen2.5-3B	0.9799	0.8220	0.8462	0.7602

Table 11: Open end numerical breakdown on a zero to one scale. Table lists component rewards for numeric value and unit across models.

Model	Open-End Numerical Reward	Open-End Unit Reward
Qwen3-4B [58]	0.3430	0.4732
gemma-3-4b-it [59]	0.2340	0.4732
Qwen2.5-7B-Instruct [60]	0.2187	0.4554
DeepSeek-R1-Distill-Qwen-1.5B [61]	0.0434	0.3929
Qwen3-8B [62]	0.2416	0.4196
DeepSeek-R1-Distill-Qwen-7B [63]	0.1051	0.4107
DeepSeek-R1-Distill-Llama-8B [64]	0.2132	0.4464
Phi-4-mini-instruct [65]	0.0487	0.3661
Fined-tuning Qwen2.5-3B	0.2870	0.4732

As shown in Tables 10 and 11, the fine-tuned Qwen2.5-3B model demonstrates strong performance on the physics test set with remarkable parameter efficiency compared to larger systems.

It achieves the highest Format Reward (0.9799) and ranks first in Accuracy Reward (0.8220) and MC Accuracy Reward (0.8462). The model's Open-End Accuracy (0.7602) is second only to Qwen3-4B (0.8162).

When compared with larger models, the improvements are substantial. Compared with Qwen2.5-7B Instruct, the Qwen2.5-3B model scores 0.2941 higher on Accuracy Reward, 0.3756 in MC Accuracy Reward, and 0.0861 in Open-End Accuracy. Relative to Qwen3-8B, the margins are 0.1530, 0.1742, and 0.0990 respectively.

The open-ended numerical evaluation (Table 11) further supports the advantage, with Qwen2.5-3B achieving an Open-End Numerical Reward of 0.2870, ranking second only to Qwen3-4B (0.3430). Its Open-End Unit Reward (0.4732) ties for the best score together with Qwen3-4B and *gemma-3-4b-it*.

Overall, the results indicate that the fine-tuned Qwen2.5-3B provides accuracy and formatting reliability comparable to or surpassing several 4B, 7B, and 8B alternatives, while requiring only 3B parameters. This makes it a strong candidate for deployment under limited compute budgets, as shown in Figs. 9 and 10.

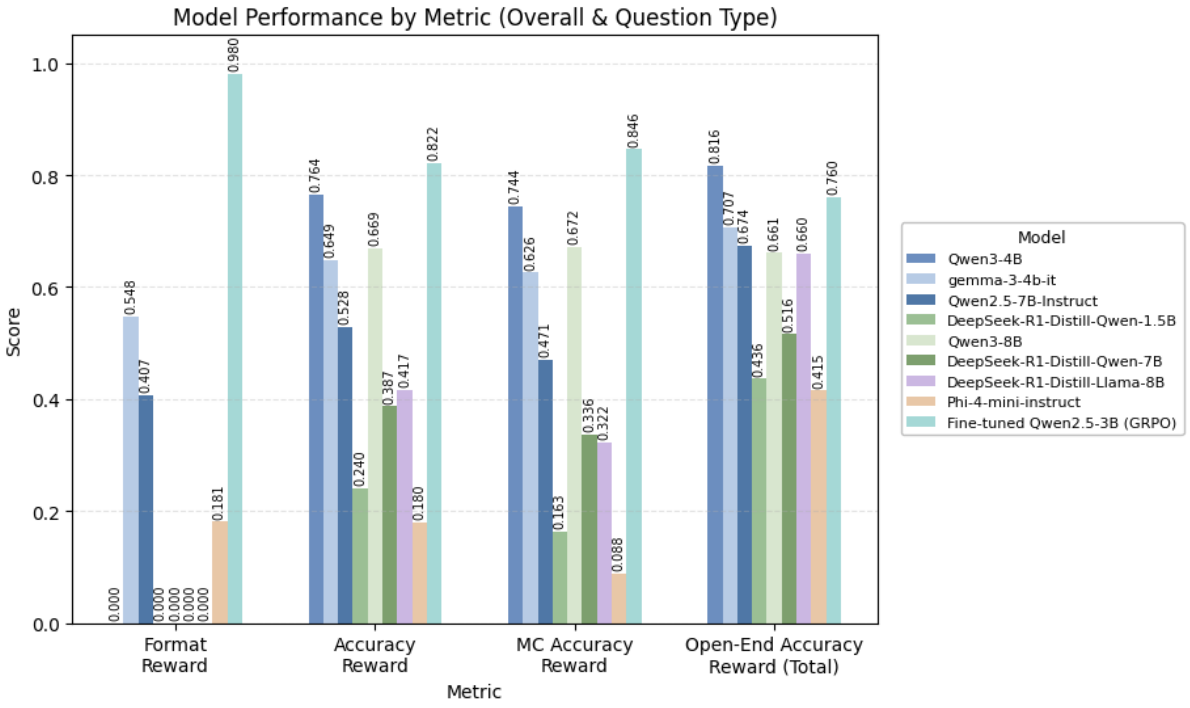

Figure 9: Model Performance by Metric (Overall & Question Type). Grouped bar chart comparing nine models across four metrics (Format Reward, Accuracy Reward, MC Accuracy Reward, and Open-End Accuracy Reward (Total)). The x-axis shows the metrics and the y-axis shows the score.

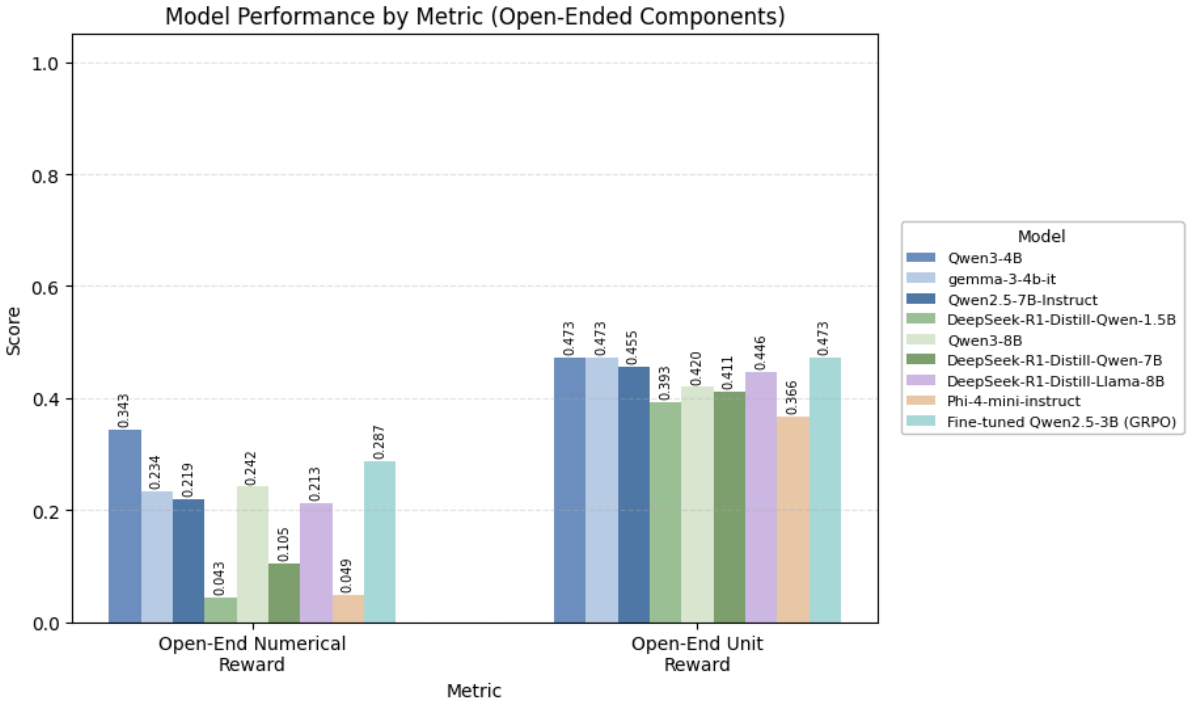

Figure 10: Model Performance by Metric (Open-Ended Components). Grouped bar chart for nine models on the two open-ended components (Open-End Numerical Reward and Open-End Unit Reward). The x-axis shows the metrics and the y-axis shows the score.

5 Discussion

The section reviews findings from GRPO fine tuning for small language models under hardware limits. Coverage includes selection of a suitable base model, presentation of new numerical scoring frameworks, and evaluation of robustness across multiple languages and datasets.

5.1 Resource Constraints on Base Model Selection

Base model selection determines pipeline feasibility including GPU memory use VRAM, context length, training stability, and token throughput. The choice directly affects answer quality and compute cost when considered with accuracy results. Due to resource limits, hyperparameters were not standardized across all models, potentially underrepresenting model specific capabilities. The selection for a 2×2080 Ti setup with GRPO training follows Table 12 and Table 13 and the reward results in Table 14.

Table 12: Resource footprint across base models during GRPO fine tuning. Table reports per GPU memory use `bnb_4bit` setting and wall clock duration in days.

Base Model	GPU1 Usage (MiB)	GPU2 Usage (MiB)	`bnb_4bit`	Duration (days)
Llama 3.2 1B Instruct	8500	8620	False	3.114
Llama 3.2 3B Instruct	10326	10618	False	1.957
Qwen 2.5 3B Instruct	10720	8796	False	4.127
Qwen 2.5 7B Instruct	8084	10958	True	2.957
Llama 3.1 8B Instruct	6350	10740	True	2.495

Table 13: Training configuration across base models with hyperparameters and context. Columns list LoRA r and α gradient accumulation per device batch maximum prompt and completion lengths and number of epochs.

Base Model	LoRA r	LoRA α	Grad. Accum.	Per-dev. Batch	Max Prompt	Max Completion	Epochs
Llama 3.2 1B Instruct	8	32	64	2	256	768	30
Llama 3.2 3B Instruct	8	32	32	4	164	256	30
Qwen 2.5 3B Instruct	8	32	64	2	256	512	30
Qwen 2.5 7B Instruct	8	32	64	2	172	256	24.38
Llama 3.1 8B Instruct	8	32	64	2	140	140	30

Based on Tables 12, 13 and the limits of a 2 × 2080 Ti setup for GRPO training, Qwen 2.5 3B was selected. It fits the available GPU memory (∼ 10.7 GiB on GPU1 and 8.8 GiB on GPU2) without 4-bit quantization, which supports stable and consistent training. The model allows a 256-token prompt and a 512-token completion, sufficient for step-by-step reasoning. Although the training time is longer at about 4.13 days, the trade-off is acceptable for greater stability and longer outputs. By comparison, Qwen 2.5 7B trains faster (about 2.96 days) but needs 4-bit quantization and has a shorter context (about 172/256 tokens), which can increase instability and truncation risk. Llama 3.2 3B is the fastest (about 1.96 days) but also has a short context (about 164/256 tokens), limiting extended reasoning. Llama 3.1 8B requires quantization to fit in memory, and Llama 3.2 1B, while easy to run and able to produce longer completions, is too small for complex reasoning tasks.

Table 14: Reward metrics before and after GRPO fine tuning across base models. Columns list base score fine tuned score and delta for format reward and accuracy reward.

Base model	Base	Fine tuned	Δ	Base	Fine tuned	Δ
	Format Reward			Accuracy Reward		
Llama 3.2 1B	0.0909	0.9697	+0.8788	0.0707	0.1616	+0.0909
Llama 3.2 3B	0.1313	0.7879	+0.6566	0.0505	0.2727	+0.2222
Qwen 2.5 3B	0.2828	0.9293	+0.6465	0.1717	0.2323	+0.0606
Qwen 2.5 7B	0.0101	0.2626	+0.2525	0.0606	0.1818	+0.1212
Llama 3.1 8B	0.0101	0.6061	+0.5960	0.0202	0.0909	+0.0707

Table 14 shows fine-tuning model achieved the highest combined reward (Format 0.9293, Accuracy 0.2323, Total 1.1616), ahead of Llama 3.2 1B (1.1313) and Llama 3.2 3B (1.0606), with the 7B and 8B models scoring lower. Overall, Qwen 2.5 3B provides the best balance of compute feasibility, stability without quantization, adequate context length, and answer quality, aligning with the goal of producing clear and well-structured explanations under tight hardware constraints.

5.2 Scoring Framework for Numerical Answers and Penalty Mapping

This section discusses criteria for constructing the accuracy-penalty function and proposes Percentage Squared Error (PSE) instead of APE. PSE grows smoothly and assigns larger penalties to larger mistakes while remaining scale-proportional (for example, 1/10 and 10/100 represent the same relative error). For numerical stability, the denominator is regularized by a small constant $\delta = 1 \times 10^{-32}$ to handle cases where the reference value approaches zero, i.e.,

$$\text{PSE} = \left(\frac{\hat{y} - y}{|y| + \delta} \right)^2 \tag{31}$$

where $\hat{y}$ denotes the predicted value and y the reference value.

After computing PSE, map it to a penalty with an upper bound of 0.5. This allows adding the unit score of 0.5 so the total stays within [0, 1]. The penalty coefficient (*coef*) sets sensitivity to error. A higher *coef* makes the penalty increase faster, even for small errors. Two mappings are used, linear clipped and sigmoid based. Both return values in [0, 0.5], but the behavior near the ceiling differs. The linear-clipped version grows proportionally with PSE and then saturates at 0.5, which is easy to interpret and limits the impact of extreme outliers because the penalty does not increase beyond the cap. The closed-form used in the main reward variant is

Linear clipped penalty.

$$\text{Penalty}_{\text{lin}} = \text{clip}\big(\text{coef} \cdot \text{PSE},\, 0,\, 0.5\big), \tag{32}$$

Used in the primary reward setting, this closed-form mapping grows linearly and then clips at 0.5, so outliers are bounded once the ceiling is reached.

With *coef*=5,000, the penalty reaches the 0.5 cap once the relative error is about 1% or higher, as shown by the equation below.

$$\text{PSE} = \frac{0.5}{5000} = 1.0 \times 10^{-4} \;\Rightarrow\; \left| \frac{\hat{y} - y}{y} \right| \approx 1\%. \tag{33}$$

With coef = 10000,

$$\text{PSE} = \frac{0.5}{10000} = 5.0 \times 10^{-5} \;\Rightarrow\; \left|\frac{\hat{y} - y}{y}\right| \approx 0.707\%. \tag{34}$$

Thus the 10k setting is stricter, assigning the full penalty at a smaller error. the penalty reaches the 0.5 cap when the relative error is about 0.7%.

In contrast to hard clipping, the sigmoid-based approach transforms the PSE with a sigmoid and aligns it to the same range, yielding a continuously differentiable curve that approaches the ceiling smoothly and maintains greater resolution among near-correct answers.

Sigmoid-based penalty.

$$\text{Penalty}_\sigma = \sigma(\mathit{coef} \cdot \text{PSE}) - 0.5, \qquad \sigma(x) = \frac{1}{1 + e^{-x}}. \tag{35}$$

This mapping produces a smooth curve with no hard corner at the cap, preserving resolution among near-correct answers. For *coef*= 50,000, the penalty is ≈ 0.25 at

$$\text{PSE} \approx \frac{\ln 3}{50{,}000} \approx 2.20 \times 10^{-5} \;\Rightarrow\; \left|\frac{\hat{y} - y}{y}\right| \approx 0.47\%, \tag{36}$$

and it rises to ≈ 0.49 near

$$\text{PSE} \approx \frac{\ln 99}{50{,}000} \approx 9.19 \times 10^{-5} \;\Rightarrow\; \left|\frac{\hat{y} - y}{y}\right| \approx 0.96\%. \tag{37}$$

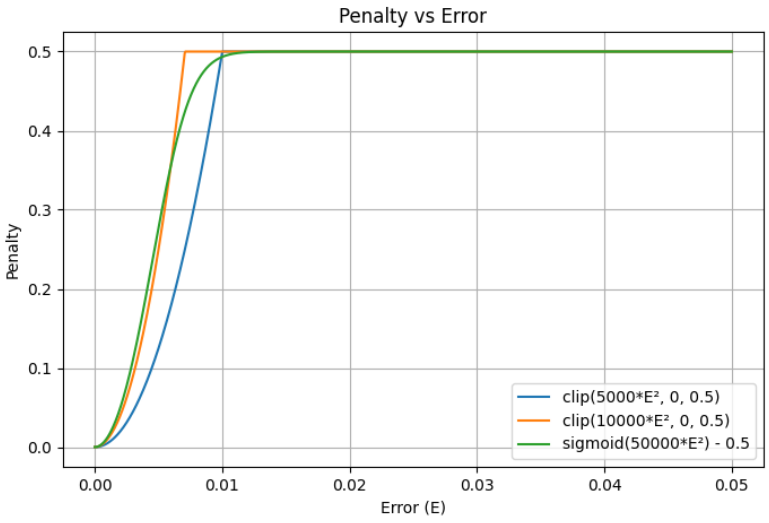

Figure 11: Penalty Functions versus Relative Error. Line plot comparing three penalty designs based on squared error with clipping or a sigmoid cap. Curves rise quickly at small error and then level off near a common maximum.

Figure 11 plots penalty versus error for three mappings: two quadratic clips (coefficients 5,000 and 10,000) and a sigmoid (coefficient 50,000). Larger coefficients make the response steeper; the 10k clip reaches the 0.5 cap at a smaller error ($\sim 0.71\%$) than the 5k clip ($\sim 1.0\%$). Clip

curves stop instantly at the cap (errors larger than the threshold receive the same penalty), whereas the sigmoid approaches the cap smoothly, maintaining rank order near the top.

To enable a fair, apples-to-apples comparison with the APE baseline, six fine-tuned checkpoints were evaluated on the same test split with the same prompt template. For clarity, each checkpoint differs only in the numeric-penalty mapping used by the reward function during GRPO training:

- `Qwen2.5-3B-GRPO-Physics-2-5000coeff` : employs a linear clipped penalty with a coefficient of 5,000
- `Qwen2.5-3B-GRPO-Physics-2-10000coeff` : applies a linear clipped penalty with a coefficient of 10,000
- `Qwen2.5-3B-GRPO-Physics-2-15000coeff` : adopts a linear clipped penalty with a coefficient of 15,000
- `Qwen2.5-3B-GRPO-Physics-2-50000coeff-sigmoid` : implements a sigmoid-based penalty with a coefficient of 50,000
- `Qwen2.5-3B-GRPO-Physics-2-100000coeff-sigmoid` : uses a sigmoid-based penalty with a coefficient of 100,000
- `Qwen2.5-3B-GRPO-Physics-2-150000coeff-sigmoid` : relies on a sigmoid-based penalty with a coefficient of 150,000

The evaluator extracted the span inside `<answer>...</answer>`, checked dimensional equivalence of units with the `pint` registry, and computed the open-ended accuracy reward using a PSE-based pipeline. For reporting, three canonical views are used—`accuracy_reward_5k`, `accuracy_reward_10k`, and `accuracy_reward_50k_sigmoid`—which correspond to the 5k/10k linear-clipped and 50k sigmoid mappings. Scoring uses two components: the unit component gives 0.5 and the numeric component starts at 0.5.

Table 15: Open end accuracy reward under different penalty settings across models. Settings include linear 5k and 10k coefficients and a 50k coefficient with sigmoid.

Model	Open-end Accuracy Reward (5k_coeff)	Open-end Accuracy Reward (10k_coeff)	Open-end Accuracy Reward (50k_coeff_sigmoid)
`Qwen2.5-3B-GRPO-Physics-2-5000coeff`	0.5625	0.5625	0.5625
`Qwen2.5-3B-GRPO-Physics-2-10000coeff`	0.5892	0.5891	0.5890
`Qwen2.5-3B-GRPO-Physics-2-15000coeff`	0.5895	0.5893	0.5894
`Qwen2.5-3B-GRPO-Physics-2-50000coeff-sigmoid`	0.5771	0.5738	0.5738
`Qwen2.5-3B-GRPO-Physics-2-100000coeff-sigmoid`	0.5772	0.5740	0.5740
`Qwen2.5-3B-GRPO-Physics-2-150000coeff-sigmoid`	0.6002	0.5980	0.5984

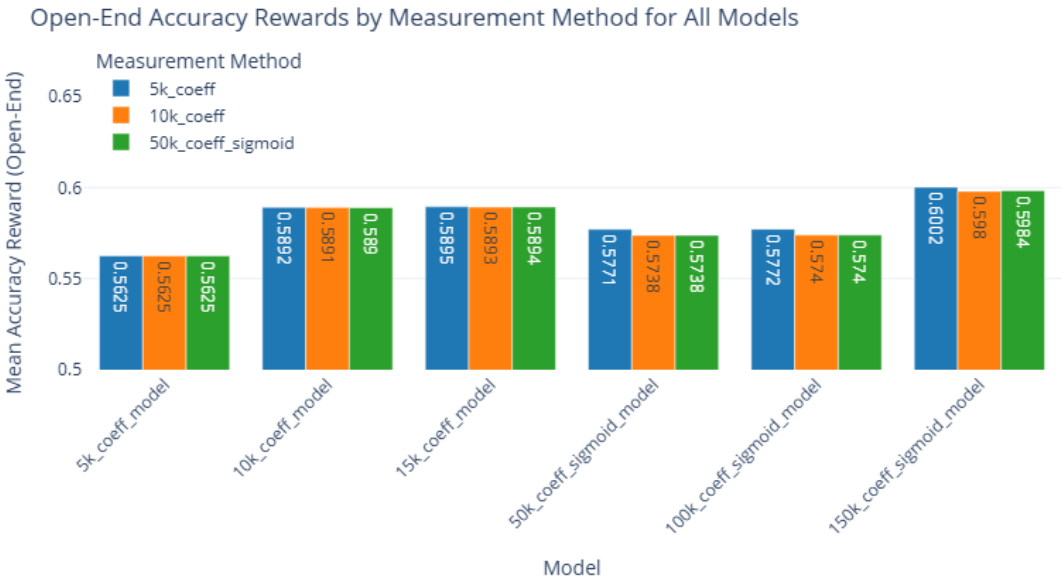

Figure 12: Mean Open End Accuracy by Penalty Setting. Bar chart comparing measurement methods 5k coefficient 10k coefficient and 50k coefficient with sigmoid across all models. Bars show mean open end accuracy.

As seen in Table 15 and Figure 12, the three measurements have almost identical mean scores for each model with a maximum difference of about 0.0033, which indicates that the choice between

clipping and sigmoid has only a small effect when the checkpoint is fixed. Differences across models are much larger: the 150k_coeff_sigmoid checkpoint records the highest mean, about 0.60; the 5k_coeff_linear model is lowest, about 0.5625; the 10k and 15k linear checkpoints cluster near 0.589; and the 50k to 100k sigmoid group ranges from 0.574 to 0.577. Overall, performance is driven mainly by the trained checkpoint, while the evaluation-time penalty mapping mostly rescales errors under the common 0.5 cap together with the shared unit and significant-figure rules.

APE-based penalty was selected as the final training objective after controlled comparisons with PSE-based penalties, both linear clipped and sigmoid, on the same test set and prompt template. APE yielded higher mean open-end accuracy across all checkpoints. Under APE, fine-tuned models reached about 0.76 total accuracy (Figure 7). Under PSE, the best mean for the same models was about 0.60 (Table 15 and Figure 12). The advantage appeared for every checkpoint and indicated a material impact of penalty-function shape on the average score. APE applies a gentle penalty that levels off. A 1% error receives about 0.10, improving treatment of small errors. In the clipped variant, PSE imposes the full 0.5 penalty even for a 1% error, causing score compression. Test results indicate stronger performance under APE, leading to selection of APE.

5.3 Robustness on Open-Ended Number Questions in Diverse Datasets

Evaluation of model robustness to numerical questions with varied formats was carried out by compiling a supplementary dataset alongside the main Hugging Face test set. Sources were Chinese high-school physics textbooks [66], with an English translation created to examine language-related limitations. Cleaning and filtering kept only open-ended questions whose answers are numeric with units (e.g., $12\,\mathrm{cm}^3$, $33.3\,\mathrm{km/h}$). Standardization was applied to reduce formatting effects unrelated to numeric correctness: unify multiplication symbols and decimal points, remove thousand separators, standardize units, and exclude items with approximation markers or extra qualifiers. After processing, the Chinese set contained 76 items, and the translated English set contained 67 items. Both sets used the same prompt template as the main test set, and answers were placed inside `<answer>` tags, enabling direct comparison of paraphrase scores. Scoring follows the same framework. The total score equals the sum of a unit score and a numeric score. The unit score is 0.5 when unit dimensions match. The numeric score starts at 0.5 and is reduced by a penalty computed by the specified function under the APE framework. The total score is stably bounded within $[0, 1]$.

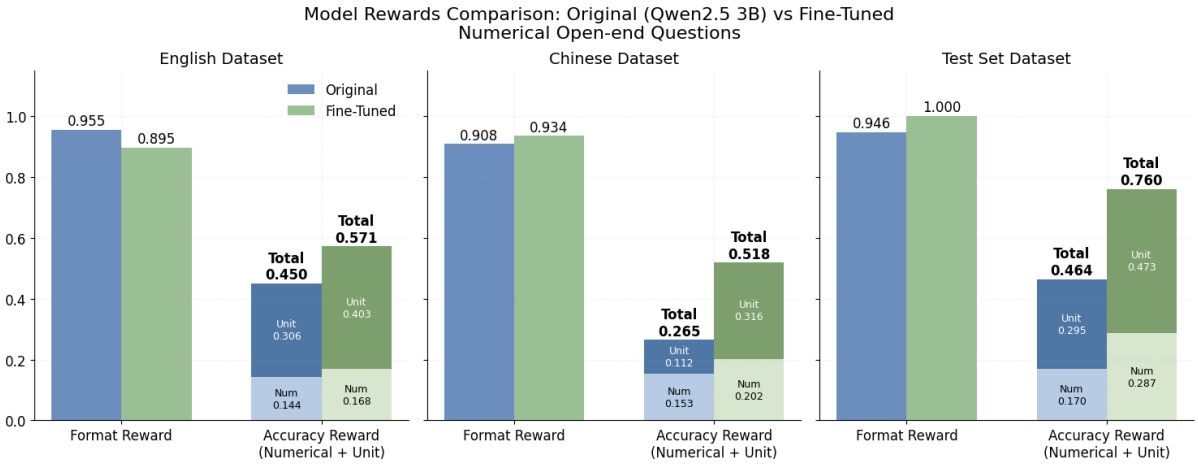

Figure 13: Robustness on Open Ended Numerical Questions across English Chinese and the Main Test Set. Three panel bar chart comparing the base model and the GRPO fine tuned model. Left panel shows the English subset. Middle panel shows the Chinese subset. Right panel shows the main test set. Each panel includes format reward and accuracy reward split into numeric and unit components.

Figure 13 presents a comparison between the original model and the fine-tuned model using Hugging Face data, tested on open-ended numerical questions in three datasets: English, Chinese, and a main test set derived from the train and test split. Fine-tuning increased Accuracy Reward in every dataset. English rose from 0.450 to 0.571, a gain of 0.121, with the unit component rising from 0.306 to 0.403 and the numeric component from 0.144 to 0.168. Chinese rose from 0.265 to 0.518, a gain of 0.253, with the unit component rising from 0.112 to 0.316 and the numeric component from 0.153 to 0.202. The main test set rose from 0.464 to 0.760, a gain of 0.296, with the unit component rising from 0.295 to 0.473 and the numeric component from 0.170 to 0.287. The English and Chinese datasets were used for testing only, without any training or fine-tuning on those data. Although the English set was translated and slightly smaller than the Chinese dataset, performance in English exceeded performance in Chinese, indicating stronger answering ability in English. For Chinese questions, generated answers appear in Chinese. Format Reward remained high overall, improving in Chinese from 0.908 to 0.934 and in the main test set from 0.946 to 1.000, with a small decrease in English from 0.955 to 0.895 caused by some outputs failing to close template tags under long reasoning. Overall results show clearer unit handling, better numeric proximity to ground truth, and greater robustness to dataset diversity after fine-tuning.

Following Figure 13, paired scatterplots illustrate the relation between model predictions and ground-truth values. The diagonal dashed line indicates ideal agreement where the predicted value equals the reference. Point proximity to this line signals higher accuracy, while larger offsets indicate error. The top row reports the Chinese dataset in Figure 14. The bottom row reports the English dataset in Figure 15. Each row presents one full-range overview panel and two zoomed panels covering 0 to 100 and 0 to 12.

Chinese Dataset.

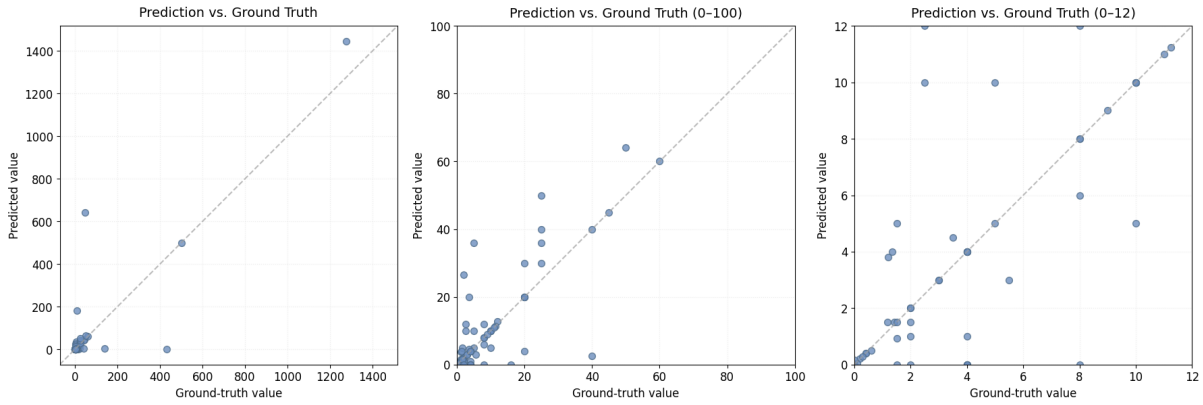

Figure 14: Chinese Subset Prediction vs Ground Truth. Scatter plots cover the full value range with zoomed views for 0 to 100 and 0 to 12. The diagonal line marks ideal agreement.

Figure 14 displays a dense cluster near the origin, indicating that most items have small numerical answers. Deviation from the ideal diagonal increases at medium to high values, pointing to scale mismatches or unit-conversion errors in a subset of items. The 0–100 zoom reveals variance that grows with the ground truth value, with both overestimation and underestimation. The 0–12 zoom shows points closely aligned with the ideal line, and remaining errors are small in absolute terms because the vertical scale is low. Overall, accuracy is strong in the low-value region that dominates the tasks, while most observed errors arise at larger true values or when multiple unit conversions are involved.

English Dataset.

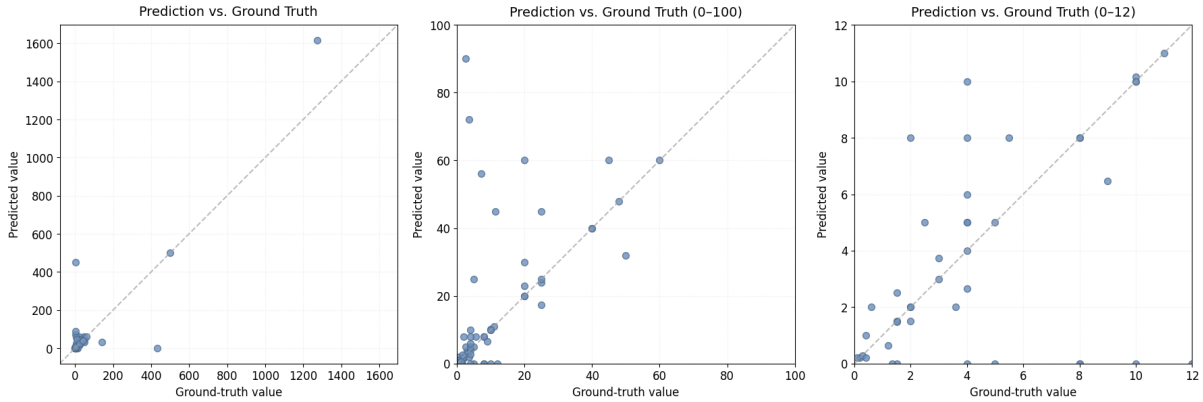

Figure 15: English Subset Prediction vs Ground Truth. Scatter plots display the full range with zoomed views for 0 to 100 and 0 to 12. The diagonal line marks ideal agreement.

Figure 15 presents a three-panel scatter plot comparing predicted with true values, with the diagonal dashed line indicating ideal agreement. Compared with the previous figure, adherence to the ideal line improves at low values, especially in the right panel covering 0 to 12 where most points lie close to the line and absolute deviations remain very small due to the narrow vertical scale. The middle panel for 0 to 100 shows tighter clustering and fewer overestimates or underestimates up to about 20. A visible spread persists in the mid-range from 30 to 60, and the left panel over the full range contains a few distant outliers at very high true values. Overall performance is stronger at low values and in the early part of 0 to 100, while the mid to high ranges remain the main area for further improvement.

6 Conclusion

The study integrates the GRPO framework with a structured reasoning template and low rank adaptation fine tuning LoRA to improve numerical accuracy and output format consistency under limited resources. Evaluation on 199 questions 56 open ended and 143 multiple choice shows clear gains. Overall accuracy rose from 0.45 to 0.82. Multiple choice accuracy rose from 0.45 to 0.85. Open ended accuracy rose from 0.46 to 0.76. Format adherence reached about 0.98. GRPO fine tuning therefore supports near perfect format control together with higher correctness on unseen data. Furthermore, cross model comparison identified Qwen 2.5 3B fine tuned as the top system for Format Reward 0.9799 Accuracy Reward 0.8220 and MC Accuracy 0.8462. Open ended accuracy 0.7602 placed second to Qwen3 4B. Results highlight strong quality per resource of the 3B model against larger models. In addition, Behaviour analysis shows higher correctness for small target values. Larger errors appear for middle to high targets. The trend holds for Chinese and English subsets and for the main set. For cross linguistic robustness on numerical questions the fine tuned model improves on every dataset and performs better in English than in Chinese even when the English set comes from translation. Format reward remains very high with a small drop in English due to occasional unclosed tags during long explanations. Moreover, APE based penalty uses a saturating forgiving curve for small deviations and delivers better outcomes than PSE based penalty with linear cutoff or sigmoid. Under APE open ended accuracy reaches about 0.76 compared with a PSE mean near 0.60 at the same checkpoint. The scoring scheme separates unit and numeric parts clearly unit 0.5 plus numeric 0.5 enabling fair and interpretable evaluation on the 0 to 1 scale.

Key constraints arise from non standardised settings across models and limited compute 2×RTX 2080 Ti. The setup may cap the potential of certain models and requires longer training for stability and sufficient context for step by step reasoning. Dataset imbalance also matters with about 29 percent open ended and 71 percent multiple choice and with Chain of Thought examples near 61 percent affecting coverage of reasoning skills and error patterns. Incomplete closing tags appear in long explanations and slightly reduce format scores in some languages.

Future work includes data balancing with more open ended items requiring multi stage unit conversion to reduce blind spots in middle to high value ranges and to strengthen cross language robustness. Larger training sets are also needed because only around 2000 examples were available due to licensing limits. Finally, Multilingual fine tuning plus standard rules for units and numeric notation will support quality across domains and languages.

References

[1] W. X. Zhao, K. Zhou, *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2025. Version v16, 11 Mar 2025; primary class cs.CL.

[2] K. T. Chitty-Venkata, M. Emani, *et al.*, "Neural architecture search for transformers: A survey," *IEEE Access*, 2022. Published 6 Oct 2022; current version 17 Oct 2022.

[3] P. Xu, X. Zhu, *et al.*, "Multimodal learning with transformers: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. Vol. 45, No. 10, Oct 2023; published 11 May 2023; current version 5 Sep 2023.

[4] K. Han, Y. Wang, *et al.*, "A survey on visual transformer," *arXiv preprint arXiv:2012.12556*, 2023. Version v6, 10 Jul 2023; submission to IEEE TPAMI; primary class cs.CV.

[5] A. Vaswani, N. Shazeer, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, (Long Beach, CA, USA), Curran Associates, Inc., 2017.

[6] M. Artetxe, J. Du, *et al.*, "On the role of bidirectionality in language model pre-training," *arXiv preprint arXiv:2205.11726*, 2022.

[7] J. Devlin, M.-W. Chang, *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2019.

[8] C. Raffel, N. Shazeer, *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[9] T. B. Brown, B. Mann, *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[10] P. Liu, W. Yuan, *et al.*, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *arXiv preprint arXiv:2107.13586*, 2021.

[11] H. Touvron, L. Martin, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[12] N. Stiennon, L. Ouyang, *et al.*, "Learning to summarize from human feedback," *arXiv preprint arXiv:2009.01325*, 2020.

[13] Z. Zhou, X. Ning, *et al.*, "A survey on efficient inference for large language models," *arXiv preprint arXiv:2404.14294*, 2024.

[14] L. Ouyang, J. Wu, *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, vol. 35, 2022. NeurIPS 2022.

[15] J. Hu, X. Wu, *et al.*, "Openrlhf: An easy-to-use, scalable and high-performance rlhf framework," *arXiv preprint arXiv:2405.11143*, 2024.

[16] J. Wang, W. Shao, *et al.*, "Adapting llama decoder to vision transformer," *arXiv preprint arXiv:2404.06773*, 2024.

[17] Qwen Team, "Qwen2.5 technical report," *arXiv preprint arXiv:2412.15115*, 2025.

[18] Z. Shao, P. Wang, *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.

[19] DeepSeek-AI, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[20] J. Wei, X. Wang, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *arXiv preprint arXiv:2201.11903*, 2022.

[21] T. Kojima, S. S. Gu, *et al.*, "Large language models are zero-shot reasoners," *arXiv preprint arXiv:2205.11916*, 2022.

[22] A. Chowdhery, S. Narang, *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[23] P. Shaw, J. Uszkoreit, *et al.*, "Self-attention with relative position representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, (New Orleans, Louisiana), pp. 464–468, Association for Computational Linguistics, 2018.

[24] K. He, X. Zhang, *et al.*, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[25] J. L. Ba, J. R. Kiros, *et al.*, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[26] I. Tenney, D. Das, *et al.*, "Bert rediscovers the classical nlp pipeline," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, (Florence, Italy), pp. 4593–4601, Association for Computational Linguistics, 2019.

[27] G. Jawahar, B. Sagot, *et al.*, "What does bert learn about the structure of language?," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, (Florence, Italy), pp. 3651–3657, Association for Computational Linguistics, 2019.

[28] Y. Bengio, R. Ducharme, *et al.*, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[29] J. Kaplan, S. McCandlish, *et al.*, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[30] H. F. T. Team, "Grpo trainer — trl documentation," 2025. Online documentation; main branch; latest stable v0.22.1; accessed 2025-09-03.

[31] W. Chung, V. Thomas, *et al.*, "Beyond variance reduction: Understanding the true impact of baselines on policy optimization," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, vol. 139 of *Proceedings of Machine Learning Research*, PMLR, 2021. ICML 2021; PMLR Vol. 139.

[32] Q. Yu, Z. Zhang, *et al.*, "Dapo: An open-source llm reinforcement learning system at scale," 2025. Technical report; Date: 17 Mar 2025.

[33] Meta, "Llama-3.2-1b-instruct — model card," 2024. Hugging Face model card; model release date 25 Sep 2024; license: Llama 3.2 Community License; accessed 2025-09-03.

[34] Meta, "meta-llama/llama-3.2-1b — model card," 2024. Model card; published Sep 25, 2024; accessed Sep 2, 2025.

[35] Meta, "meta-llama/llama-3.2-3b — model card," 2024. Model card; published Sep 25, 2024; accessed Sep 2, 2025.

[36] Meta, "Llama 3.2 model cards and prompt formats." Online documentation; accessed Sep 2, 2025.

[37] M. AI, "Introducing quantized llama models with increased speed and reduced memory," 2024. Blog post; Oct 24, 2024; accessed Sep 2, 2025.

[38] N. NGC, "Meta llama 3.2 3b instruct onnx int4 rtx — model card," 2024. Model card; modified Nov 15, 2024; accessed Sep 2, 2025.

[39] Q. Team, "Qwen2.5: A party of foundation models!," 2024. Blog post; 19 Sep 2024; accessed 2025-09-03.

[40] Qwen, "Qwen/qwen2.5-3b — model card," 2025. Model card; published Jul 21, 2025; accessed Sep 2, 2025.

[41] Qwen, "Qwen/qwen2.5-7b — model card," 2025. Model card; published Jul 21, 2025; accessed Sep 2, 2025.

[42] Q. Team, "Qwen2.5-llm: Extending the boundary of llms," 2024. Blog post; Nov 22, 2024; accessed Sep 2, 2025.

[43] Meta, "meta-llama/llama-3.1-8b — model card," 2024. Model card; published Jul 23, 2024; accessed Sep 2, 2025.

[44] H. Face, "Llama 3.1 — 405b, 70b & 8b with multilinguality and long context," 2024. Blog post; Jul 23, 2024; accessed Sep 2, 2025.

[45] E. Hu, Y. Shen, *et al.*, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021. Version v2, 16 Oct 2021; primary class cs.CL.

[46] D. Biderman, J. G. Ortiz, *et al.*, "Lora learns less and forgets less," *arXiv preprint arXiv:2405.09673*, 2024. Version v1, 15 May 2024; primary class cs.LG.

[47] AI-MO, "Numinamath-tir," 2025. Dataset; online; accessed Aug 15, 2025.

[48] Hugging Face, "Fine-tuning llms with grpo (trl) — cookbook," 2025. Online tutorial; accessed Aug 15, 2025.

[49] cais, "Mmlu (high_school_physics subset)," 2025. Dataset; online; accessed Aug 17, 2025.

[50] derek thomas, "Scienceqa," 2025. Dataset; online; accessed Aug 17, 2025.

[51] mrohith29, "high-school-physics," 2025. Dataset; online; accessed Aug 17, 2025.

[52] cristiano sartori, "conceptual_physics," 2025. Dataset; online; accessed Aug 17, 2025.

[53] qgallouedec, "physics-problems," 2025. Dataset; online; accessed Aug 17, 2025.

[54] BaunRobotics, "physics," 2025. Dataset; online; accessed Aug 17, 2025.

[55] nathannarrik, "physics_tutor," 2025. Dataset; online; accessed Aug 17, 2025.

[56] psamtam, "high_school_physics_dataset," 2025. Dataset; online; accessed Aug 17, 2025.

[57] Ppear, "Qwen2.5 3b grpo physics_expo," 2025. Model repository; online; accessed Aug 23, 2025.

[58] Q. Team, "Qwen3-4b." `https://huggingface.co/Qwen/Qwen3-4B`, 2025. Hugging Face model card; accessed 2025-09-04.

[59] Google, "Gemma 3 4b it." https://huggingface.co/google/gemma-3-4b-it, 2025. Hugging Face model card; accessed 2025-09-04.

[60] Q. Team, "Qwen2.5-7b-instruct." https://huggingface.co/Qwen/Qwen2.5-7B-Instruct, 2025. Hugging Face model card; accessed 2025-09-04.

[61] DeepSeek-AI, "Deepseek-r1-distill-qwen-1.5b." https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B, 2025. Hugging Face model card; accessed 2025-09-04.

[62] Q. Team, "Qwen3-8b." https://huggingface.co/Qwen/Qwen3-8B, 2025. Hugging Face model card; accessed 2025-09-04.

[63] DeepSeek-AI, "Deepseek-r1-distill-qwen-7b." https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B, 2025. Hugging Face model card; accessed 2025-09-04.

[64] DeepSeek-AI, "Deepseek-r1-distill-llama-8b." https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B, 2025. Hugging Face model card; accessed 2025-09-04.

[65] Microsoft, "Phi-4-mini-instruct." https://huggingface.co/microsoft/Phi-4-mini-instruct, 2025. Hugging Face model card; accessed 2025-09-04.

[66] P. Qiancheng and S. Huang, eds., *Senior High School Textbook: Physics (Selective Compulsory, Book 3)*. Beijing, China: People's Education Press, 2020. [in Chinese].

Appendix

A Applying GRPO in TRL for Post-Training a Large Language Model on Reasoning Tasks

This appendix describes the training pipeline and GRPO settings used for post-training a small mathematical reasoning model. The process follows the public TRL GRPO workflow, and the conceptual steps are consistent with the official documentation and examples in the cookbook [48].

A.1 Required Environment

For fine-tuning the model, it is necessary to have several key libraries: `CUDA`, `PyTorch`, `transformers`, `trl`, `accelerate`, `bitsandbytes`, `peft`, and `math-verify`.

A.2 Data and Prompt Formatting

The AI-MO/NuminaMath-TIR dataset was used for evaluation-driven post-training. All prompts are reformatted into an instruction style with a system prompt that guides the model to analyze the problem before giving its final answer. The reasoning process and the final answer are enclosed in `<think>` and `<answer>` tags, respectively.

Table 16: System prompt and conversation template for evaluation driven post training. Table lists the instruction text with `<think>` and `<answer>` tags the message roles used in the prompt and the target completion format.

```
SYSTEM_PROMPT = (
  "A conversation between User and Assistant.  The user asks a
question, and the Assistant
  solves it.  The Assistant first thinks about the reasoning process in
the mind and then
  provides the user with the answer.  The reasoning process and the
final answer are enclosed
  within <think>...</think> and <answer>...</answer> tags,
respectively."
)

prompt = [
  {"role":  "system", "content":  SYSTEM_PROMPT},
  {"role":  "user", "content":  example["problem"]},
]

target_completion = (
  f"<think>{example['rationale']}</think>"
  f"<answer>{example['final_answer']}</answer>"
)
```

A.3 Base Model and LoRA Configuration

The initial policy model used is `Qwen/Qwen2-0.5B-Instruct`, which is fine-tuned with LoRA. The settings are: $r = 8$, `lora_alpha` = 32, `lora_dropout` = 0.1, and `target_modules` = ["q_proj", "v_proj"]. This configuration results in only 540,672 trainable parameters out of a total of 494,573,440 parameters (∼0.1093%). As a result, the training process is more efficient in terms of both time and memory usage.

A.4 Reward Design and GRPO Update

The scalar reward consists of two components:

1. **Answer correctness** — determines correctness of the final answer. The first mathematical expression from the model output and the gold answer is extracted using a LaTeX parser. A verifier compares them, treating equivalent forms as the same (e.g., 1/2 and 0.5). A score of 1.0 is assigned for a match, and 0.0 for a mismatch or parsing error. If the gold answer cannot be parsed, a score of 1.0 is given to avoid penalising noisy data.
2. **Format reward** — encourages a step-by-step structure in `<think>` and concise output in `<answer>`.

GRPO updates the policy by increasing the relative advantage of higher-reward responses within a group while constraining divergence from a reference policy.

A.5 Key Hyperparameters

The GRPO configuration includes specific settings such as: the number of responses generated per prompt, the group size for calculating advantages, and the number of gradient accumulation steps. The specific values used in each run are presented in Table 17. Training proceeds until the prescribed number of epochs is completed.

Table 17: Main training parameters for the GRPO setup. Entries include learning rate gradient accumulation steps bf16 use maximum completion length number of generations and maximum prompt length.

Hyperparameter	Value
learning_rate	1e-5
gradient_accumulation_steps	16
bf16	True
max_completion_length	64
num_generations	4
max_prompt_length	128