Master's Thesis

Learning from Within: Hidden-State Dynamics as Rewards for Training LLMs

Department of Computer Science
Ludwig-Maximilians-Universität München

Ilir Hajrullahu
Matriculation number: 12802929

Munich, September 28th, 2025

Submitted in partial fulfillment of the requirements for the degree of M. Sc.
Supervised by Dr. Yunpu Ma

Acknowledgments

I would like to sincerely thank my supervisor, Dr.Yunpu Ma, for the possibility to write my thesis at the TRESP Lab, his constructive feedback, and invaluable support throughout the course of this thesis. His expertise and encouragement have been crucial in shaping both the direction and the quality of my research.
I am also very grateful to Jinhe Bi for his guidance, insightful advice, and generous support during the development of this work. His input greatly helped me navigate challenges and improve the clarity of my contributions.
On a more personal note, I wish to express my deepest gratitude to my wife, Xheneta, and to my family for their unwavering support, patience, and encouragement. Their belief in me provided the strength and motivation to complete this journey.
Finally, I dedicate this thesis to the memory of my father, whose appreciation for education and hard work continues to inspire me every day.

Munich, September 2025

Abstract

Large language models (LLMs) have achieved impressive success across a wide range of reasoning-intensive tasks, but their alignment with human-preferred behavior typically depends on reinforcement learning (RL) with correctness-based rewards. Such rewards are often sparse, delayed, and computationally costly to evaluate, making them a bottleneck for scalable RL fine-tuning. This thesis explores whether intrinsic properties of the models themselves—specifically, changes in hidden-state representations as a model reasons through a problem—can serve as meaningful reward signals. These dynamics reflect how confidently and consistently the model progresses through its reasoning trajectory, offering a potentially cheaper and denser alternative to external correctness checks.

We conduct two complementary sets of experiments to evaluate this idea. In the first, we instrument GRPO training runs driven by external correctness rewards (a weighted sum of 90% accuracy and 10% format) and collect hidden-state statistics purely as observational probes. This setting allows us to test whether hidden-state dynamics correlate with correctness, independent of optimization. Results show a clear positive alignment: rollouts with stronger internal signals are substantially more likely to produce correct answers, with the effect being strongest in larger models. Quantitative analyses using AUROC, AUPR, and FPR@95 confirm that these signals provide non-trivial predictive power.

In the second set of experiments, we replace external rewards entirely with intrinsic ones and train multiple model scales on reasoning benchmarks. Here, both the 1.5B and 3B models quickly collapsed in accuracy, leading to early termination of training, while the 7B model completed the full run but still failed to achieve stable accuracy gains. In all cases, the intrinsic reward itself was readily optimizable, but task-level correctness consistently degraded. A contributing factor is the heavy reliance of EasyR1 on strict formatting: without external incentives to preserve boxed answers and `<think>` tags, models trained with intrinsic rewards alone often produced outputs that optimized the hidden-state trajectory but could not be parsed as correct.

Taken together, these findings establish that hidden-state dynamics encode meaningful information about reasoning quality. While they are not yet sufficient to drive learning when used directly, they provide a valuable foundation for future alignment techniques. In particular, strategies such as reward shaping, percentile-based filtering, discretized reward assignment, hybrid intrinsic–extrinsic signals, and data-efficient training may help stabilize optimization and fully harness hidden-state dynamics as reinforcement signals.

Contents

1 Introduction

Large Language Models (LLMs) have achieved remarkable performance across a wide range of natural language processing tasks, driven by advances in transformer architectures, large-scale pretraining, and instruction tuning [1, 2]. However, aligning such models with task-specific goals or human preferences often relies on reinforcement learning from human feedback (RLHF) or other external reward mechanisms. These approaches require costly human annotation, domain-specific verifiers, or curated evaluation datasets, which are not always available — especially for specialized or low-resource domains.
This thesis investigates an alternative approach: training LLMs using *intrinsic* or *internal* rewards derived solely from the model itself. Specifically, it focuses on probability-based reward functions, where the model's own per-token likelihood for a reference answer serves as the reward signal [3, 4]. This eliminates the dependency on external evaluators, enabling model training in any domain where only input–output pairs are available. The internal reward acts as a self-assessment of answer quality, leveraging the model's internal confidence to guide optimization.
This thesis is situated entirely in the text-only LLM setting, without multimodal components. As the core optimization framework, it employs Generalized Reinforced Policy Optimization (GRPO) — a recent variant of reinforcement learning fine-tuning that allows flexible reward shaping and efficient scaling to large models. By combining GRPO with intrinsic probability rewards, this thesis explores whether models can improve reasoning quality and task performance without external feedback, and compares their performance directly to models trained with external reward signals.
The contributions of this thesis are threefold:

1. It formalizes a probability-based intrinsic reward function tailored to the LLM setting.

2. It integrates this reward into a GRPO training pipeline and evaluates its effectiveness on open-source instruction-tuned models such as Qwen [5].

3. It provides a detailed empirical comparison between internal-reward-trained models and external-reward-trained models, analyzing performance, computational efficiency, and domain-general applicability.

By focusing on internal rewards, this thesis aims to lower the barrier for domain adaptation and reasoning improvement in LLMs, making reinforcement learning fine-tuning feasible even in contexts where external reward signals are scarce or non-existent.

1.1 Machine Learning

Machine Learning (ML) is a field of artificial intelligence concerned with building algorithms that can learn patterns from data and improve performance over time without being explicitly programmed for every possible scenario [6, 7]. Formally, we can consider a model as a parameterized function:

$$f_\theta : \mathcal{X} \to \mathcal{Y}$$

where:

- $\mathcal{X}$ is the input space,
- $\mathcal{Y}$ is the output space,
- $\theta \in \mathbb{R}^n$ is the set of model parameters (weights and biases) that determine the function's behavior.

1.1.1 Parameters and Learning

The learning process consists of adjusting θ such that the model predictions $f_\theta(x)$ are as close as possible to the desired outputs y for given training examples (x, y). This is typically achieved by minimizing a *loss function* $L(\theta)$:

$$\theta^* = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \ell\big(f_\theta(x_i), y_i\big)$$

where ℓ is the sample-wise loss (e.g., mean squared error, cross-entropy), and N is the number of training examples.

1.1.2 Training Process

A common optimization approach is stochastic gradient descent (SGD) [8] and its variants (e.g., Adam, RMSProp), which iteratively update parameters in the opposite direction of the loss gradient:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

where η is the learning rate, controlling the step size.

1.1.3 Types of Machine Learning

Depending on the availability and nature of labels, ML can be categorized into:

- **Supervised learning:** The model is trained on labeled pairs (x, y), learning a mapping from inputs to known outputs.
- **Unsupervised learning:** The model is trained without explicit labels, aiming to discover hidden patterns or structures in the data (e.g., clustering, dimensionality reduction).
- **Self-supervised learning:** A form of supervised learning where the labels are automatically derived from the input data itself (e.g., predicting missing words in a sentence). Large Language Models (LLMs) are commonly trained this way.
- **Reinforcement learning:** An agent interacts with an environment and learns to take actions that maximize a cumulative reward signal.

1.1.4 Evaluation

After training, the model is evaluated on unseen data (*test set*) to measure generalization performance using metrics such as accuracy, F1-score, or perplexity (in the case of language models).
This foundational understanding of ML concepts is essential for comprehending how Large Language Models operate and how they are trained to predict the next token in a sequence.

1.2 Large language models

Large Language Models (LLMs) are deep neural networks trained to model the probability distribution of sequences of tokens. Formally, given a sequence of tokens:

$$x = (x_1, x_2, \ldots, x_T)$$

the model estimates the joint probability as:

$$P(x) = \prod_{t=1}^{T} P(x_t \mid x_{<t})$$

where $x_{<t}$ denotes all tokens preceding position t.

1.2.1 Next-Token Prediction

At inference time, LLMs predict the next token x_t given the preceding context $x_{<t}$:

$$x_t = \arg\max_{v \in V} \; P(v \mid x_{<t}; \theta)$$

where:

- V is the vocabulary,
- θ are the learned model parameters.

The conditional probability $P(v \mid x_{<t}; \theta)$ is obtained by applying the *softmax* function to the output logits z_v:

$$P(v \mid x_{<t}; \theta) = \frac{\exp(z_v)}{\sum_{u \in V} \exp(z_u)}$$

1.2.2 Training Objective

LLMs are typically trained to minimize the negative log-likelihood (cross-entropy loss):

$$\mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \log P_\theta(x_t \mid x_{<t})$$

This encourages the model to assign high probability to the correct next token in a sequence.

1.2.3 Decoding Strategies

During generation, various decoding strategies can be applied [9]:

- **Greedy decoding**: selecting the token with the highest probability at each step,
- **Random sampling**: sampling tokens proportionally to their probabilities,
- **Temperature scaling**: adjusting the sharpness of the probability distribution before sampling, where lower temperatures ($\tau < 1$) make the distribution more peaked (deterministic) and higher temperatures ($\tau > 1$) make it flatter (more diverse):

$$P(v \mid x_{<t}; \theta, \tau) = \frac{\exp(z_v/\tau)}{\sum_{u \in V} \exp(z_u/\tau)}$$

- **Top-k sampling**: restricting sampling to the k most probable tokens,
- **Nucleus (Top-p) sampling**: sampling from the smallest set of tokens whose cumulative probability exceeds p.

Most state-of-the-art LLMs are implemented using the *Transformer* architecture, described in detail in Section 1.3.

1.3 Transformers

Transformers have emerged as the foundational architecture for modern LLMs, replacing earlier recurrent and convolutional models in natural language processing (NLP) [1]. The architecture relies entirely on attention mechanisms, removing recurrence and enabling highly parallelized training (see Figure 1). In this thesis, we focus on the decoder stack, as used in autoregressive large language models such as GPT and Qwen, where the model predicts the next token based on all previous tokens.

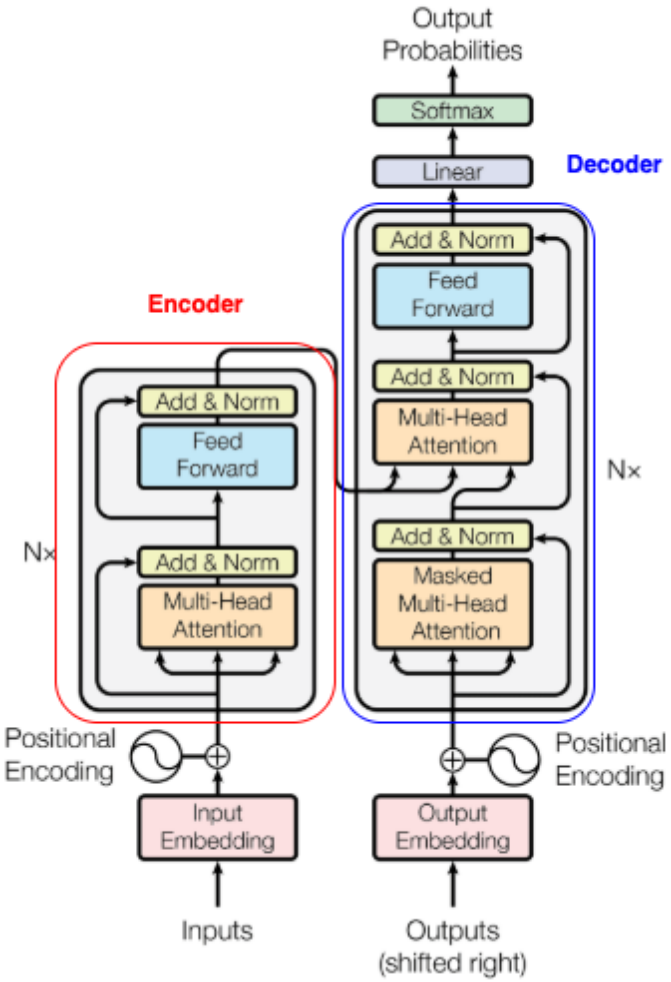

Figure 1: Transformer architecture with encoder (left, red) and decoder (right, blue), adapted from [1].

Core Components

A standard transformer consists of stacked layers, each containing:

- Multi-head self-attention mechanisms,
- Position-wise feed-forward networks,

- Residual connections and layer normalization.

For autoregressive models (e.g., GPT-family), only the *decoder stack* is used.

1.3.1 Learnable Parameters

Training a transformer means learning all its weight matrices and biases, which include:

- **Embedding matrices:** Token embeddings and positional encodings that map discrete token IDs to continuous vectors.
- **Attention projections:** Weight matrices for queries (W_Q), keys (W_K), and values (W_V) in each attention head.
- **Feed-forward networks:** Two linear transformations (W_1, W_2) with a nonlinearity (e.g., GELU) in between, applied independently to each token position.
- **Output projection:** A linear layer mapping hidden states back to vocabulary logits.
- **Layer normalization parameters:** Scale and shift terms for stabilizing training.

All these parameters, collectively denoted as θ, are optimized jointly to minimize the training objective.

1.3.2 Input Representation

Tokens are first mapped to continuous embeddings:

$$E = (e_1, e_2, \ldots, e_T), \quad e_t \in \mathbb{R}^d$$

Positional encodings (learned or sinusoidal) are added to embeddings to incorporate sequence order, since attention is inherently permutation-invariant.

1.3.3 Masked Self-Attention

The attention mechanism computes contextualized token representations:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V$$

where:

- Q, K, V are obtained from linear projections of E using W_Q, W_K, W_V,
- M is a causal mask that prevents attending to future tokens.

Multi-head attention uses multiple sets of Q, K, V projections to capture different relational patterns.

1.3.4 Advantages for LLMs

Self-attention enables each token to directly attend to all previous tokens, efficiently modeling long-range dependencies in a single operation. This ability to leverage global context has been crucial for improvements in reasoning, summarization, and other language tasks [10].
In the context of this thesis, the token-level probability outputs from transformer decoders form the basis for intrinsic reward computation, allowing direct measurement of the model's internal confidence without external verifiers.

2 Related Work

This chapter reviews the landscape of reinforcement learning approaches applied to large language models (LLMs), with a particular emphasis on alignment strategies that balance efficiency and scalability. The discussion progresses from traditional reinforcement learning from human feedback (RLHF), to verifiable reward frameworks (RLVR) that leverage programmatic correctness checks, and finally to more recent efforts that eliminate external reward dependencies by exploiting intrinsic signals derived from the models themselves. Together, these lines of research provide the methodological context and motivation for this thesis, which investigates whether hidden-state dynamics can serve as a reliable basis for reinforcement learning in reasoning tasks.

2.1 Reinforcement Learning for LLMs

The introduction of reinforcement learning into LLM post-training began with reinforcement learning from human feedback (RLHF), where a reward model trained on human preference data guides policy optimization to better align model outputs with human intent [2]. In recent years, especially from 2024 to 2025, research has increasingly emphasized reasoning-specific RL methods, verifiable reward signals, and scaling laws for reasoning alignment [11, 12, 13]. The DeepSeek-R1 framework and its many open replications have popularized techniques such as Grouped Reinforcement Policy Optimization (GRPO) and demonstrated strong performance improvements in math, coding, and logic benchmarks [14]. Comprehensive surveys such as "100 Days After DeepSeek-R1" document best practices for data curation, KL-penalty tuning, rollout sampling strategies, and reasoning-centric evaluation [14].
A recurring insight is the shift from general preference-based rewards toward *task-grounded* signals, which reduce reward hacking and align training more closely with downstream evaluation objectives. This shift underpins many of the methodological advances surveyed by [14] and subsequent replications.

2.2 RL with Verifiable Rewards (RLVR) for LLMs

RL with verifiable rewards (RLVR) replaces the learned reward model with an *automatically checkable* signal, such as exact string match for math answers, execution-based correctness for code, or formal verification in structured domains [15]. This allows scalable, low-latency feedback during RL without the need for expensive human annotations. RLVR has been central to recent reasoning pipelines, especially GRPO-based training with programmatic correctness checks, and is a core component of many high-performing open-source reasoning LLMs [14].
However, RLVR is inherently constrained to domains with objective verifiers (math, coding, formal logic). Several works have proposed *verifier-reduced* or *verifier-free* variants to extend coverage. For example, **NOVER** incorporates supervised signals into an RL objective without using an explicit external verifier, enabling RL-style improvements in general text-to-text generation [16]. Other works focus on refining *credit assignment* in RLVR; **KTAE** (Key-Token Advantage Estimation) introduces token-level advantage esti-

mation based on the statistical association between specific tokens and rollout correctness, improving both accuracy and output conciseness without additional models [17].

2.3 RL for LLMs *without* External Rewards

A growing body of research removes the dependency on external verifiers entirely by using the model's own *intrinsic* signals as feedback. **INTUITOR** [18] is a method which implements reinforcement learning from internal feedback (RLIF) by replacing the GRPO reward with the model's self-assessed certainty (confidence) in its generated answers, showing competitive in-domain performance and strong generalization to out-of-domain tasks.

Related approaches focus on confidence- and entropy-based objectives. **RLSC** (Reinforcement Learning via Self-Confidence) [19] rewards high-confidence generations, yielding measurable reasoning gains with minimal training cost on Qwen-based models. **Entropy Minimization** [20] optimizes the negative entropy of token distributions directly, without labels or external rewards, achieving performance competitive with GRPO on reasoning tasks and enabling inference-time uncertainty reduction without parameter updates.

These intrinsic-reward approaches align with findings from verification-free reasoning research [21], which show that LLMs possess substantial latent knowledge and that leveraging internal certainty signals can drive domain-agnostic improvements. By removing the need for curated verifiers, these methods open scalable paths for RL training across domains where explicit reward signals are scarce or unavailable—an approach central to this thesis's focus on text-only LLMs.

3 Methodology

This chapter describes the training methodology employed in this thesis. All experiments are implemented using the open-source **EasyR1** framework[1], which provides a modular and scalable environment for reinforcement learning (RL) post-training of LLMs. EasyR1 supports algorithms such as PPO and GRPO, integrates seamlessly with distributed training setups, and offers a flexible interface for defining custom reward functions—making it well-suited for integrating the proposed *Chain of Enlightenment (CoE)* reward.
We first outline how RL is typically applied to LLMs, including the widely used *supervised fine-tuning* (SFT) phase that precedes RL optimization. In the standard RLHF pipeline [2], a pretrained base model is initially adapted to follow instructions via SFT on curated human-annotated data. A separate reward model is then trained from human preference data, and policy optimization (e.g., PPO or GRPO) is performed with the reward model as the feedback signal.
Recent advances in reasoning-centric LLM training have demonstrated that the SFT stage can be skipped entirely. For example, the DeepSeek-R1 framework [11] directly applies *Grouped Reinforcement Policy Optimization* (GRPO) to the pretrained base model, optimizing it using domain-specific verifiable rewards for reasoning tasks. This direct-to-RL approach reduces complexity, eliminates intermediate fine-tuning stages, and focuses optimization efforts on the final reasoning behavior.
In this thesis, we follow a similar philosophy by starting directly from a pretrained base model and applying RL without an SFT stage. However, instead of using external, task-specific verifiers as in RLVR, we introduce an *intrinsic reward mechanism* based on the CoE framework. This approach computes rewards from the model's own hidden state dynamics during inference, capturing the evolution of internal representations across transformer layers. By leveraging these internal signals, we can train LLMs in a *domain-agnostic* manner, removing the dependency on manually engineered or domain-restricted external reward functions while still retaining the benefits of RL-based post-training.

3.1 Reinforcement Learning for LLMs

Reinforcement learning has become a key component in aligning large language models (LLMs) with desired behaviors and task requirements. The canonical setup is *reinforcement learning from human feedback* (RLHF) [2], which is structured around three main stages and is illustrated in Figure 2:

1. **Supervised Fine-Tuning (SFT):** The pretrained base model is first fine-tuned on high-quality, human-curated instruction–response pairs. This step adapts the model to an instruction-following style before reinforcement learning.

2. **Reward Model Training:** A separate reward model is trained to predict human preference scores, given pairs of model outputs for the same prompt. This reward model acts as a proxy for human judgment during RL.

[1] `https://github.com/hiyouga/EasyR1`

3. **Policy Optimization:** The SFT-adapted model is then optimized using an RL algorithm—commonly Proximal Policy Optimization (PPO) or its variant Grouped Reinforcement Policy Optimization (GRPO). Optimization maximizes the learned reward while applying KL regularization to keep the fine-tuned policy close to the original distribution.

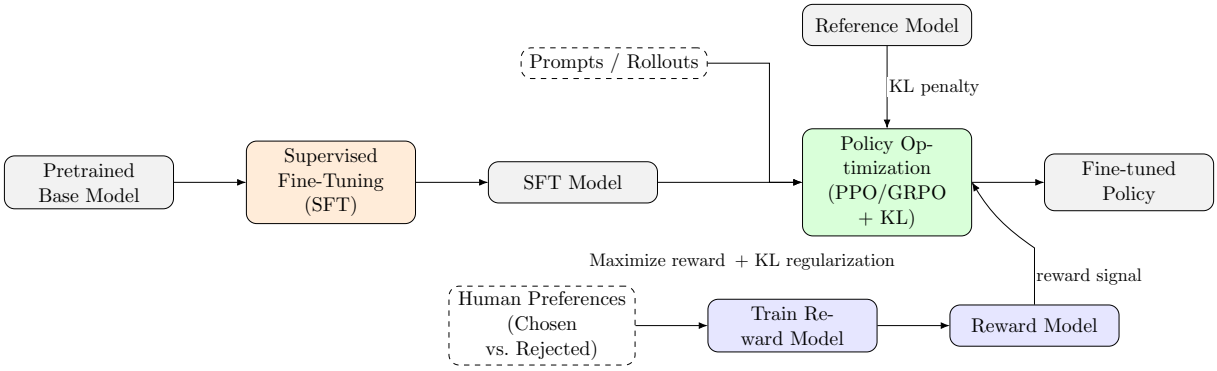

Figure 2: RLHF pipeline: a pretrained model is adapted via SFT, a reward model is trained from human preferences, and the policy is optimized with PPO/GRPO under a KL constraint to a reference model.

Direct-to-RL Approaches

While SFT is common, recent reasoning-focused pipelines such as DeepSeek-R1 [11] skip the SFT phase entirely. Instead, they perform direct policy optimization starting from the pretrained base model, using *verifiable* task-specific rewards (e.g., exact match in math problems, unit tests for code). This reduces intermediate steps, avoids potential overfitting from small SFT datasets, and focuses optimization on the final objective from the start.

Position of This Thesis

Our work adopts the direct-to-RL philosophy but replaces external, domain-specific verifiers with an intrinsic reward signal based on the model's hidden state dynamics—the *Chain of Enlightenment (CoE)* method (see Section 3.3). This enables RL training in settings where external verifiers are unavailable, while retaining the benefits of iterative policy improvement.

3.2 Group Relative Policy Optimization (GRPO)

Group Relative Policy Optimization (GRPO) [22] is a reinforcement learning algorithm tailored for large language model (LLM) training, especially in reasoning-intensive settings. Unlike standard Proximal Policy Optimization (PPO) [23], which relies on a learned value function for advantage estimation, GRPO removes the critic entirely and computes *group-relative* advantages by comparing rewards among multiple responses to the same prompt.

Let a prompt be represented as a token sequence:

$$x = (x_1, x_2, \ldots, x_T),$$

and let $\{y^{(i)}\}_{i=1}^{G}$ be G different completions sampled from the current model parameters θ. Each completion receives a scalar reward $r^{(i)}$, and the *group mean reward* is:

$$\bar{r} = \frac{1}{G} \sum_{j=1}^{G} r^{(j)}.$$

The group-relative advantage for the i-th completion is then:

$$A^{(i)} = r^{(i)} - \bar{r}.$$

This removes prompt-level bias and stabilizes training by making rewards comparable across completions of the same prompt.
The GRPO objective is:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_{x,y^{(i)}} \left[\min \left(\rho^{(i)}(\theta) \cdot A^{(i)},\ \text{clip} \left(\rho^{(i)}(\theta), 1-\epsilon, 1+\epsilon \right) \cdot A^{(i)} \right) - \beta \cdot \text{KL}\big(P_\theta(\cdot \mid x) \,\|\, P_{\theta_{\text{ref}}}(\cdot \mid x)\big) \right],$$

where:

- $\rho^{(i)}(\theta) = \frac{P_\theta(y^{(i)}|x)}{P_{\theta_{\text{old}}}(y^{(i)}|x)}$ is the likelihood ratio,
- ϵ is the clipping parameter to prevent destructive updates,
- β controls the KL penalty against a reference model $P_{\theta_{\text{ref}}}$.

A visual comparison of PPO and GRPO is provided in Figure 3. While PPO uses an actor–critic setup with a learned value function, GRPO removes the critic and instead uses group-relative advantages computed across responses to the same prompt. This structural simplification makes GRPO more efficient for reasoning LLMs, where reward magnitudes can vary widely.

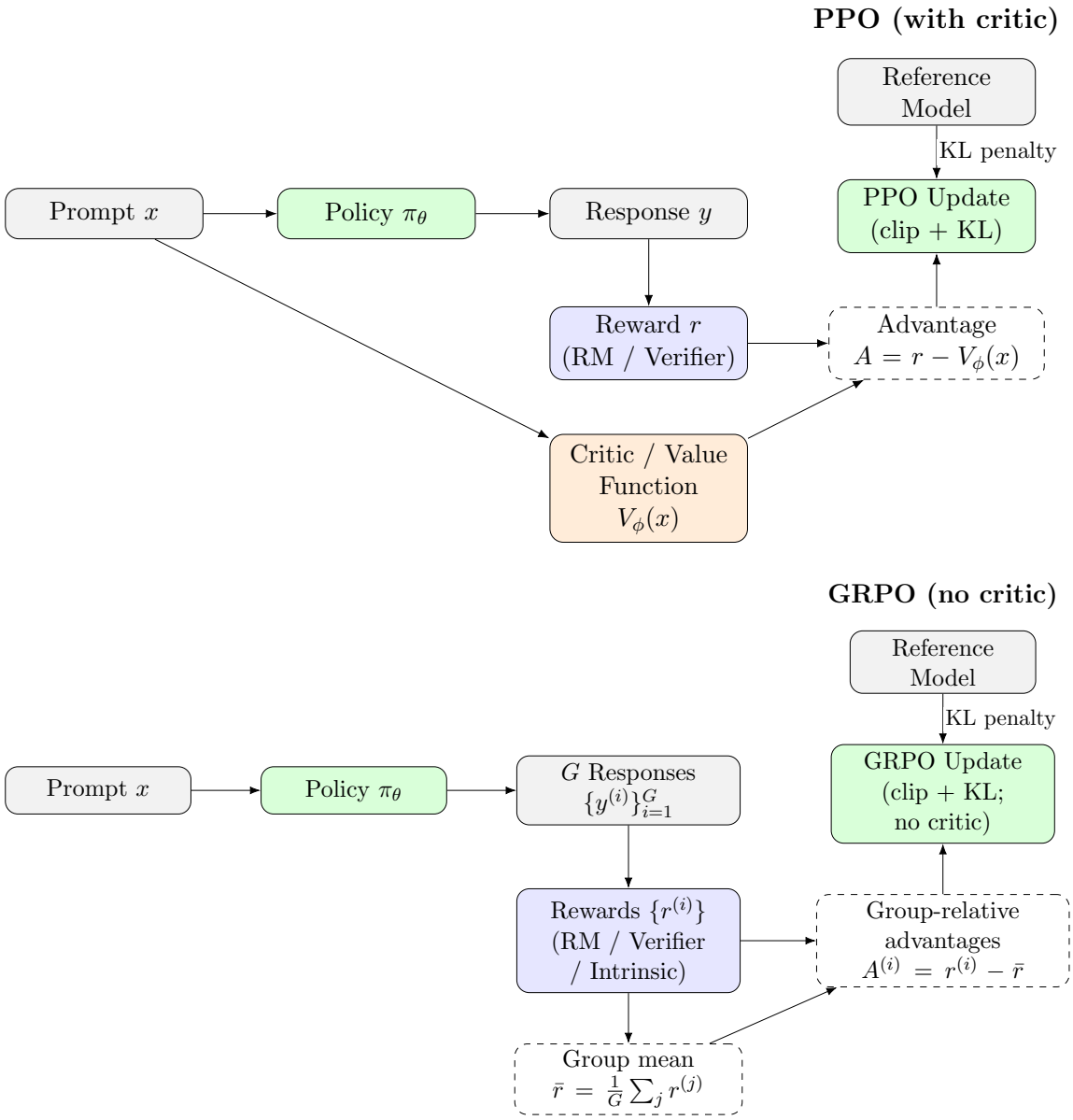

Figure 3: Conceptual comparison of PPO and GRPO. **PPO** uses an actor–critic setup where a value function (critic) estimates $V_\phi(x)$ and advantages $A = r - V$. **GRPO** removes the critic and uses group-relative advantages $A^{(i)} = r^{(i)} - \bar{r}$ from multiple responses to the same prompt. Both use a clipped update with a KL penalty to a reference model.

In this thesis, GRPO is used as the main optimization algorithm, with the external verifier replaced by our proposed *internal* Chain of Enlightenment (CoE) reward computed from hidden states.

3.3 From Hidden-State Dynamics to an Intrinsic Reward

We construct an intrinsic, trajectory-level signal from hidden-state dynamics and use it to drive policy optimization with grouped relative advantages. The process is illustrated

in Figure 4. Unlike conventional reinforcement learning with external reward models, our approach exploits *internal* signals directly from the policy's hidden representations.

Notation. Let x denote a prompt and $\hat{y} = (y_1, \ldots, y_T)$ a sampled response. During a forward pass of the actor (decoder-only transformer), we expose per-layer, per-token hidden states

$$\mathbf{H}_t^{(\ell)} \in \mathbb{R}^d, \qquad \ell = 0, \ldots, L, \quad t = 1, \ldots, T.$$

Layerwise dynamics. We measure representation change across layers at fixed token positions:

$$\text{First order ("velocity"):} \quad \Delta_{\ell,t}^{(1)} = \mathbf{H}_t^{(\ell+1)} - \mathbf{H}_t^{(\ell)}, \tag{1}$$

$$\text{Second order ("curvature"):} \quad \Delta_{\ell,t}^{(2)} = \mathbf{H}_t^{(\ell+1)} - 2\,\mathbf{H}_t^{(\ell)} + \mathbf{H}_t^{(\ell-1)}. \tag{2}$$

Normalization. To ensure comparability across sequences, we scale differences by a single denominator:

$$Z = \left\|\mathbf{H}_{\text{avg}}^{(L)} - \mathbf{H}_{\text{avg}}^{(0)}\right\|_2 + \varepsilon, \qquad \mathbf{H}_{\text{avg}}^{(\ell)} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{H}_t^{(\ell)}.$$

This stabilizes energy magnitudes and aligns training with the rollout code path.

Energy definition. We define token-level contributions and an overall sequence energy:

$$e_t = \frac{1}{L-1}\sum_{\ell=0}^{L-1} \frac{\|\Delta_{\ell,t}^{(1)}\|_2}{Z} + \frac{1}{L-2}\sum_{\ell=1}^{L-1} \frac{\|\Delta_{\ell,t}^{(2)}\|_2}{Z}, \qquad E = \frac{1}{T}\sum_{t=1}^{T} e_t. \tag{3}$$

Here, E reflects how much hidden states "move" and "curve" while generating the response.

Groupwise normalization. For each prompt, we draw G responses and compute $\{E_j\}_{j=1}^{G}$. To remove scale ambiguity, we normalize per prompt:

$$\tilde{E}_j = \begin{cases} \dfrac{E_j - \min_j E_j}{\max_j E_j - \min_j E_j + \epsilon} & \text{if } \max_j E_j \neq \min_j E_j, \\ 0 & \text{otherwise.} \end{cases}$$

This yields $\tilde{E}_j \in [0, 1]$, preserving only relative rankings within the group.
In practice, an additional *per-prompt UID normalization* step was applied before rewards were passed into the GRPO update. This ensures that, for each prompt, the rollout with the highest intrinsic score is mapped to 1.0, the lowest to 0.0, and all others scaled proportionally in between. This second normalization stage provides scale invariance across batches and prevents instability when absolute magnitudes differ between prompts.

Token-level shaping. The scalar $\tilde{E}_j$ is distributed uniformly across valid response tokens so that rewards sum to $\tilde{E}_j$, improving credit assignment without training a critic.

From rewards to advantages. The normalized rewards $\{r_{\text{int}}^{(i)}\}$ are aggregated into a group mean and relative advantages:

$$\bar{r} \;=\; \frac{1}{G}\sum_{j=1}^{G} r_{\text{int}}^{(j)}, \qquad A^{(i)} \;=\; r_{\text{int}}^{(i)} - \bar{r}.$$

These $A^{(i)}$ values drive a GRPO update with PPO-style clipping and (optionally) a KL penalty to a frozen reference model.

Pipeline illustration. Figure 4 shows the pipeline. A policy π_θ generates G responses per prompt. For each response, hidden states across layers are collected and transformed by the internal reward module into per-response rewards. These are normalized within the group, converted into relative advantages, and used to update the policy. No external reward model or verifier is used; the diagram explicitly crosses them out.

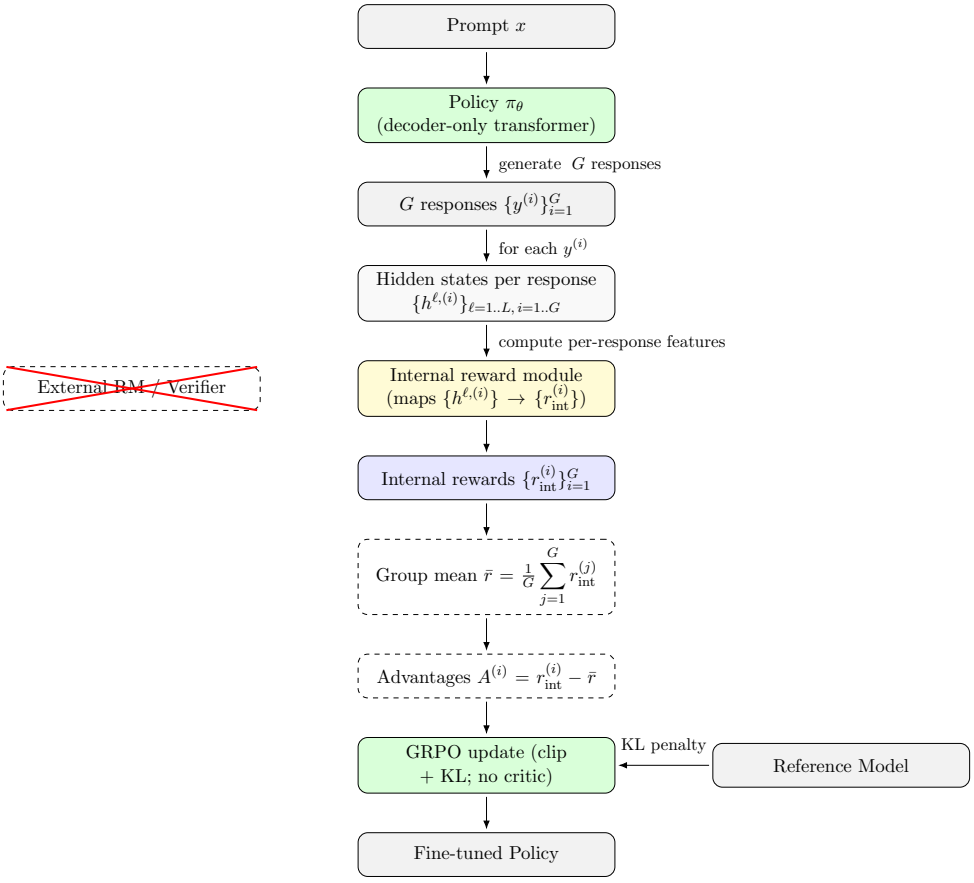

Figure 4: Internal reward with multiple responses. The actor generates G responses per prompt, hidden states are converted into per-response intrinsic rewards, which are normalized and turned into relative advantages for a GRPO update. External reward models are not used.

4 Implementation

This chapter describes how the conceptual framework introduced in Chapter 3 was realized in practice. Whereas the methodology presented the high-level design and motivation, here we focus on the concrete execution: the training loop, reward computation, batching scheme, and runtime orchestration. Figures from the methodology (e.g. Figure 4) illustrated the conceptual flow; in this chapter we emphasize the actual implementation details and algorithms.

4.1 System Architecture

The runtime is organized around a single controller that coordinates three roles:

- **Actor:** the trainable transformer model, sharded with FSDP and updated via policy-gradient steps.
- **Rollout engine:** a lightweight decoding backend that produces multiple candidate responses per prompt.
- **Reference policy:** an optional frozen copy used only for KL regularization.

The controller samples prompts, requests k rollouts per prompt, computes intrinsic rewards by inspecting hidden states, normalizes them within each group, and finally issues an update to the actor. This separation between actor and rollout engine enables both high throughput and memory efficiency.

4.2 Technology Stack

This chapter summarizes the software and hardware technologies underpinning our system. The stack is organized in layers—from data handling and deep learning frameworks through distributed training, algorithms, monitoring, and deployment—reflecting how components interact during large-scale experiments. The objective is reproducibility, scalability, and practical operability on HPC clusters while supporting modern reinforcement-learning post-training of LLMs. A schematic overview of this layered architecture is shown in Figure 5.

Deep Learning Frameworks
PyTorch; *FSDP* (Fully Sharded Data Parallel); Hugging Face *Transformers*; Gradient checkpointing; Mixed precision (BF16/FP16); FlashAttention

Algorithms & Training
GRPO (group-relative PPO, no critic); PPO (baseline); KL regularization; CoE intrinsic reward (hidden-state dynamics); Token-level shaping

Models
Qwen2.5-7B-Instruct (base); support for VLMs (vision-language); Tensor parallelism; Weight sync Actor $\leftrightarrow$ Rollout

Distributed & Infrastructure
Ray (orchestration); SLURM (HPC scheduling); Multi-GPU/Multi-node; Docker/Enroot; NVIDIA NGC registry

Data & Processing
Hugging Face Datasets; JSON/PyArrow/HDF5; Jinja2 prompt templates; MathRuler grading; SymPy parsing; LaTeX processing

Monitoring & Experiment Management
Weights & Biases (W&B), SwanLab; Console logs; Metadata and checkpoints

Deployment & Inference
Inference endpoints; Quantization; Export tools; API interfaces

Figure 5: Technology stack used in this project, layered from core training components down to runtime, data tooling, observability, and deployment.

4.2.1 Rationale and interactions

The system architecture can be understood as a layered stack, where each component builds on the previous to ensure scalability, stability, and reproducibility:

- **Frameworks layer:** Provides the foundation with PyTorch, FSDP, Hugging Face Transformers, and mixed precision training to enable efficient large-scale training and memory feasibility.
- **Algorithms layer:** Implements GRPO (without critic) combined with KL regularization and CoE-based intrinsic rewards. Token-level shaping improves credit assignment without the need for a value model.
- **Models layer:** Centers on Qwen2.5-7B-Instruct, supporting tensor-parallel inference and weight synchronization between actor and rollout engines.
- **Distributed & Infrastructure layer:** Relies on Ray, SLURM, Docker/Enroot, and NGC containers to provide orchestration, scheduling, and reproducible environments for multi-node GPU clusters.

- **Data & Processing layer:** Standardizes inputs using Hugging Face Datasets, Jinja2 prompting, and mathematical tooling (MathRuler, SymPy, LaTeX) to ensure consistent grading and evaluation.
- **Monitoring layer:** Uses W&B and SwanLab to capture metrics, artifacts, and checkpoints, underpinning transparency and reproducibility.
- **Deployment layer:** Covers inference endpoints, quantization, and model export, enabling downstream use of trained models.

Together, these layers constitute a coherent stack that balances *throughput* (tensor/multi-GPU parallelism, rollout separation), *stability* (KL regularization, normalized intrinsic rewards), and *reproducibility* (containers, logging, checkpoints), enabling rigorous experimentation at scale.

4.3 Intrinsic Reward Computation

The intrinsic reward signal is computed directly from hidden-state dynamics. While Chapter 3.3 introduced this idea conceptually, here we describe the concrete implementation details as realized in our codebase.

1. For each sampled response, hidden states are exposed at every layer.
2. Instead of operating on all token positions, we apply a **mean-pooling step across response tokens** at each layer. This produces one representative vector per layer, reducing memory and storage overhead while still capturing trajectory-level dynamics.
3. First- and second-order differences across layers are then computed on these pooled vectors (Eqs. 1, 2), yielding $\Delta^{(1)}$ and $\Delta^{(2)}$ signals that describe the magnitude and curvature of representation change.
4. The differences are normalized by a global scale factor Z (Eq. 3), ensuring comparability across responses of varying lengths and magnitudes.
5. The resulting statistics (mean norms of $\Delta^{(1)}$ and $\Delta^{(2)}$) are aggregated into a single sequence-level energy score E.
6. Within each group of G rollouts, per-sequence energies are min–max normalized to produce $\tilde{E}$ values in $[0, 1]$, preserving only their relative ranking.
7. **Per-prompt UID normalization:** before rewards are passed into the GRPO update, each rollout group (identified by a prompt UID) is normalized again. The highest-scoring response is mapped to 1.0, the lowest to 0.0, and intermediate ones scaled proportionally. This step ensures scale invariance across prompts and batches.
8. Finally, the scalar $\tilde{E}$ is distributed uniformly across the response tokens. This shaping improves credit assignment in the absence of a critic by attributing the reward back to all tokens.

This implementation differs slightly from the conceptual formulation: rather than computing dynamics at the token level and averaging afterward, we first mean-pool over tokens at each layer and then compute inter-layer dynamics. This design choice makes the approach computationally feasible on large models, while retaining the key property that the reward reflects how much the hidden-state trajectory "moves" and "curves" across the network during generation.

4.4 Training Loop with Relative Advantages

Each training step processes a batch of prompts, generates multiple responses per prompt, evaluates them with the intrinsic reward, and updates the actor with groupwise relative advantages. The full procedure is summarized in Algorithm 1.

Algorithm 1 One training iteration (intrinsic CoE + grouped relative advantages)

Inputs: batch size B, group size k, actor π_θ, rollout backend G, (optional) reference π_{ref}, KL coefficient $\beta \geq 0$
Sample prompts $\{x_i\}_{i=1}^{B}$
for $i = 1$ to B **do**
  $\{\hat{y}_{i,j}\}_{j=1}^{k} \leftarrow \textsc{Generate}(G, \pi_\theta, x_i, k)$ ▷ decode k responses per prompt
end for
Compute token log-probabilities $\{\log \pi_\theta(y_{i,j,t} \mid x_i, y_{i,j,<t})\}$ under the current actor
if KL regularization is enabled **then**
  Compute $\{\log \pi_{\text{ref}}(y_{i,j,t} \mid x_i, y_{i,j,<t})\}$ and per-token $\text{KL}_{i,j,t}$
end if
for each sequence (i, j) **do** ▷ intrinsic CoE energy from hidden-state dynamics
  Expose per-layer hidden states for $\hat{y}_{i,j}$
  Compute first- and second-order layerwise differences
  Normalize by scale Z; obtain $d1\_\text{mean}^{(i,j)}$, $d2\_\text{mean}^{(i,j)}$
  $E_{i,j} \leftarrow d1\_\text{mean}^{(i,j)} + d2\_\text{mean}^{(i,j)}$
end for
for each prompt i **do** ▷ per-prompt min–max normalization
  $\tilde{E}_{i,j} \leftarrow \dfrac{E_{i,j} - \min_j E_{i,j}}{\max_j E_{i,j} - \min_j E_{i,j} + \epsilon}$ for $j = 1..k$
  $a_{i,j} \leftarrow \tilde{E}_{i,j} - \dfrac{1}{k}\sum_{j'=1}^{k} \tilde{E}_{i,j'}$ (relative advantage)
  Convert $\tilde{E}_{i,j}$ to token rewards $\{r_{i,j,t}\}$ with $\sum_t r_{i,j,t} = \tilde{E}_{i,j}$
end for
Define the objective:

$$\mathcal{L}(\theta) = \sum_{i=1}^{B}\sum_{j=1}^{k} a_{i,j} \sum_{t} \log \pi_\theta(y_{i,j,t} \mid x_i, y_{i,j,<t}) \; - \; \beta \sum_{i,j,t} \text{KL}_{i,j,t}$$

UpdateActor$(\pi_\theta, \mathcal{L})$ ▷ take a gradient step on actor parameters
Return: updated actor π_θ

Because advantages are computed relatively within each group, no critic network is needed. This both simplifies the stack and reduces compute.

4.5 Hybrid Execution

The implementation combines FSDP for actor training with a dedicated rollout backend:

Actor. The trainable policy is wrapped with PyTorch FSDP. Parameters, gradients, and optimizer state are sharded across data-parallel ranks. Activation checkpointing and mixed precision are used to reduce memory.

Rollout. Decoding runs on a specialized inference engine (e.g. tensor-parallel backend) to maximize throughput. The rollout process has no optimizer; it mirrors the actor's parameters.

Synchronization. After each update, a lightweight mapping synchronizes actor parameters with the rollout engine. This avoids keeping duplicate model copies in memory.

Reference. When KL regularization is enabled, a frozen copy of the policy is used solely for KL computations. It can be offloaded to conserve GPU memory.

4.6 Data Flow and Batching

Prompts are tokenized with the same chat template used during evaluation. Batches are organized hierarchically: global batches feed one iteration, mini-batches define optimizer steps, and micro-batches are accumulated under FSDP.
Rollouts operate on prompt-only inputs, producing responses that are concatenated and masked so that gradients apply only to response tokens. All reward signals (energy, normalized scores, advantages) are first computed at the sequence level and then shaped across tokens.

4.7 Evaluation and Diagnostics

During training, we monitor whether intrinsic rewards align with correctness by computing AUROC, AUPR, and FPR@95 between energies $\{E\}$ and ground-truth correctness labels. These are computed without groupwise normalization.
We also conduct ablations on key knobs: group size k, inclusion of curvature terms, choice of normalization, and presence of KL. Our implementation logs all hyperparameters, random seeds, and hardware details, ensuring reproducibility.

4.8 Complexity and Scaling

Let B be prompts per iteration, k responses per prompt, T average response length, L layers, and d hidden size. Rollout cost is $O(BkT)$ forward tokens. Computing the CoE reward adds $O(BkTLd)$ vector operations (mainly norms and differences), which

are bandwidth-friendly and parallelizable. Because no critic is trained, backpropagation is limited to log-prob terms, so overall runtime is dominated by rollout throughput and FSDP communication.

5 Experiments

This chapter presents the experimental setup used to evaluate our proposed approach. We first detail the hardware and infrastructure on which experiments were executed, followed by the key hyperparameters chosen for training. Subsequently, we describe the models under study and the datasets employed, including training and evaluation splits. We then outline the baselines against which our method is compared and introduce the evaluation metrics used to assess performance. Together, these components provide a reproducible and transparent foundation for the subsequent Results chapter, where we analyze and interpret the outcomes of these experiments.

5.1 Hardware and Infrastructure

Experiments were conducted on three different high-performance computing (HPC) systems: **LRZ** [24], **Horeka** [25], and **FAU** [26]. Each system provided multi-GPU nodes with large memory capacity and high-speed interconnects, scheduled via the `SLURM` workload manager.
The following configurations were used for training runs:

- **LRZ (Leibniz Supercomputing Centre)**: Partition `mcml-hgx-h100-94x4`, equipped with **4× NVIDIA H100 (80 GB)** GPUs per node, 768 GB of RAM, and a maximum job runtime of 48 hours.
- **Horeka (KIT, Karlsruhe)**: Two partitions were employed:
  - `accelerated-h100` with **4× NVIDIA H100 (80 GB)** GPUs per node, 768 GB RAM, and 48-hour maximum runtime.
  - `accelerated` with **4× NVIDIA A100 (40 GB)** GPUs per node, 512 GB RAM, and 48-hour maximum runtime.
- **FAU (Friedrich-Alexander-Universität Erlangen–Nürnberg)**: Partition `a100`, equipped with **8× NVIDIA A100 (40 GB)** GPUs per node, either 1 TB or 2 TB of RAM, and a maximum runtime of 24 hours.

Checkpoints and logs were stored on distributed file systems (DSS/Lustre) accessible across nodes. Experiment tracking was managed with `Weights & Biases (W&B)` and `SwanLab`, which recorded hyperparameters, metrics, and system diagnostics. This enabled reproducibility and resumption of experiments across cluster sessions.
Overall, this combination of clusters provided sufficient flexibility to run large-scale training with Qwen2.5 models (1.5B, 3B, 7B), balancing memory, compute throughput, and runtime constraints depending on the experimental setup.

5.2 Hyperparameters

All experiments used a consistent set of hyperparameters across model sizes (Qwen2.5-1.5B, 3B, and 7B). Training was conducted on the `math12k` dataset, with the `train` split for optimization and the `test` split for evaluation. Each sample consists of a *prompt* (the

math problem) and a *response* (the model's solution). To ensure feasibility across different model sizes, prompts were limited to a maximum of 2048 tokens, while responses were also truncated at 2048 tokens. This distinction is important: the prompt length constrains the input sequence, whereas the response length constrains the maximum number of tokens generated by the model.

Rollout generation was performed with a batch size of 512, while validation used a batch size of 1024. A fixed random seed (1) ensured reproducibility, and prompts were shuffled during training.

Optimization employed Group Relative Policy Optimization (GRPO) with KL-regularization. The KL coefficient was set to 10^{-2} using a low-variance penalty, ensuring that the actor policy remained close to the reference model. The AdamW optimizer was used with a learning rate of 1×10^{-6}, weight decay of 1×10^{-2}, and gradient clipping at 1.0. Training was carried out for 15 epochs.

For each prompt, the actor generated $n = 5$ responses during training, using nucleus sampling with $p = 0.99$ and temperature $T = 1.0$. Evaluation employed deterministic decoding with a single response ($n = 1$) and a reduced temperature of $T = 0.5$. Rewards were computed via a batch reward function that directly compared generated answers against references.

The most important hyperparameters are summarized in Table 1. For completeness, the full configuration file is included in Listing 1 (see Appendix A.1).

Table 1: Key hyperparameters used in all experiments.

Category	**Value**
Dataset	`math12k` (train/test splits)
Prompt length (max)	2048 tokens (input problem)
Response length (max)	2048 tokens (generated solution)
Rollout batch size	512
Validation batch size	1024
Random seed	1
Algorithm	GRPO (Group Relative Policy Optimization)
KL coefficient	1.0×10^{-2} (low-variance)
Disable KL	false
Optimizer	AdamW
Learning rate	1.0×10^{-6}
Weight decay	1.0×10^{-2}
Gradient clipping	1.0
Epochs	15
Rollout n	5 responses per prompt
Sampling	Nucleus ($p = 0.99$), $T = 1.0$
Validation rollout n	1 response per prompt
Validation temperature	$T = 0.5$

5.3 Models

All experiments were conducted using models from the `Qwen2.5` family, an open-source suite of large language models released by Alibaba Cloud [27]. We specifically used the **instruction-tuned (Instruct)** variants, which differ from their base counterparts in the following way:

- **Base models** are pretrained autoregressive transformers trained on massive text corpora to predict the next token. While they provide strong language modeling capabilities, they are not optimized to follow human instructions directly.
- **Instruct models** are further fine-tuned on curated instruction–response datasets. This post-training step improves alignment with human prompts, enabling the models to follow instructions more reliably, generate structured solutions, and perform better in reasoning benchmarks.

We chose the `Instruct` variants because our study focuses on reinforcement learning fine-tuning (RLFT) for reasoning tasks, where starting from an instruction-aligned model provides a stronger baseline and reduces the reliance on supervised fine-tuning stages.
The following Qwen2.5-Instruct models were used:

- **Qwen2.5-1.5B-Instruct** [28]: 1.5 billion parameters, context length of up to 32k tokens. This lightweight model served as a baseline for testing training configurations with reduced compute cost.
- **Qwen2.5-3B-Instruct** [29]: 3 billion parameters, 32k token context length. It represents a mid-sized trade-off between computational efficiency and reasoning ability.
- **Qwen2.5-7B-Instruct** [30]: 7 billion parameters, 32k token context length. As the largest model used in this thesis, it required multi-GPU setups but demonstrated the strongest reasoning performance.

All three models are decoder-only transformers, instruction-tuned for better alignment. In this thesis, they were fine-tuned with Group Relative Policy Optimization (GRPO) using both intrinsic and external reward signals, enabling a systematic comparison across different parameter scales.

5.4 Dataset

We conduct all experiments on the **Math12k** dataset [31], an open-source collection of mathematical word problems released together with the `EasyR1` framework. Math12k comprises **12,500** problem–solution pairs, split into `train` (**12,000**) and `test` (**500**) examples; this split is used consistently throughout this work.
According to its dataset card, Math12k is a conversion of the `math_splits` from OpenAI's PRM800K repository, created via a lightweight script that extracts only the `problem` and `answer` fields and pushes them to Hugging Face; the resulting splits are precisely 12k/500. The dataset card lists the license as `MIT`. We also note that EasyR1's documentation references Math12k as the canonical text dataset example for the framework.
Each record contains:

- **problem** — the textual math question presented to the model,
- **answer** — the ground-truth solution string,
- **images** — optional image inputs for multimodal extensions (unused here).

Table 2 shows a representative entry.

Table 2: Example entry from the Math12k dataset.

Column	Content
problem	If a box contains 12 apples and 3 are removed, how many apples remain?
answer	9
images	[] (empty; text-only example)

Prompt formatting. To standardize inputs across training and evaluation, each `problem` is rendered through a Jinja2 template [32]. We bypass any model-provided chat template (`override_chat_template = null`) and inject the dataset field into `content`. The exact template used in this work is:

```
{{ content | trim }}
You FIRST think about the reasoning process as an internal
   monologue and then
provide the final answer. The reasoning process MUST BE enclosed
   within <think> </think> tags.
The final answer MUST BE put in \boxed{}.
```

Here, `content` receives the raw `problem` text (trimmed). The instructions enforce a two-part output: (i) an internal reasoning trace delimited by `<think>...</think>` (ignored by evaluators), and (ii) a canonical final answer inside $\boxed{\cdot}$. During scoring, only the boxed expression is parsed and compared to the ground truth (e.g., with SymPy-based equivalence checks [33]); generations without a valid `\boxed{...}` are marked incorrect. This scheme yields reproducible prompts and a robust extraction rule for evaluation and reward computation.
The dataset is large yet tractable for RL, diverse in problem types requiring multi-step reasoning, natively compatible with `EasyR1`, and widely adopted in reasoning-alignment work, enabling comparisons with prior art.

5.5 Baselines

To evaluate the effectiveness of our proposed intrinsic reward method, we established a set of baselines derived from the `Qwen2.5` model family. Rather than relying on external results, we deliberately reproduced the training of baseline models ourselves in order to validate the experimental pipeline and to convince ourselves that performance differences observed later are genuine and not artifacts of mismatched setups. This ensures that all comparisons are fair, consistent, and reproducible within the scope of this thesis.

- **Instruction-tuned base models:** The original Qwen2.5-Instruct checkpoints (1.5B, 3B, and 7B parameters), released by Alibaba Cloud [27, 28, 29, 30]. These models have undergone supervised instruction tuning but no reinforcement learning. They serve as the natural starting point for assessing the benefits of further RL fine-tuning.
- **GRPO-trained models on Math12k:** The same Qwen2.5-Instruct checkpoints, additionally fine-tuned with Group Relative Policy Optimization (GRPO) on the Math12k dataset. These runs were executed by us using the hyperparameters described in Section 5.1, with $n = 5$ rollouts per prompt and KL-regularization enabled. By training these models ourselves, we establish a controlled baseline that isolates the effect of reinforcement learning from larger-scale pretraining or third-party implementation details.

Table 3 summarizes the performance of these baselines across three benchmarks: the `Math12k` test split, and the widely used GSM8K benchmark in both *flexible extract* and *strict math* evaluation modes. The flexible evaluation counts semantically equivalent expressions as correct, while the strict setting requires exact symbolic equivalence.

Table 3: Baseline performance of Qwen2.5-Instruct models and their GRPO-trained variants with external reward on Math12k and GSM8K benchmarks. All GRPO results were reproduced within this thesis to ensure comparability and consistency.

Model	**Math12k test**	**GSM8K-flexible**	**GSM8K-strict**
Qwen2.5-1.5B-Instruct (base)	0.500	0.5800	0.1200
Qwen2.5-1.5B-Instruct (external)	0.600	0.6035	0.1926
Qwen2.5-3B-Instruct (base)	0.642	0.6100	0.0015
Qwen2.5-3B-Instruct (external)	0.698	0.5853	0.0038
Qwen2.5-7B-Instruct (base)	0.732	0.7741	0.3685
Qwen2.5-7B-Instruct (external)	0.7994	0.8226	0.8097

5.6 Evaluation Metrics

The effectiveness of reinforcement learning fine-tuning (RLFT) was assessed using both **task-level metrics** and **reward–signal diagnostics**. Together, these metrics allow us to evaluate not only end-task performance but also the validity of the proposed intrinsic reward.

Task accuracy. On both Math12k [31] and GSM8K, we report the fraction of problems for which the model output matches the ground-truth answer. For GSM8K, we consider two evaluation modes:

- *Flexible extract:* counts predictions correct if the extracted numerical answer matches, allowing for variations in formatting.
- *Strict math:* requires exact symbolic equivalence, evaluated with `SymPy` [33].

Reward–correctness alignment. To assess whether the proposed internal reward signal (d1+d2 hidden-state dynamics) is informative, we log intrinsic reward values for each rollout alongside correctness labels. We then compute standard classification metrics, treating the d1+d2 score as a predictor of correctness:

- **AUROC (Area Under the Receiver Operating Characteristic curve):** quantifies the probability that a randomly chosen correct rollout is assigned a higher d1+d2 score than a randomly chosen incorrect rollout. AUROC is insensitive to the absolute scale of the score and provides a global measure of separability. A value of 0.5 indicates random guessing, while 1.0 indicates perfect discrimination.
- **AUPR (Area Under the Precision–Recall curve):** evaluates the trade-off between precision (fraction of predicted correct rollouts that are truly correct) and recall (fraction of all correct rollouts that are identified). AUPR is especially informative in imbalanced settings where incorrect rollouts dominate, since it focuses on the model's ability to concentrate correctness among the highest-scoring samples.
- **FPR@95 (False Positive Rate at 95% True Positive Rate):** measures how many incorrect rollouts are included among the highest-scoring rollouts once 95% of correct rollouts are recovered. This metric is widely used in out-of-distribution detection and safety-critical evaluation, since it reflects the cost of misclassifying errors as "highly confident" outputs.

These metrics collectively quantify whether higher intrinsic scores correspond to a higher probability of correctness. AUROC provides a global view of discriminative power, AUPR captures performance under skewed class distributions, and FPR@95 emphasizes reliability under high-recall operating points. Using all three ensures that our evaluation of the intrinsic reward is robust to different practical perspectives on what "good correlation" means.

Training dynamics. We track KL divergence between the actor and reference model, log-probabilities of generated tokens, and rollout statistics (batch size, length, number of valid `\boxed{}` answers). These provide insight into stability and convergence.

Internal reward diagnostics. For selected runs, we additionally conduct a ranking analysis: with batch size 512 and rollout factor 8 (yielding 4096 rollouts per step), responses are sorted by their d1+d2 score. We then examine whether higher-ranked responses are more likely to be correct. This diagnostic is presented in Section 6.1 (Results). Together, these metrics provide a comprehensive view of both task-level performance and the informativeness of the proposed intrinsic reward.

6 Results

This chapter presents the empirical findings of our study. Building on the experimental setup described in Chapter 5, we now report the performance of the proposed intrinsic-reward method and compare it against baseline models. The results are organized to highlight both task-level improvements and internal reward diagnostics. We begin by analyzing performance across different model sizes on Math12k and GSM8K, followed by an investigation of the correlation between the intrinsic reward signal and rollout correctness. We then examine training dynamics and efficiency, and finally summarize the key insights. Together, these results provide evidence for the effectiveness of hidden-state–based intrinsic rewards in reinforcement learning fine-tuning of large language models.

6.1 Reward–Correctness Alignment

A central question in this thesis is whether the intrinsic reward signal derived from hidden-state dynamics (d1 and d2 values, as introduced in Section 3.3) is actually meaningful with respect to task performance. If the internal score assigned to a rollout is not correlated with its correctness, then training with such a signal would likely be ineffective. Conversely, if higher internal scores consistently correspond to more accurate solutions, this would provide strong evidence that hidden-state dynamics capture useful information about reasoning quality.

To investigate this, we first conducted a controlled experiment during **standard GRPO training with an external correctness-based reward**. We selected the Qwen2.5 models as evaluation candidates and collected metadata at step 23 of epoch 1 on the Math12k dataset. Although optimization was driven solely by the correctness-based reward, we instrumented the system to additionally compute d1 and d2 dynamics and log them alongside correctness for each rollout (see Listing 2 in the Appendix for an example entry). Crucially, these intrinsic signals did not affect training—they served purely as *observational probes*, allowing us to test whether they encode information about correctness independently of optimization. Correctness was determined by extracting the boxed final answer and comparing it against reference solutions using symbolic equivalence (SymPy [33]), ensuring that algebraically equivalent answers were accepted while formatting artifacts were ignored.

Note that while GRPO training itself used $n = 5$ rollouts per prompt (see Section 5.1), metadata collection employed $n = 8$ rollouts. This choice increased the diversity of responses available for analysis, without affecting the training procedure.

We then ranked all rollouts by their d1+d2 score. If hidden-state dynamics carry meaningful signals, responses with higher d1+d2 values should be more likely correct than those with lower values. As shown in Figures 6–8, this hypothesis held: correct rollouts clustered toward the top of the ranking, whereas incorrect rollouts dominated the lower tail. Quantitatively, AUROC and AUPR values (Table 4) confirmed that d1+d2 is predictive of correctness, with the strongest correlation in the 7B model. These results establish that hidden-state dynamics contain non-trivial, predictive information about reasoning quality when observed in an external-reward training regime.

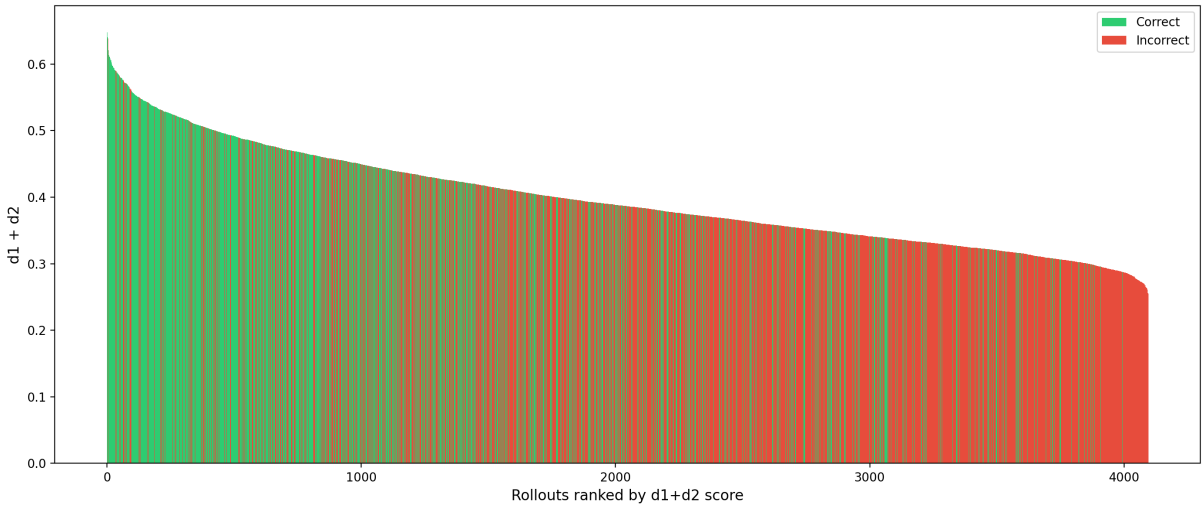

Figure 6: Qwen2.5-7B-Instruct rollouts collected at step 23 of epoch 1 under external correctness-based reward. Batch size: 512, rollout factor: 8, yielding 4096 rollouts.

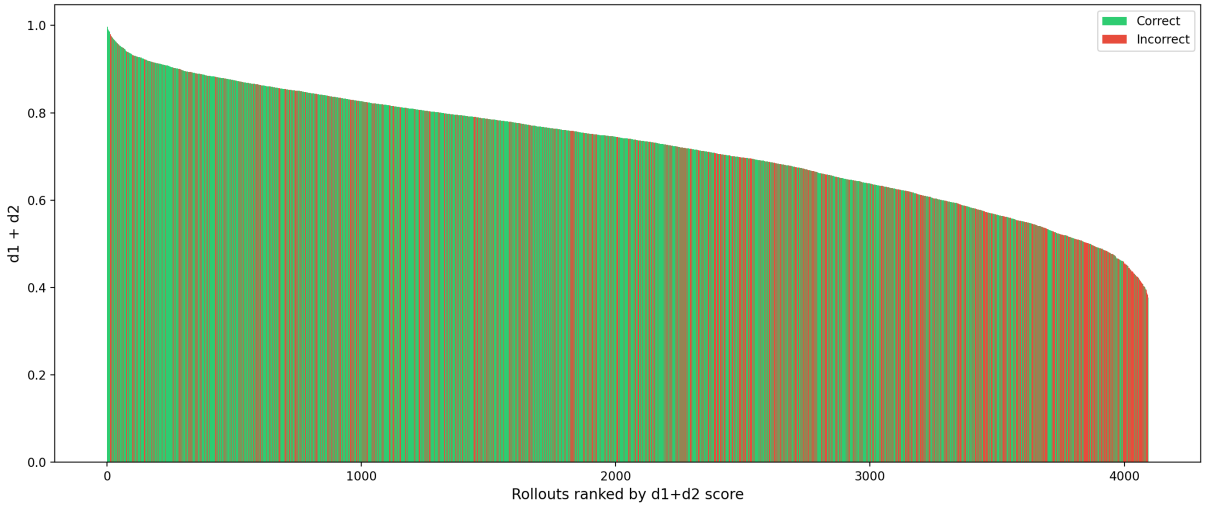

Figure 7: Qwen2.5-3B-Instruct rollouts collected at step 23 of epoch 1 under external correctness-based reward. Batch size: 512, rollout factor: 8, yielding 4096 rollouts.

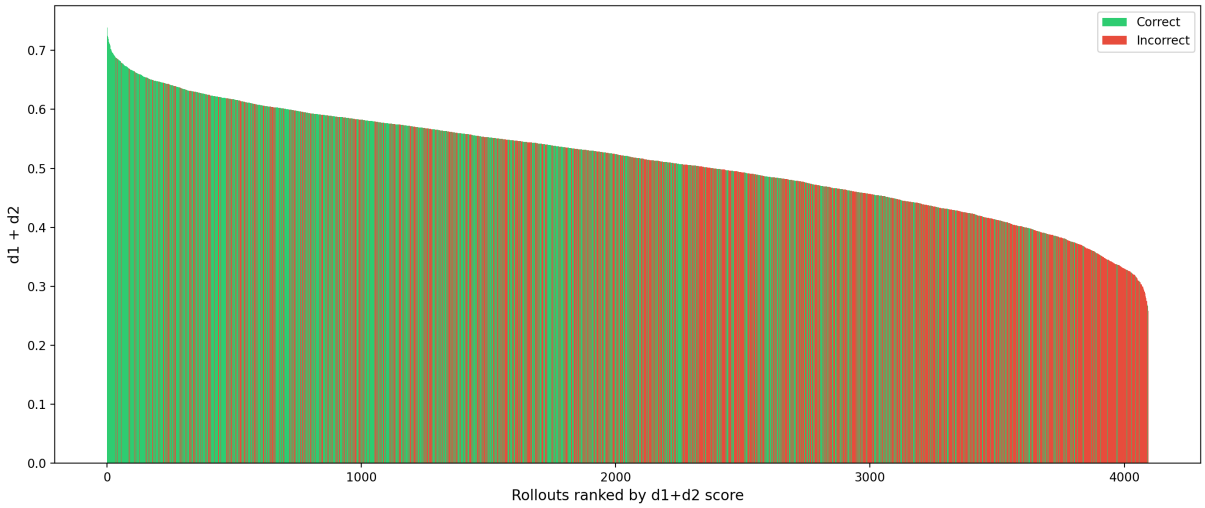

Figure 8: Qwen2.5-1.5B-Instruct rollouts collected at step 23 of epoch 1 under external correctness-based reward. Batch size: 512, rollout factor: 8, yielding 4096 rollouts.

Table 4: Reward–correctness alignment under external reward at step 23 of epoch 1. Values report AUROC, AUPR, and FPR@95 computed over 4096 rollouts. Arrows indicate whether higher or lower values are preferable.

Model	**AUROC ↑**	**AUPR ↑**	**FPR@95 ↓**
Qwen2.5-1.5B-Instruct (external)	0.72	0.69	0.79
Qwen2.5-3B-Instruct (external)	0.62	0.69	0.85
Qwen2.5-7B-Instruct (external)	0.78	0.68	0.74

Having validated the observational signal, we next turned to a more demanding test: **training directly with d1+d2 as the reward**. Here the intrinsic score fully replaced the external correctness-based reward, meaning that optimization was guided exclusively by hidden-state dynamics. We again trained all three Qwen2.5 models for one epoch on Math12k, and collected metadata at step 23. Figures 9–11 show the ranked rollouts after this intrinsic-reward training. For the smaller 1.5B and 3B models, the correlation between score and correctness largely collapsed: correct and incorrect rollouts appeared intermixed across the ranking, and quantitative metrics (Table 5) hovered around chance level. Only the 7B model preserved a meaningful signal, with AUROC 0.81 and AUPR 0.92, though optimization still failed to yield clear accuracy gains.

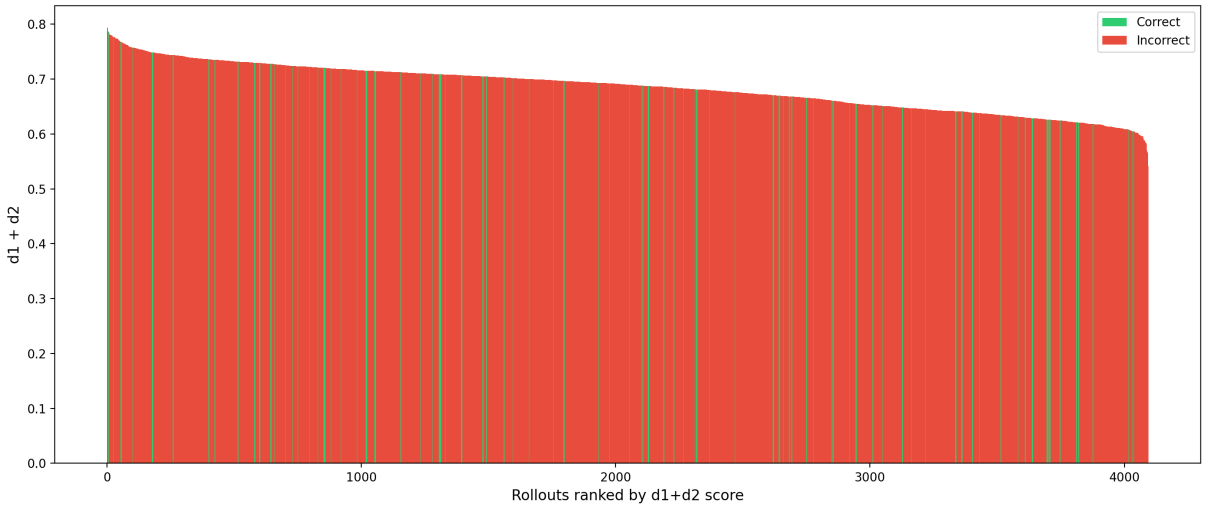

Figure 9: Qwen2.5-1.5B-Instruct after 1 epoch of training with d1+d2 as reward. Batch size: 512, rollout factor: 8, yielding 4096 rollouts. Correct rollouts are scattered throughout the ranking, showing weak correlation.

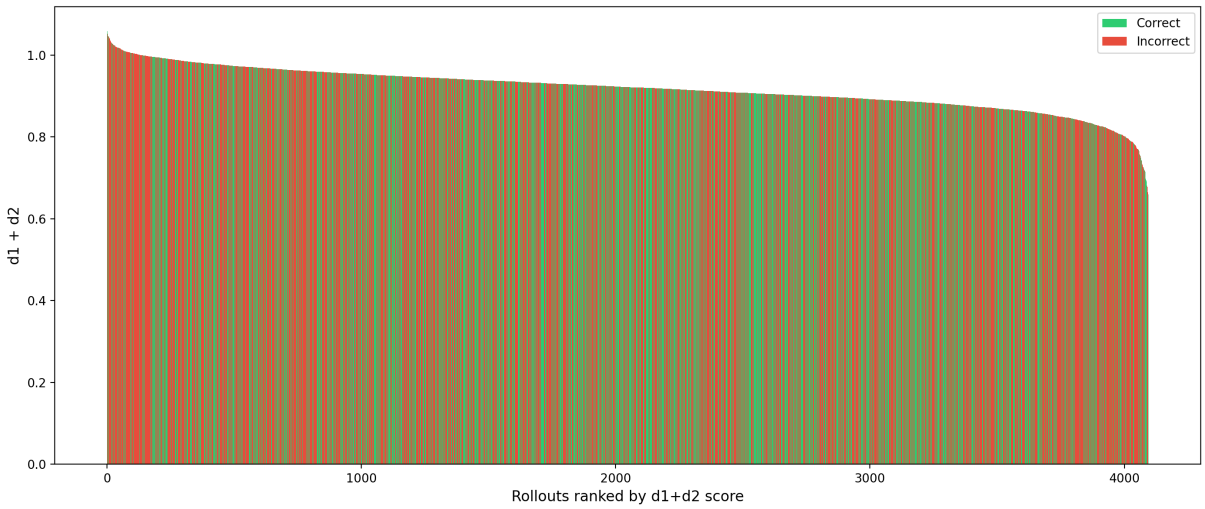

Figure 10: Qwen2.5-3B-Instruct after 1 epoch of training with d1+d2 as reward. Batch size: 512, rollout factor: 8, yielding 4096 rollouts. Correlation with correctness is minimal.

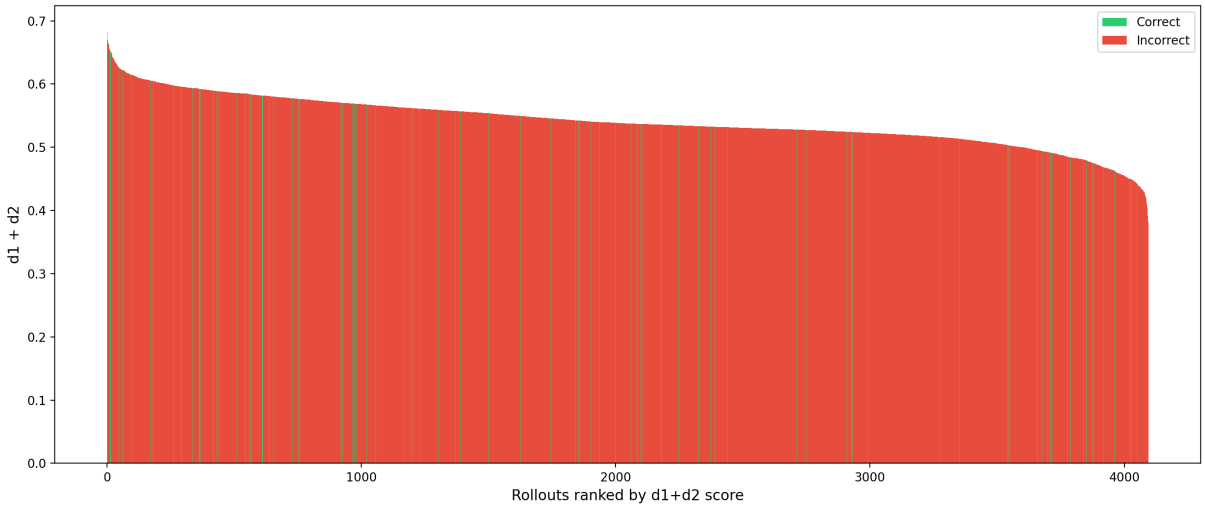

Figure 11: Qwen2.5-7B-Instruct after 1 epoch of training with d1+d2 as reward. Batch size: 512, rollout factor: 8, yielding 4096 rollouts. A stronger correlation emerges, but optimization remains unstable.

Table 5: Reward–correctness alignment when d1+d2 was used as the internal reward at step 23 of epoch 1. Values report AUROC, AUPR, and FPR@95 over 4096 rollouts.

Model	**AUROC ↑**	**AUPR ↑**	**FPR@95 ↓**
Qwen2.5-1.5B-Instruct (internal)	0.51	0.08	0.93
Qwen2.5-3B-Instruct (internal)	0.49	0.40	0.91
Qwen2.5-7B-Instruct (internal)	0.59	0.09	0.97

Taken together, these results reveal a nuanced picture. When observed passively in an external-reward regime, hidden-state dynamics correlate reliably with correctness across model scales. However, when used directly as the optimization signal, their effectiveness depends heavily on scale: smaller models fail to exploit the signal, while the 7B model shows partial but unstable gains. This divergence highlights the complexity of aligning hidden-state dynamics with learning, and suggests that while the signal is real and predictive, its direct use as a reward is not sufficient for stable fine-tuning.

6.2 Task-Level Performance with Intrinsic Reward

This section presents the results of reinforcement learning fine-tuning (RLFT) when training Qwen2.5-Instruct models of different sizes with intrinsic rewards ($d1 + d2$) compared to external correctness-based rewards. The experiments cover three model scales: 1.5B, 3B, and 7B parameters. For each model, we plot the trajectory of the intrinsic reward signal ($d1+d2$), the accuracy reward, and the format reward during training, and compare final task-level results against both base and external reward fine-tuned baselines.
Training the **1.5B model** with internal reward showed a consistent rise in the intrinsic signal $d1 + d2$ during training (Figure 12), suggesting that the optimization process

reinforced hidden-state dynamics. However, this gain did not translate into task-level correctness: the accuracy reward quickly collapsed to near-zero after a short increase at the beginning. The format reward also rapidly declined, stabilizing at negligible values. Due to the lack of progress, this run was stopped after **55 steps** (instead of the full 345), both to preserve computational resources and because the trajectory indicated that further training was unlikely to yield improvements. Task-level evaluation (Table 6) confirms this: the internal reward model performed substantially worse than both the base model and the external reward baseline.

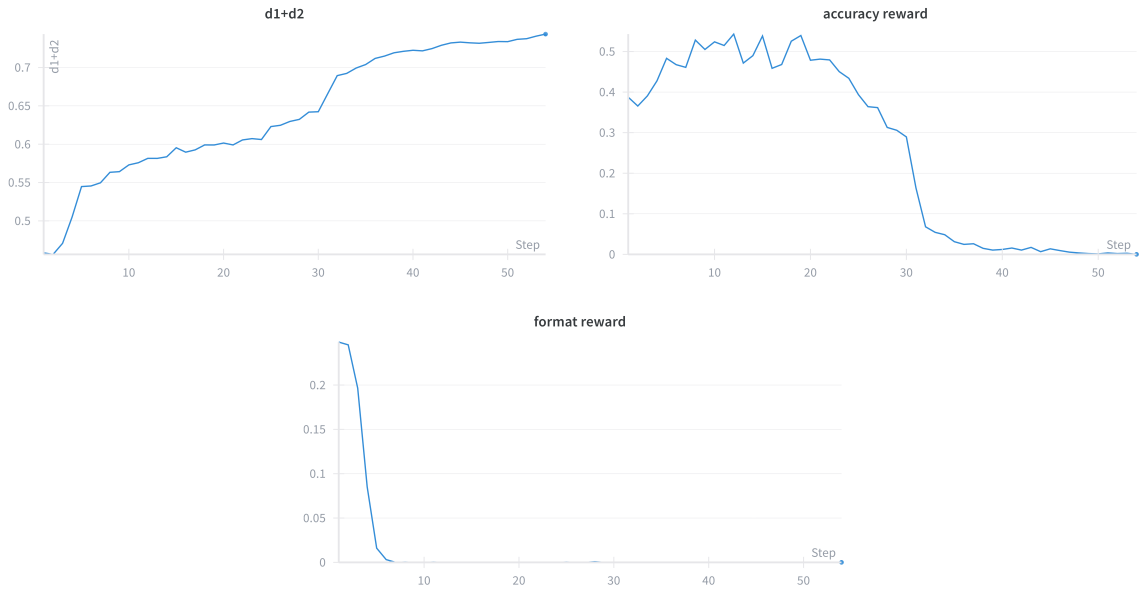

Figure 12: Qwen2.5-1.5B-Instruct trained with internal reward: evolution of $d1+d2$ (top-left), accuracy reward (top-right), and format reward (bottom).

The **3B model** presented a slightly more nuanced picture. The $d1 + d2$ score steadily increased and stabilized at a higher plateau than its initialization (Figure 13), showing that the model could optimize the intrinsic objective. Nevertheless, the accuracy reward consistently decreased and remained low throughout training, similar to the 1.5B case. The format reward fluctuated around small values without a clear upward trend. Given the lack of improvement, this experiment was stopped after **115 steps** (instead of the full 345). Evaluation again shows that the intrinsic reward model underperformed both the base and external reward models across all benchmarks.

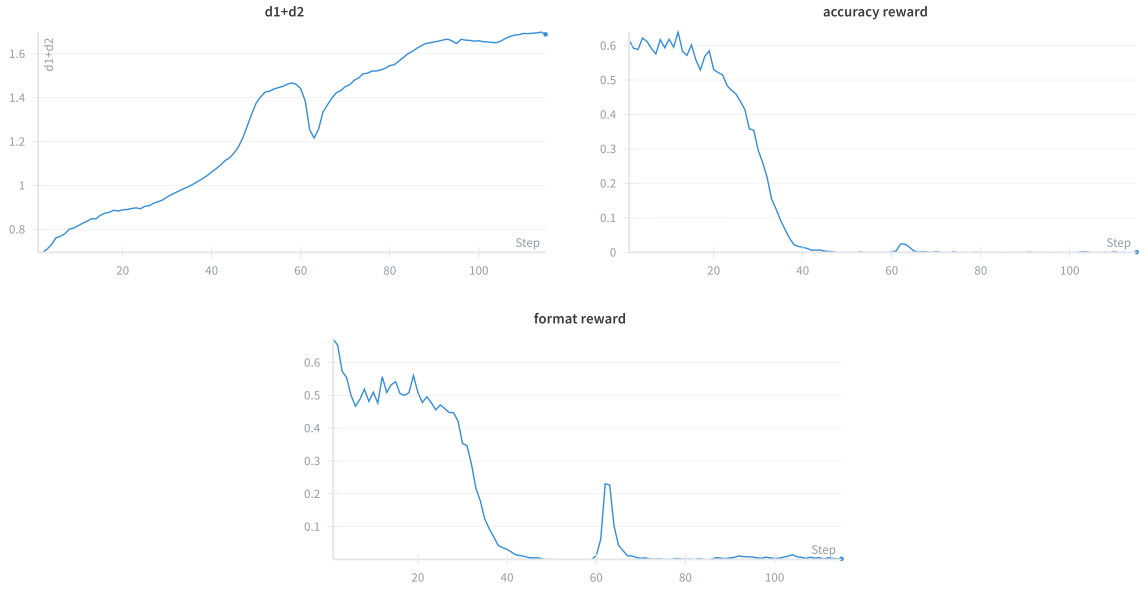

Figure 13: Qwen2.5-3B-Instruct trained with internal reward: evolution of $d1 + d2$ (top-left), accuracy reward (top-right), and format reward (bottom).

For the **7B model**, the divergence between intrinsic and external signals became most pronounced. The $d1 + d2$ reward exhibited a smooth and stable increase, reaching significantly higher levels compared to the smaller models (Figure 14). Accuracy reward initially rose above 0.5 but quickly decayed toward zero as training progressed. Similarly, the format reward showed an early spike but then collapsed to negligible values. Unlike the smaller models, the 7B run was completed for the full **345 steps**, to fully assess whether scale alone could make intrinsic-only training viable. While the task-level results (Table 6) show slightly better performance than the smaller intrinsic runs, the model still fails to match the external reward baseline. This indicates that, even at larger scale, optimizing hidden-state–based objectives alone does not guarantee correctness or formatting quality.

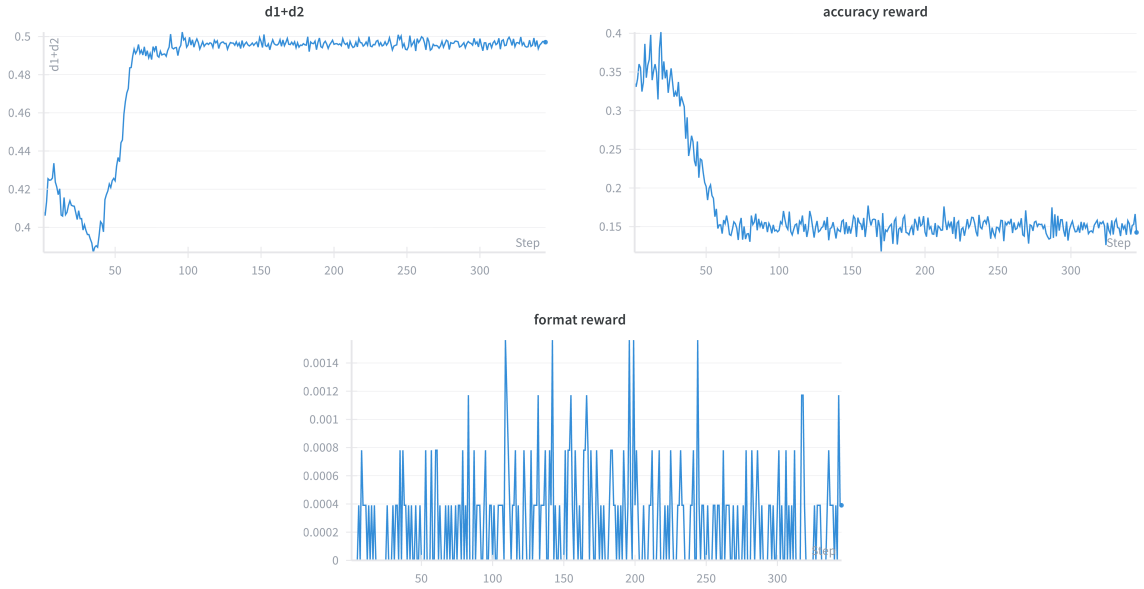

Figure 14: Qwen2.5-7B-Instruct trained with internal reward: evolution of $d1 + d2$ (top-left), accuracy reward (top-right), and format reward (bottom).

Across all scales, the $d1 + d2$ intrinsic signal was consistently optimizable, with higher-capacity models showing smoother and stronger growth. However, accuracy and format rewards consistently degraded when trained under intrinsic reward alone. These findings underscore the central limitation: while hidden-state dynamics encode valuable diagnostic information, their raw use as reinforcement signals fails to yield performance improvements.
An additional factor behind the observed degradation may lie in **formatting errors**. EasyR1's reward pipeline relies heavily on correct formatting (boxed answers and `<think>` tags) to assign accuracy labels. When training with internal rewards only, the model has no explicit incentive to preserve formatting and may therefore generate outputs that optimize the hidden-state trajectory but cannot be parsed as correct by the external grader. This explains why the $d1 + d2$ signal continues to improve, while accuracy and format scores collapse: the model learns to exploit the internal objective without producing valid task outputs.

To highlight this contrast, Table 6 reports the task-level results for each model under three settings: the **base model** (pretrained without RLFT), **external reward training** (GRPO with correctness + format reward, weighted as $0.9 \cdot \text{accuracy} + 0.1 \cdot \text{format}$), and **intrinsic reward training** (with $d1+d2$ only). All values are reported with two decimals for consistency.

Table 6: Task-level performance of Qwen2.5-Instruct models under different training regimes. Intrinsic runs were stopped early for 1.5B (55 steps) and 3B (115 steps) due to lack of improvement, while the 7B run was trained for the full 345 steps. Results are reported with two decimals.

Model / Training	**Math12k test**	**GSM8K-flexible**	**GSM8K-strict**
Qwen2.5-1.5B-Instruct (base)	0.50	0.58	0.12
Qwen2.5-1.5B-Instruct (external)	0.60	0.60	0.19
Qwen2.5-1.5B-Instruct (internal)	0.00	0.53	0.32
Qwen2.5-3B-Instruct (base)	0.64	0.61	0.00
Qwen2.5-3B-Instruct (external)	0.70	0.59	0.00
Qwen2.5-3B-Instruct (internal)	0.00	0.13	0.00
Qwen2.5-7B-Instruct (base)	0.73	0.77	0.37
Qwen2.5-7B-Instruct (external)	0.80	0.82	0.81
Qwen2.5-7B-Instruct (internal)	0.04	0.76	0.63

Overall, the results emphasize a paradox: intrinsic signals like $d1 + d2$ are readily optimizable and correlate with correctness in observational studies, yet when used in isolation they do not reliably improve task accuracy. This mismatch may partly stem from EasyR1's reliance on strict formatting, which internal rewards do not enforce. External reward training remains necessary to anchor correctness, while intrinsic rewards may serve as complementary diagnostics or auxiliary objectives for future hybrid strategies.

7 Conclusion and Future Work

This thesis set out to investigate the efficiency of reinforcement learning fine-tuning (RLFT) for large language models, with a specific focus on the use of hidden-state dynamics as an intrinsic reward signal. In recent years, reinforcement learning from human feedback (RLHF) and related methods have become central to aligning large language models with human preferences. However, these methods are computationally expensive, rely on high-quality external feedback, and suffer from inefficiencies that limit their scalability. Against this backdrop, we explored the hypothesis that internal signals already present in the model's hidden-state transitions could serve as a cheaper and more efficient proxy reward.

The first part of the work established a solid experimental foundation. We implemented reinforcement learning experiments on multiple HPC clusters, using Qwen2.5-Instruct models of different scales (1.5B, 3B, and 7B parameters). Our experimental setup included detailed hyperparameter management, careful dataset preparation (Math12k and GSM8K), and robust baselines established through GRPO training with external correctness-based reward. In this external setup, rewards were computed as a weighted combination of accuracy (90%) and format (10%), ensuring that both the correctness of the final answer and its parseability were preserved. These steps ensured that the experiments were reproducible, well-controlled, and grounded in existing benchmarks.

We then turned to the central research question: do hidden-state dynamics, in particular the d1 and d2 signals extracted from intermediate activations, correlate with task performance? Through a detailed analysis of metadata logged during GRPO training, we demonstrated that higher d1+d2 values were indeed associated with a higher probability of correctness. This alignment was visible both in visual analyses of ranked rollouts and in quantitative metrics such as AUROC, AUPR, and FPR@95. These findings provided strong evidence that hidden-state dynamics encode non-trivial information about reasoning quality. Importantly, these results were consistent across model scales in the metadata analysis, with the larger Qwen2.5-7B-Instruct model showing the strongest correlation. However, when applied as a reward signal for training, even the 7B model did not reach stable convergence: intrinsic signals kept rising smoothly but without translating into stable task-level improvements.

When we attempted to replace the external reward with the intrinsic d1+d2 reward in GRPO training, the results were less encouraging. Despite the observed correlations, task-level accuracy on Math12k and GSM8K did not improve. In some cases, performance even decreased. This highlights a key lesson: correlation between an internal signal and correctness does not guarantee that the signal can serve as a useful training objective. The intrinsic reward signal, while promising as a diagnostic, was not sufficient in its raw form to drive meaningful learning progress when used alone. One additional contributing factor was formatting: since EasyR1 relies on strict output formats (boxed answers and `<think>` tags) to assign correctness, models trained solely on intrinsic signals often abandoned these conventions, further reducing measured accuracy. This negative result is an important contribution, as it sheds light on the limitations of naive intrinsic reward approaches and clarifies the gap between observational alignment and optimization.

In summary, the thesis achieved several important goals:

- Established a reproducible experimental pipeline for RLFT on Qwen2.5-Instruct models.
- Demonstrated that hidden-state dynamics (d1+d2) correlate with rollout correctness.
- Validated these correlations through both visual inspection and quantitative metrics.
- Tested the feasibility of using intrinsic rewards directly for training, and identified their limitations in improving task-level performance.

Although the experiments did not show clear gains when replacing external rewards with intrinsic signals, the findings point toward several avenues for future work. If more time and resources were available, the next steps would focus on making the intrinsic signal more usable for training through **reward shaping**.

One promising approach is to exploit the ranking structure of d1+d2 scores. In the current formulation, all rollouts received continuous values that were directly used as rewards. However, rollouts in the middle of the distribution often provide diluted or noisy information, since they neither represent the clearly correct nor the clearly incorrect behaviors. Instead of using the full distribution, future work could focus on the extremes: the top-x percentile of rollouts with the highest d1+d2 scores, and the bottom-y percentile with the lowest scores. Rollouts in between could be ignored, reducing noise in the training signal.

In addition, discretizing the reward could help stabilize optimization. Rather than assigning small continuous differences, the system could map the top-x percentile of rollouts to a reward of 1.0 and the bottom-y percentile to 0.0. This binary scheme would create a clearer separation between desirable and undesirable outputs. Such shaping strategies have been shown in other reinforcement learning domains to reduce variance and help the policy focus on the most informative examples. Applied here, they could turn the observed correlation between d1+d2 and correctness into a stronger driver of learning.

Another direction is to combine intrinsic and external rewards. While the d1+d2 signal alone was insufficient, it may still carry complementary information that could be valuable when balanced against correctness-based signals. A hybrid reward could, for instance, encourage exploration of reasoning strategies (via d1+d2) while maintaining fidelity to ground truth answers (via correctness reward). This kind of multi-objective reward shaping may bridge the gap between correlation and utility.

Finally, future research should also consider scaling both the dataset and the model. The Math12k dataset, while convenient and directly compatible with EasyR1, is small compared to the complexity of real-world reasoning tasks. Larger datasets with more diverse reasoning problems may provide stronger training signals and allow intrinsic rewards to demonstrate their potential. Similarly, training with larger models beyond 7B parameters could reveal whether the signal becomes more actionable at scale.

A further promising line of research involves **data filtering**. Since reinforcement learning fine-tuning is highly data-hungry, strategies to achieve comparable or even better accuracy with fewer training samples would be of great practical importance. One approach is to selectively filter prompts and rollouts before training, prioritizing those that provide the most informative signal. For instance, prompts where the model's hidden-state dynamics

diverge sharply could be oversampled, while uninformative or redundant cases could be downsampled. In this way, the same—or higher—performance might be achieved with a fraction of the data, reducing training cost and increasing efficiency. Such filtering could be combined with percentile-based reward shaping, ensuring that only the most useful training examples are reinforced. This approach directly addresses the inefficiencies documented in this thesis and offers a practical path toward less data-hungry alignment methods.

Overall, this thesis has taken a step toward understanding the role of hidden-state dynamics in reinforcement learning fine-tuning for large language models. The findings highlight both promise and limitations: the d1+d2 signals clearly correlate with correctness but do not, in their raw form, translate into performance improvements when used as a reward. This dual outcome provides clarity for the field. It shows where current approaches fall short, while also pointing to concrete strategies—such as reward shaping, discrete reward assignment, hybrid signals, and data filtering—that could unlock the potential of intrinsic rewards in the future. In this sense, the work not only documents the challenges of efficient RLFT but also lays the groundwork for the next generation of methods that aim to make alignment cheaper, more scalable, and more robust.

A Appendix

A.1 Training Hyperparameters

The following listing shows the main hyperparameter configuration used for training the Qwen2.5-Instruct models with GRPO and intrinsic rewards (magnitude + acceleration of hidden-state dynamics). As shown in Listing 1, this configuration defines dataset splits, rollout parameters, optimization settings, and metadata generation frequency. It served as the basis for all reinforcement learning experiments described in Chapter 5.

```yaml
data:
  train_files: hiyouga/math12k@train
  val_files: hiyouga/math12k@test
  rollout_batch_size: 512
  val_batch_size: 1024
  max_prompt_length: 2048
  max_response_length: 2048

algorithm:
  adv_estimator: grpo
  kl_coef: 1.0e-2
  disable_kl: false
  use_kl_loss: true

worker:
  actor:
    global_batch_size: 128
    micro_batch_size_per_device_for_update: 4
    optim:
      lr: 1.0e-6
      weight_decay: 1.0e-2
      strategy: adamw
  rollout:
    n: 5   # training rollouts
    temperature: 1.0
    top_p: 0.99

reward:
  reward_strategy: "d1_plus_d2"

trainer:
  total_epochs: 15
  val_freq: 5
  save_freq: 5
  save_metadata: true
  metadata_save_freq: 1
```

Listing 1: Excerpt of training configuration

A.2 Example Metadata Entry

During training, metadata was stored in JSON format. Each entry corresponds to one prompt, with multiple responses (rollouts), rewards, and correctness labels. These metadata entries were the basis for the reward–correctness alignment analyses in Chapter 6. An excerpt is provided in Listing 2, showing a single prompt with eight responses from the Qwen2.5-3B-Instruct model at step 1.


```
{
  "step": 1,
  "prompt_index": 0,
  "rollout_n": 8,
  "prompt": {
    "text": "Pizzas are sized by diameter... Provide reasoning
       in <think>...</think> and final answer in \\boxed{}.",
    "question": "Pizzas are sized by diameter. What percent
       increase in area results..."
  },
  "ground_truth": { "text": "44%", "available": true },
  "responses": [
    {
      "response_index": 0,
      "text": "... \\boxed{44}",
      "rewards": {
        "overall": 0.655,
        "d1_mean": 0.307,
        "d2_mean": 0.348,
        "angular_coe": 0.173,
        "combined_coe": 0.287,
        "accuracy": 1.0,
        "format": 1.0
      }
    },
    {
      "response_index": 1,
      "text": "... \\boxed{44%}",
      "rewards": {
        "overall": 0.757,
        "d1_mean": 0.353,
        "d2_mean": 0.404,
        "angular_coe": 0.177,
        "combined_coe": 0.329,
        "accuracy": 1.0,
        "format": 1.0
      }
    },
    ...
  ],
  "model_info": {
```

```
    "model_name": "Qwen2.5-3B-Instruct"
  }
}
```

Listing 2: Excerpt from metadata.json for Qwen2.5-3B-Instruct, step 1 with 8 rollouts.

B Electronic appendix

Data, code repository, training runs, trained models and figures are provided in electronic form.

List of Figures

List of Tables

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[2] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 27 730–27 744.

[3] G. Cui, Y. Gu, Y. Zhang, Y. Dong, and J. Tang, "Self-rewarding language models," *arXiv preprint arXiv:2501.00001*, 2025.

[4] F. Luo, T. Zhang, Z. Liu, and M. Sun, "Intrinsic rewards for large language models," *arXiv preprint arXiv:2502.00002*, 2025.

[5] J. Bai, Y. Dong, L. Hou, N. Ding, Y. Gu, Z. Liu, and M. Sun, "Qwen: Open-source large language models by alibaba cloud," *arXiv preprint arXiv:2307.00075*, 2023.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[8] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[9] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," in *International Conference on Learning Representations (ICLR)*, 2020.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[11] Y. Guo *et al.*, "Scaling laws for reasoning in large language models," *arXiv preprint arXiv:2501.01234*, 2025.

[12] X. Hu *et al.*, "Reinforcement learning for reasoning in llms," *arXiv preprint arXiv:2501.02345*, 2025.

[13] A. Jaech *et al.*, "Towards reasoning-centric reinforcement learning for llms," *arXiv preprint arXiv:2412.04567*, 2024.

[14] Z. Zhang *et al.*, "100 days after deepseek-r1: A survey of reasoning-focused reinforcement learning for llms," *arXiv preprint arXiv:2503.01234*, 2025.

[15] T. Gao *et al.*, "Rl with verifiable rewards for large language models," *arXiv preprint arXiv:2502.05678*, 2025.

[16] L. Liu *et al.*, "Nover: Reinforcement learning without explicit verifiers," *arXiv preprint arXiv:2502.07891*, 2025.

[17] H. Sun *et al.*, "Ktae: Key-token advantage estimation for rl with verifiable rewards," *arXiv preprint arXiv:2503.03456*, 2025.

[18] Y. Zhao *et al.*, "Intuitor: Reinforcement learning from internal feedback," *arXiv preprint arXiv:2503.05678*, 2025.

[19] J. Li *et al.*, "Rlsc: Reinforcement learning via self-confidence," *arXiv preprint arXiv:2503.07812*, 2025.

[20] R. Agarwal *et al.*, "Entropy minimization for intrinsic reward learning in llms," *arXiv preprint arXiv:2503.08901*, 2025.

[21] J. Baur *et al.*, "Visulogic: Verification-free reasoning in llms," *arXiv preprint arXiv:2504.01234*, 2025.

[22] D.-A. Team, "Deepseekmath: Scaling open-source math reasoning with group relative policy optimization," *arXiv preprint arXiv:2402.03300*, 2024.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," in *arXiv preprint arXiv:1707.06347*, 2017.

[24] Leibniz Supercomputing Centre, "High performance computing at lrz," 2025, accessed: September 2025. [Online]. Available: https://doku.lrz.de/high-performance-computing-10613431.html

[25] Karlsruhe Institute of Technology, "Horeka supercomputer user documentation," 2025, accessed: September 2025. [Online]. Available: https://www.nhr.kit.edu/userdocs/horeka/

[26] Friedrich-Alexander-Universität Erlangen–Nürnberg, "Hpc clusters at fau," 2025, accessed: September 2025. [Online]. Available: https://hpc.fau.de/systems-services/documentation-instructions/clusters/

[27] Alibaba Cloud, "Qwen2.5 llm: Extending the boundary of llms," 2025, accessed: September 2025. [Online]. Available: https://www.alibabacloud.com/blog/qwen2-5-llm-extending-the-boundary-of-llms_601786

[28] ——, "Qwen2.5-1.5b-instruct," 2025, accessed: September 2025. [Online]. Available: https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct

[29] ——, "Qwen2.5-3b-instruct," 2025, accessed: September 2025. [Online]. Available: https://huggingface.co/Qwen/Qwen2.5-3B-Instruct

[30] ——, "Qwen2.5-7b-instruct," 2025, accessed: September 2025. [Online]. Available: https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

[31] hiyouga, "Math12k," 2024, accessed: September 2025. [Online]. Available: https://huggingface.co/datasets/hiyouga/math12k

[32] P. Projects, "Jinja2 templating engine," 2025, accessed: September 2025. [Online]. Available: https://jinja.palletsprojects.com/en/stable/

[33] SymPy Development Team, "Sympy: Symbolic mathematics in python," 2025, accessed: September 2025. [Online]. Available: https://www.sympy.org/en/index.html