Dyve: Thinking Fast and Slow for Dynamic Process Verification

Jianyuan Zhong*, **Zeju Li***, **Zhijian Xu, Xiangyu Wen, Qiang Xu**†
The Chinese University of Hong Kong
{jyzhong, zjli24, zjxu21, xywen22, qxu}@cse.cuhk.edu.hk

Abstract

Large Language Models (LLMs) have advanced significantly in complex reasoning, often leveraging external verifiers to improve multi-step process reliability. However, existing process verification methods face critical limitations: discriminative Process Reward Models (PRMs) often provide overly simplistic binary feedback and struggle with incomplete reasoning traces, while sophisticated Generative Reward Models (GenRMs) can be computationally expensive. Furthermore, curating quality supervision data for process verifier is of challenging. Therefore, we present Dyve, a dynamic process verifier that enhances reasoning error detection in LLMs by integrating fast (*System 1*) and slow (*System 2*) thinking, inspired by Kahneman's Systems Theory. Dyve adaptively applies immediate token-level confirmation for straightforward steps and comprehensive analysis for complex ones. To address data challenges and enable its adaptive fast and slow thinking, Dyve employs a novel step-wise consensus-filtered supervision strategy. This strategy leverages Monte Carlo estimation, LLM-as-a-Judge, and specialized reasoning models to extract the high-quality training signals from noisy rollouts. Experimental results on ProcessBench and the MATH dataset confirm that Dyve significantly outperforms existing process-based verifiers and boosts performance in Best-of-N settings, while maintaining computational efficiency through strategic resource allocation. Our code, data and model are released at: https://github.com/staymylove/Dyve

1 Introduction

Large Language Models (LLMs) have significantly enhanced their reasoning capabilities by shifting from rapid, intuitive System 1 responses to more deliberate, extended System 2 thinking (Team et al., 2025; Arrieta et al., 2025; Guo et al., 2025). While enabling more complex problem-solving in math and scientific reasoning, this has also introduced new challenges in process verification, particularly in the reliable evaluation of multi-step and potentially incomplete reasoning traces.

Existing process verification methods, while crucial, exhibit notable limitations when applied to these advanced reasoning tasks. Discriminative Process Reward Models (PRMs), for example, are essential for detecting errors but often provide overly simplistic "System 1-like" binary feedback (yes/no predictions) for each step (Lightman et al., 2023b; Zhang et al., 2025). This approach is often insufficient for capturing nuanced errors in complex reasoning and can struggle with the reliable assessment of incomplete traces. Conversely, while more sophisticated Generative Reward Models (GenRMs) can offer deeper, System 2-style analytical feedback, they tend to be computationally expensive, limiting their practical applicability for step-wise verification.

Furthermore, a pervasive challenge across all current verification approaches is the reliance on supervision signals. High-quality human annotations for process supervision (Lightman et al., 2023b) are prohibitively expensive, and automatically generated labels, often derived from Monte Carlo estimation methods (Wang et al., 2024; Luo et al., 2024), can be noisy and weak (Zhang et al., 2025).

Our work introduces Dyve (**Dy**namic Process **Ve**rifier), a specialized generative process verification model that dynamically detects process errors using fast and slow thinking, inspired by Kahneman's Systems Theory (Kahneman, 2012). For any given reasoning trace, Dyve adaptively applies either ***System 1*** verification, supplying immediate, single-token confirmation for clearly correct and straightforward steps, or ***System 2*** analysis, performing a comprehensive, deeper evaluation for

* Equal contribution.
† Corresponding author.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 22332–22345
November 4-9, 2025

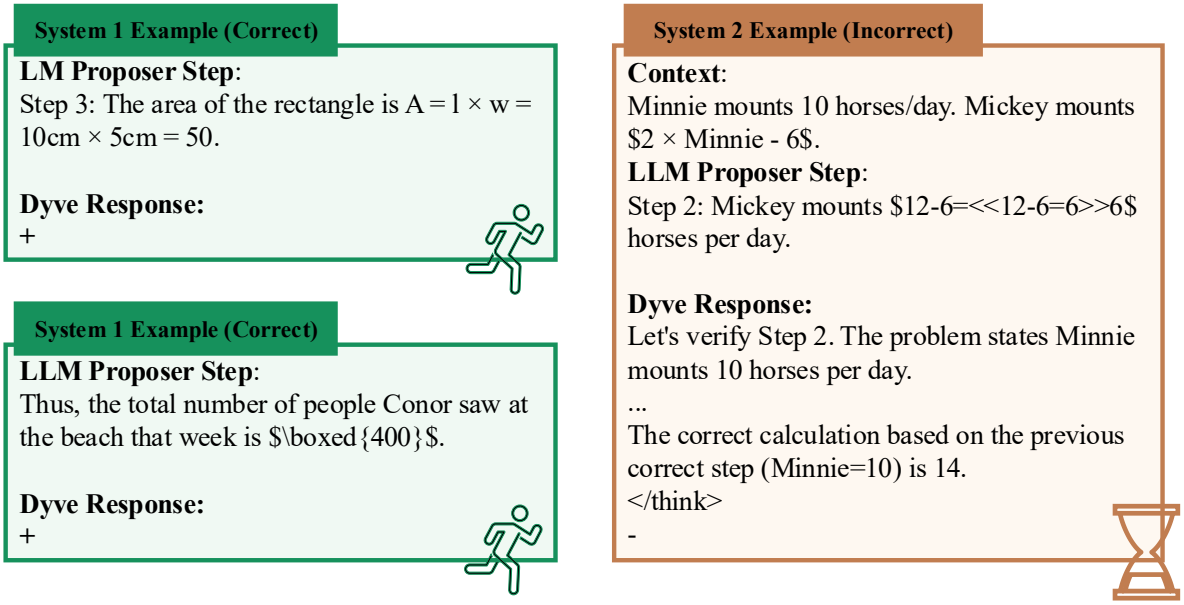

Figure 1: Dyve demonstrating its learned adaptability by dynamically selecting its verification strategy. For simple, correct steps (**left**), it provides concise System 1 confirmation using a '+' token, which signifies the step is correct. For more complex or incorrect steps (**right**), it employs detailed System 2 analysis, including an explanatory refutation, and concludes with a verdict token, such as '-' which signifies the step is incorrect.

complex or potentially erroneous ones. This dual-system approach allows Dyve to strategically allocate computational resources, balancing verification depth with efficiency.

Crucially, to enable this adaptive fast and slow thinking and overcome the limitations of costly or noisy supervision, we introduce a novel ***step-wise consensus-filtered process supervision*** technique. This method leverages Monte Carlo estimation to generate multiple rollouts per query, then uses an LLM-as-a-Judge (Gu et al., 2024) to assess the full reasoning trace, and finally employs a powerful reasoning LLM for detailed step-by-step analysis to flag steps that require System 2 verification. Through this pipeline, we curate approximately 117K high-quality training examples from 1.2M noisy Monte Carlo rollouts. Our findings underscore that the quality of supervision data, rather than mere quantity, is paramount for effectively training a robust and adaptive process-based verifier like Dyve.

Our contributions include:

- We introduce Dyve, a dynamic process verification framework that combines rapid System 1 validation with comprehensive System 2 correction and context-aware error recovery.
- We propose a novel ***step-wise consensus-filtered process supervision*** technique that employs Monte Carlo estimation and an LLM-as-a-Judge to reliably label each reasoning step.
- We demonstrate empirically that Dyve substantially outperforms existing PRMs and LLM-as-judges baselines on the ProcessBench benchmark and significantly boosts the performance of Proposer LLMs in Best-of-N settings on MATH-500.

2 Related Work

2.1 The Spectrum of Process Verification: Speed vs Depth

Verifying the multi-step reasoning of LLMs is crucial, but current verifiers often trade speed for analytical depth. At one end, Outcome Reward Models (ORMs) (Cobbe et al., 2021b; Yang et al., 2024) evaluate only final answers, offering high speed but no insight into the reasoning process. Process Reward Models (PRMs) (Lightman et al., 2023a; Zhang et al., 2025; Wang et al., 2024) improve on this by providing rapid, "System 1-like" binary feedback per step. However, their simplicity can hinder nuanced error diagnosis and struggle with incomplete reasoning traces. At the other end of the spectrum, Generative Reward Models (GenRMs) (Zhang et al., 2024b) offer "System 2-like" depth with detailed explanatory judgments (Wei et al., 2022), but are computationally intensive, and their verbose outputs are not always ideal for efficient, iterative verification.

Dyve aims to overcome this speed-versus-depth trade-off. Inspired by Kahneman's dual-process

Systems Theory (Kahneman, 2012), Dyve dynamically adapts its strategy. It employs rapid, PRM-like System 1 token-level confirmation for straightforward steps and a more comprehensive, GenRM-like System 2 analysis for complex or potentially erroneous ones. This learned adaptability optimizes both verification accuracy and resource allocation but necessitates highly specialized supervision signals.

2.2 Supervision for Process Verifiers

The efficacy of process verifiers, especially adaptive ones like Dyve, depends on training data quality. Human annotations (e.g., PRM800k (Lightman et al., 2023b)) offer high fidelity but are expensive and slow to scale. Consequently, automated data generation using Monte Carlo (MC) estimation or MCTS-based exploration is common (Luo et al., 2024; Wang et al., 2024; Zhang et al., 2024a), but often yields noisy or misleading labels (Zhang et al., 2025). Recent efforts use consensus mechanisms or LLM-as-a-Judge filtering (Gu et al., 2024; Zhang et al., 2025) to improve label quality, typically at the trace level.

Our step-wise consensus-filtered process supervision technique is designed to create the specialized data Dyve needs. This multi-stage pipeline first generates reasoning traces using Monte Carlo estimation and filters them at the trace level with an LLM-as-a-Judge. Subsequently, a powerful reasoning LLM meticulously analyzes each step, flagging it to generate distinct training signals for Dyve's System 1 (rapid confirmation) and System 2 (detailed analysis) modes. This process is crucial for enabling Dyve's unique adaptive verification capabilities without expensive human annotation.

3 Method

Dyve is designed to dynamically adapt its verification strategy. It distinguishes between rapid, intuitive checks (System 1) for straightforward reasoning steps and more deliberate, analytical evaluations (System 2) for complex or potentially erroneous steps.

3.1 Core Principle: Learned Adaptive Verification via Dual Output Modes

Dyve's core principle is a **learned adaptive verification** mechanism that employs a dual-output strategy inspired by fast and slow thinking. For straightforward reasoning steps, Dyve activates its **System**

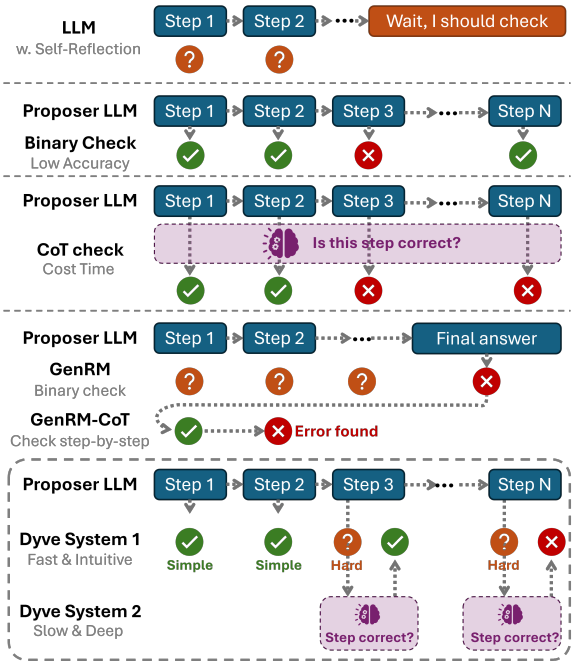

Figure 2: (1) LLM self-reflection is unreliable (2) Binary verification lacks depth, (3) Chain-of-Thought (CoT) verification is deeper but more expensive, (4) GenRM with CoT combines generation and verification without step-wise assessment, (5) Dyve, our proposed framework that dynamically combines fast System 1 and deep System 2 verification.

1 Output Mode, providing a rapid, low-latency confirmation, typically a single predefined token like "+" to signal correctness. However, when faced with complex, uncertain, or erroneous steps, Dyve shifts to its **System 2 Output Mode**. In this mode, it generates a comprehensive analytical response, offering a detailed natural language explanation enclosed in special `<think>...</think>` tags, followed by a final verdict token ("+" or "-").

Crucially, this dynamic selection between fast and deep verification isn't governed by hard-coded rules or heuristics. Instead, Dyve **learns to choose and generate the appropriate output mode** based on the input reasoning trace entirely through supervised fine-tuning (SFT). This allows Dyve to strategically allocate computational resources, using quick checks for simple steps and more intensive analysis for complex ones.

3.2 The Step-wise Consensus-Filtered Process Supervision Pipeline

Our multi-stage data pipeline is designed to curate training instances that teach Dyve its adaptive verification capabilities. Given a problem P and a sequence of reasoning steps $\{s_1, s_2, \ldots, s_T\}$ generated by a Proposer LLM, our pipeline associates

each step s_t with a target verification output, v_t. This output v_t manifests in one of two forms corresponding to Dyve's dual output modes: either a System 1 (fast) verification (v_t^{S1}), which is a concise token-level confirmation (e.g., +) for correct and straightforward steps, or a System 2 (slow) verification (v_t^{S2}), which is a detailed natural language explanation of s_t's correctness or error followed by a final verdict token.

The pipeline transforms raw reasoning traces into structured training data examples. For a complete reasoning trace of T steps that is entirely correct and composed of straightforward steps, the curated training example, denoted $\mathcal{E}_{\text{S1-complete}}$, takes the form:

$$\mathcal{E}_{\text{S1-complete}} = (P, \{s_1, v_1^{\text{S1}}, s_2, v_2^{\text{S1}}, \ldots, s_T, v_T^{\text{S1}}\}). \quad (1)$$

Conversely, if a trace proceeds with System 1 verifications but encounters an error at step $s_{t'}$, where $1 \leq t' \leq T$, the curated training example, $\mathcal{E}_{\text{S2-diverge}}$, is structured as:

$$\mathcal{E}_{\text{S2-diverge}} = (P, \{s_1, v_1^{\text{S1}}, \ldots, s_{t'-1}, v_{t'-1}^{\text{S1}}, s_{t'}, v_{t'}^{\text{S2}}\}). \quad (2)$$

In this scenario, the reasoning sequence for training purposes concludes at step t'.

Initial Rollout Generation. We begin by collecting 15K query-problem pairs from established mathematical reasoning datasets, primarily GSM8K (Cobbe et al., 2021a) and MATH (Hendrycks et al., 2021). To generate diverse reasoning traces (rollouts) for each query, we employ an automated process supervision technique based on Monte Carlo Tree Search (MCTS), leveraging OmegaPRM (Luo et al., 2024). OmegaPRM utilizes a divide-and-conquer MCTS algorithm to efficiently explore the solution space. Within this MCTS, partially completed reasoning paths (states s) are explored, and a selection strategy, such as an Upper Confidence Bound (UCB) score, guides the search towards promising or under-explored paths:

$$U(s) = c_{puct} \frac{\sqrt{\sum_i N(s_i)}}{1 + N(s)}, \quad (3)$$

where $N(s)$ is the visit count of state s, $\sum_i N(s_i)$ represents the total visits to sibling or relevant nodes, and c_{puct} is an exploration constant. This MCTS-driven approach (detailed in Appendix A.3) allows us to generate approximately 20 rollouts per query. This corpus is augmented with open-source PRM data from MathShepherd (Wang et al., 2024) and RLHFlow, while PRM800k (Lightman et al., 2023b) is excluded to prevent data leakage. This stage yields approximately 1.2 million initial rollouts, each containing step-by-step reasoning along with potentially noisy process labels (e.g., initial error step hypotheses from OmegaPRM).

Consensus Filtering and Rebalancing. To reduce noise from the initial rollouts, we employ DeepSeek V3 as an LLM-as-a-Judge. It verifies the initial error steps identified by OmegaPRM within each full rollout. Rollouts where the judge confirms the error assessment are retained; contradictory are discarded. This filtering process removes approximately 50% of the noisy rollouts. Subsequently, the dataset is rebalanced to ensure an appropriate distribution of positive (correct) and negative (incorrect) step examples, resulting in a curated set of approximately 117K high-quality reasoning traces.

System 1 and System 2 Target Output Generation. This pivotal stage creates the distinct training signals for Dyve's dual output modes from the filtered, high-fidelity traces. For correct steps (requiring System 1 thinking), Dyve is trained to provide simple token-level confirmation (e.g., the + token), which constitutes v_t^{S1}. For steps identified as incorrect or requiring deeper analysis (System 2 thinking), Dyve is trained using detailed error explanations. These detailed System 2 explanations are wrapped within `<think>...</think>` tags and a concluding verdict token `-`, are generated by the `DeepSeek-R1-Distill-Qwen-32B` model and constitute v_t^{S2}. The output of this pipeline is the final training dataset. The resulting data example is illustrated in Appendix A.4.

3.3 Dyve Training

We select `DeepSeek-R1-Distill-Qwen-14B` as the base model for Dyve and train it using Supervised Fine-Tuning (SFT) on the dataset curated as described in Section 3.2. For each training example, $\mathcal{E}_{\text{S1-complete}}$ (Eq. 1) or $\mathcal{E}_{\text{S2-diverge}}$ (Eq. 2), Dyve is tasked with generating the target verification output (v_t^{S1} or v_t^{S2}) for a given step s_t, conditioned on the problem P and the preceding reasoning context (e.g., $s_1, v_1, \ldots, s_{t-1}, v_{t-1}$). This is framed as a standard autoregressive, next-token prediction task, where we minimize the cross-entropy loss between the model's predicted token probabilities and the target tokens of the verification output v_t. For a

dataset of N such step-level training instances, the loss is:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{L^{(i)}} \log p_\theta\left(y_k^{(i)} \mid \text{input}^{(i)}, y_{<k}^{(i)}\right), \tag{4}$$

where θ represents the model parameters. For the i-th training instance, $\text{input}^{(i)}$ is the prompt containing P and the reasoning history up to $s_t^{(i)}$, and $y^{(i)}$ is the target verification sequence $v_t^{(i)}$. $L^{(i)}$ is the length of this target sequence $y^{(i)}$. By training on examples that require generating either short v_t^{S1} or longer v_t^{S2} sequences based on the input context, Dyve learns its adaptive verification strategy.

3.4 Inference with Dyve

During inference, the Dyve model sequentially verifies a sequence of reasoning steps $\{s_1, \ldots, s_T\}$ for a given problem P. For each step s_t, Dyve processes P and the cumulative steps $s_{1:t}$ to generate a verification response r_t:

$$r_t = \text{Dyve}(P, s_{1:t}; \theta). \tag{5}$$

This response r_t manifests as either a concise System 1 confirmation (e.g., +) or an elaborate System 2 analysis, its token length varying accordingly from one to 8192 tokens.

A parsing function, $\text{Parse}(r_t)$, then extracts a binary outcome (correct/error) from r_t; for System 1 outputs, this is direct, while for System 2's detailed explanations, the verdict is typically found at the end. If $\text{Parse}(r_t)$ indicates an error (e.g., returns 0), verification halts, returning the erroneous step's index t and the response r_t (which may contain the System 2 explanation). Otherwise, verification proceeds to the next step, s_{t+1}.

4 Experiments

To evaluate the capabilities of Dyve, particularly its core principle of *learned adaptive verification*, we conduct a series of experiments. These experiments are designed to: (1) assess Dyve's proficiency in identifying errors within multi-step reasoning processes, (2) quantify its dynamic allocation of System 1 (fast) and System 2 (deep) verification resources, (3) compare its performance and efficiency against established baselines, including other process reward models (PRMs) and LLM-as-Judge setups, and (4) evaluate its synergy when integrated with Proposer LLMs in a Best-of-N reasoning framework. All experiments were conducted on 8 × NVIDIA A800-SXM4-80GB GPUs. Interested readers may refer to Appendix A.1 for a detailed experimental setup.

4.1 Benchmarks and Evaluation Protocol

We utilize two primary benchmarks to evaluate different facets of Dyve's performance:

ProcessBench (Zheng et al., 2024) comprises four sets of test data derived from GSM8K (Cobbe et al., 2021a), MATH (Hendrycks et al., 2021), OlympiadBench (He et al., 2024), and Omni-MATH (Gao et al., 2024). It includes 3,400 test cases, covering high-school to Olympiad-level math problems. Each case provides a step-by-step solution with error locations annotated by experts. *Models are given $s_{1:t}$, from the first to the last step, and must identify the earliest error or confirm that all steps are correct.* For each ProcessBench subset, we calculate the accuracies for erroneous and correct samples and compute their harmonic mean as the F1 score.

MATH-500 (Lightman et al., 2023b) evaluates Dyve's integration with a Proposer LLM. We measure performance using maj@k and rm@k metrics as defined in (Yang et al., 2024) and apply a Best-of-N decoding strategy. Due to inconsistent results from different evaluation tools, we manually verified all reported outcomes.

Evaluation Protocol for Baselines. To ensure fair and rigorous comparisons, especially against LLM-as-Judge baselines on ProcessBench, we employed a standardized evaluation protocol. For each step s_t of a given reasoning trace, baseline LLMs were prompted with the problem context and the specific step using the format shown in Figure 3.

This step-by-step prompting ensures that baselines perform the same sequential verification task as Dyve. For models marked with an asterisk (*) in our results table (Table 1), this standardized prompting and evaluation was conducted using our custom implementation to maintain consistency.

4.2 ProcessBench Evaluation

We first evaluate Dyve's core capability in identifying process errors using the ProcessBench benchmark. The comprehensive results, including comparisons against various baselines, are presented in Table 1.

Overall Performance. As demonstrated in Table 1, Dyve 14B obtains substantial improvement

Table 1: Performance comparison on ProcessBench (F1 Scores). Dyve 14B leverages a dual reasoning approach (fast System 1 and deep System 2). Models marked with (*) are evaluated using our custom implementation aligning with the standardized protocol. The "Type" column clarifies the model's nature or the specific training data subset used.

Model	Type / Training Data	GSM8K	MATH	OlympiadBench	OmniMATH
Existing Process Reward Models (PRMs) - System 1 Verification					
Qwen2.5-Math-7B-PRM	System 1	39.4*	52.2*	39.4*	33.1*
Math-Shepherd-PRM-7B	System 1	47.9	29.5	24.8	23.8
RLHFlow-PRM-Mistral-8B	System 1	50.4	33.4	13.8	15.8
RLHFlow-PRM-Deepseek-8B	System 1	38.8	33.8	16.9	16.9
Skywork-PRM-1.5B	System 1	59.0	48.0	19.3	19.2
Skywork-PRM-7B	System 1	64.1*	43.2*	16.2*	17.9*
LLM-as-Judge Baselines (Standard Prompting)					
Llama-3.1-8B-Instruct	LLM-as-Judge	27.5*	26.7*	18.5*	19.2*
GPT-4o	LLM-as-Judge	61.9*	53.9*	48.3*	44.6*
QwQ-32B-Preview	LLM-as-Judge	62.3*	52.7*	46.2*	43.9*
Our Models and Ablations					
DeepSeek-R1-Distill-Qwen-14B	LLM-as-Judge (Base Model)	67.3*	38.8*	29.9*	32.1*
Qwen2.5-14B-Instruct	Trained on System 1+2 Data	51.7*	42.3*	30.2*	23.6*
DeepSeek-R1-Distill-Qwen-14B	Trained on System 2 Data Only	70.0*	60.1*	50.1*	49.6*
Dyve 14B (Ours)	**Trained on System 1+2 Data**	**68.5**	**58.3**	**49.0**	**47.2**

Standardized Prompt Format

```
Problem: [Problem Description]
Step 1: [s_1]
...
Step t: [s_t]

Is Step t correct given the problem and
previous steps?

You must answer with '+'
for correct or '-' for incorrect
at the end of your response.
```

Figure 3: Standardized prompt format used for evaluating baseline LLMs on each reasoning step s_t. LLMs were required to output '+' for a correct step or '-' for an incorrect step.

in F1 scores across all four ProcessBench subsets over other baselines, with 68.5 on GSM8K, 58.3 on MATH, 49.0 on OlympiadBench, and 47.2 on OmniMATH. This underscores the effectiveness of its learned adaptive verification strategy, which combines rapid System 1 checks with deep System 2 analysis. Dyve not only outperforms existing PRMs that typically rely on simpler System 1 verification but also surpasses strong LLM-as-Judge baselines, including powerful models like GPT-4o, especially on more complex benchmarks such as OmniBench and OlympiadBench.

Comparison with Specialized Baselines. To further understand the contributions of Dyve's training methodology and base model choice, we compare it against two specialized baselines derived from our rebuttal analysis. First, when the `DeepSeek-R1-Distill-Qwen-14B` model is trained *only on System 1 type data*, it achieves strong scores on GSM8K (66.3) and MATH (56.0). However, its performance drops notably on OlympiadBench (36.1) and OmniMATH (37.7) compared to the full Dyve model. This indicates that while System 1 training is effective for problems closer to its training distribution, the System 2 component, learned from data with detailed reasoning traces, is crucial for generalization to more complex, Olympiad-level mathematics that require deeper error analysis. Second, we trained `Qwen2.5-14B-Instruct` using our full *System 1+2 data*. This model achieved F1 scores of 51.7 (GSM8K), 42.3 (MATH), 30.2 (OlympiadBench), and 23.6 (OmniMATH). These results are significantly lower than Dyve. This suggests that `DeepSeek-R1-Distill-Qwen-14B` is a more suitable base model for learning Dyve's adaptive verification from our specialized dataset, potentially due to its inherent reasoning capabilities or better alignment with the training data format that includes metacognitive elements like `<think>` tags for System 2 responses. The standard `DeepSeek-R1-Distill-Qwen-14B` when used as an LLM-as-Judge also shows weaker performance (e.g., 38.8 on MATH, 29.9 on OlympiadBench) compared to Dyve, highlighting that Dyve's fine-

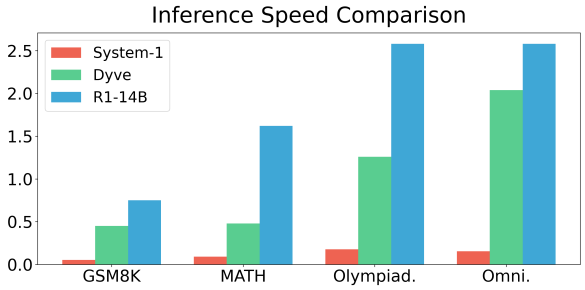

Figure 4: Inference speed comparison on ProcesBench, time per sample in seconds, for System-1, Dyve, and DeepSeek-R1-14B.

tuning on our step-wise consensus-filtered data is key to its superior verification abilities.

4.2.1 Analysis of Dyve's Adaptive Verification

A core tenet of Dyve is its learned ability to dynamically switch between fast, token-level System 1 verification for clear steps and more comprehensive System 2 analysis for ambiguous or erroneous ones. This adaptive behavior is not rule-based but emerges from training on curated data as in Figure 1. To quantify this, we analyzed the proportion of System 1 and System 2 responses from Dyve 14B on ProcessBench datasets, as shown in Table 2. Dyve exhibits a clear adaptive strategy.

On **GSM8K**, System 2 verification is predominant (System 1 usage: 27.5%). This is likely due to GSM8K's condensed reasoning traces where individual steps can encompass multiple calculations or implicit assumptions, benefiting from deeper scrutiny. Strategic System 1 use for clearer substeps still contributes to efficiency (discussed in Section 4.2.2).

For more complex datasets like **MATH, OlympiadBench, and OmniMATH**, which involve longer reasoning chains, Dyve increases its reliance on System 1 verification (30.7%, 41.0%, and 32.6% respectively). This allows efficient processing of straightforward segments, reserving intensive System 2 analysis for genuinely complex or erroneous steps.

This dynamic allocation is crucial for maintaining high performance on challenging problems while managing computational resources. Figure 1 illustrates Dyve's learned capability, showing a concise System 1 confirmation for a correct arithmetic step versus a detailed System 2 thought process and refutation for an incorrect step requiring multi-step derivation.

4.2.2 Comparison on Inference Time

The adaptive nature of Dyve is designed not only for accuracy but also for efficiency. Figure 4 compares the average inference time per sample on ProcessBench for Dyve 14B, a hypothetical System 1-only version, and the base DeepSeek-R1-14B model when performing detailed System 2-like analysis for every step.

As illustrated in Figure 4, Dyve 14B is notably faster than its base model when the latter is forced to generate detailed step-by-step explanations for every step. While a pure System 1 verifier is inherently the fastest, Dyve strikes a balance. Its efficiency gains are directly attributable to its dynamic allocation of System 1 verification for a substantial portion of steps as quantified in Table 2. For instance, on GSM8K, even with 72.5% System 2 usage, the average time per sample for Dyve (0.47s, from Table 2) is considerably lower than if all steps required full System 2 analysis. This demonstrates that Dyve effectively conserves computational resources without unduly sacrificing verification depth, making it well-suited for applications requiring both accuracy and reasonable processing speed.

4.2.3 Impact of Model Choice and Step-wise Consensus Filtering

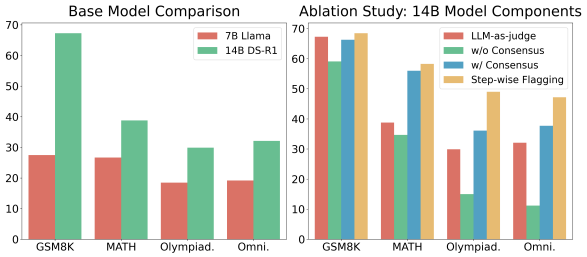

Figure 5: Impact of model choice and step-wise consensus filtering on performance across GSM8K, MATH, OlympiadBench, and OmniMATH. The figure illustrates improvements achieved through consensus filtering and step-wise flagging, highlighting the superior performance of the 14B reasoning model over the 7B Llama.

Our step-wise consensus-filtered process supervision pipeline is crucial for Dyve's adaptive verification strategy. Figure 5 illustrates the impact of our model and data curation choices.

The left panel of Figure 5 compares `Llama-3.1-8B-Instruct` and `DeepSeek-R1-Distill-Qwen-14B` as Dyve's base models, both trained on our System 1+2 dataset. The 14B DeepSeek-R1 consistently outperformed the 8B Llama, validating our choice of a more capable architecture for this nuanced

Table 2: Dyve 14B: Distribution of System 1 vs. System 2 responses and average verification time per sample on ProcessBench datasets.

Dataset	System 1 Resp.	System 2 Resp.	System 1 %	Avg. Time (s)
GSM8K	420	1106	27.5%	0.47
MATH	1263	2849	30.7%	0.51
OlympiadBench	2102	3027	41.0%	1.25
OmniMATH	1336	2766	32.6%	2.05

verification task.

The right panel of Figure 5 highlights the progressive benefits of our data curation for the `DeepSeek-R1-Distill-Qwen-14B` model. While training on unfiltered Monte Carlo data ("w/o Consensus") offered some improvement over raw LLM-as-Judge, Consensus Filtering significantly boosted performance (e.g., MATH F1 from 34.7 to 56.0). Crucially, the final Stepwise Flagging stage, where a reasoning LLM (`DeepSeek-R1-Distill-Qwen-32B`) created explicit System 1 ('+') and System 2 (error explanations) target outputs, led to full Dyve 14B performance (e.g., MATH F1 58.3, GSM8K 68.5). This final step is vital for enabling Dyve's adaptive System 1 / System 2 verification mechanism, underscoring the importance of our data pipeline in achieving Dyve's high performance.

4.3 Integrating Dyve with Proposer LLMs

We integrate Dyve as a process verifier to assist Proposer LLMs (Qwen-Math-7B and Deepseek-R1-Distill-Qwen-14B) on MATH-500. For fairness, we compare three setups across Best-of-N (N = 1, 2, 4, 8) decoding settings: Dyve verification, System 1 only, and Majority Vote (no verification).

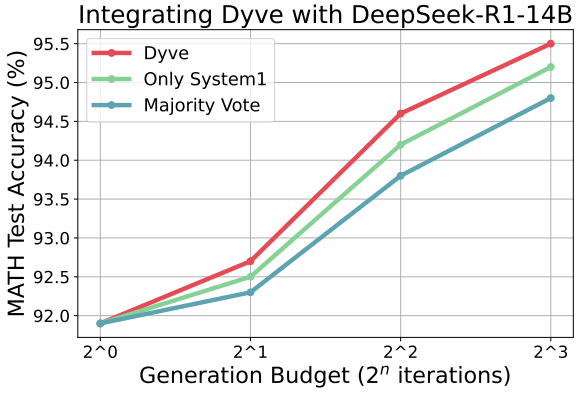

Figure 6: Comparison of Dyve, Dyve System1 and Majority Vote with different generation budget when integrating with the reasoning model DeepSeek-R1-Distill-Qwen-14B as Proposer LLMs.

Results and Analysis As shown in Figures 6 and 7, our proposed Dyve framework consistently demonstrates superior performance across different Proposer LLMs, irrespective of their inherent reasoning capabilities. Specifically, Dyve's integration of fast and slow verification mechanisms outperforms both the Majority Voting and the System 1 only verification baselines when coupled with Best-of-N decoding.

When integrated with the reasoning model, DeepSeek-R1-14B (Figure 6), Dyve achieves a peak accuracy of 95.5% with a generation budget of N = 8. This significantly surpasses the performance of both Majority Vote and relying solely on System 1 verification with the same underlying Proposer LLM. Similarly, when Dyve is integrated with Qwen2.5-MATH-7B-Instruct, a model with comparatively less emphasis on complex reasoning (Figure 7), it still reaches an impressive accuracy of 90.4% at N = 8. Notably, Dyve's performance with Qwen2.5-MATH-7B-Instruct also exceeds that of the Majority Vote and System 1 only approaches using the same Proposer LLM.

These results demonstrate Dyve's robustness and generalizability. By combining fast and slow verification, Dyve effectively guides both strong reasoning models and more general language models towards accurate solutions, highlighting its broad applicability.

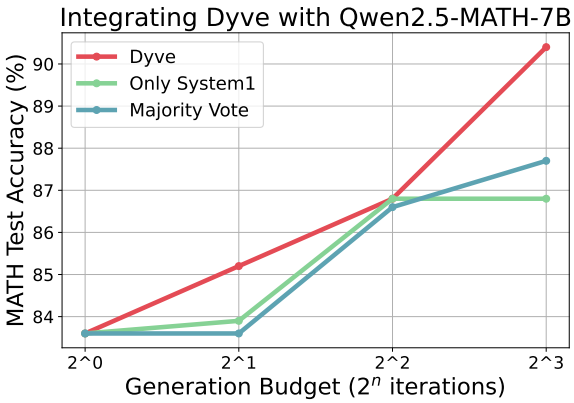

Figure 7: Comparison of Dyve, Dyve System1 and Majority Vote with different generation budget when integrating with the language model Qwen2.5-MATH-7B-Instruct as Proposer LLMs.

Table 3: Performance (F1 Score) of various base models before (Base) and after fine-tuning. The results show consistent improvement across all architectures. 'OlympiadB.' is OlympiadBench.

Model	SFT Data	GSM8K	MATH	OlympiadB.	OmniMATH
Qwen2.5 14B	Base	28.7	27.9	23.7	24.6
Qwen2.5 14B	System 1 + System 2	51.7	42.3	30.2	23.6
Qwen3-8B	Base	28.3	24.0	13.9	17.3
Qwen3-8B	System 1 + System 2	35.5	33.7	21.9	18.9
DeepSeek-R1 14B	Base	67.3	38.8	29.9	32.1
DeepSeek-R1 14B	System 1 + System 2 (Dyve)	**68.5**	**58.3**	**49.0**	**47.2**

4.3.1 Ablation on Base Models and Training Data

To validate that Dyve's performance gains are attributable to our adaptive verification framework and specialized data, rather than solely the strength of the base model, we conducted two key ablation studies.

Effectiveness of Our Curated Data The results, presented in Table 3, shows that our System 1 + System 2 fine-tuning methodology provides substantial and consistent performance gains across all models. For instance, Qwen2.5 14B's F1 score on GSM8K nearly doubles from 28.7 to 51.7, and DeepSeek-R1 14B's MATH score jumps from 38.8 to 58.3. This demonstrates that the performance gain is attributable to our adaptive verification framework and specialized training data, not just the base model's inherent strength.

Justification for DeepSeek-R1 14B as Base Model While our method improves all models, the results also justify our choice of DeepSeek-R1 14B as the base for Dyve. After fine-tuning, it achieves the highest performance across all benchmarks, particularly on the more challenging ones. Its F1 score of 58.3 on MATH and 49.0 on OlympiadBench significantly surpasses the other fine-tuned models. This suggests that while our data provides a strong learning signal for any model, a more capable reasoning model like DeepSeek-R1 14B is better able to leverage this signal for complex verification tasks.

Dyve vs. System-2-Only Verification. To quantify the trade-off between Dyve's adaptive strategy and a uniformly deep verification approach, we trained a model exclusively on our "System 2 Only" data. As shown in Table 4, this model can achieve marginally higher F1 scores.

Table 4: Accuracy trade-off between our adaptive Dyve model and a model trained exclusively on System 2 data. Dyve achieves comparable performance with significantly higher efficiency.

Dataset	**System 1 + System 2 (Dyve)**	**System 2 Only**
GSM8K	68.5	70.0
MATH	58.3	60.1
OlympiadBench	49.0	50.1
OmniMATH	47.2	49.6

However, this minor accuracy gain comes at a significant efficiency cost. As illustrated previously in Figure 4, the "System 2 Only" approach is substantially slower than our model. In contrast, Dyve's strength lies in its ability to strategically allocate its verification resources. It applies fast, token-level System 1 checks for the of straightforward steps and reserves the deep analysis of System 2 for only the most complex or potentially erroneous ones. This allows Dyve to achieve highly competitive accuracy with far greater efficiency.

5 Conclusion

Our study demonstrates Dyve's, with a dual reasoning approach, superior performance in mathematical reasoning verification. The consensus filtering and step-wise flagging significantly enhanced model accuracy and robustness. Ablation studies confirm the 14B model's advantages over smaller variants for complex reasoning tasks, establishing Dyve as an effective solution for precise and efficient error detection.

6 Acknowledgment

This work was supported in part by the CUHK Strategic Seed Funding for Collaborative Research Scheme under Grant No. 3136023., and in part by the CUHK Research Matching Scheme under Grant No. 7106937, 8601130, and 8601440.

7 Broader Ethical Impact

Our method is centered on rigorous verification of AI reasoning, ensuring each step is systematically validated for enhanced reliability and transparency. By exclusively using publicly available datasets under their proper licenses, we adhere to responsible research practices. We believe that improving verification in AI reasoning not only boosts system robustness but also exemplifies ethical AI development.

8 Limitations

While Dyve demonstrates strong performance, it shares several limitations common to verification-based systems. Its effectiveness naturally depends on the complexity of the reasoning tasks, and more intricate multi-step problems may require further adaptation or deeper analysis. In addition, although our consensus-filtered process supervision considerably enhances signal quality, a modest level of noise remains inherent in any automated estimation process. Finally, the overall performance is influenced by the quality and diversity of the training data, suggesting that further efforts in data curation and filtering could yield even more robust results. These aspects offer promising directions for future research.

References

Aitor Arrieta, Miriam Ugarte, Pablo Valle, José Antonio Parejo, and Sergio Segura. 2025. Early external safety testing of openai's o3-mini: Insights from the pre-deployment evaluation. Preprint, arXiv:2501.17749.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021a. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168.

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

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. ArXiv, abs/2410.07985.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Yuanzhuo Wang, and Jian Guo. 2024. A survey on llm-as-a-judge. ArXiv, abs/2411.15594.

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

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. Preprint, arXiv:2402.14008.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Xiaodong Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. ArXiv, abs/2103.03874.

Daniel Kahneman. 2012. Thinking, fast and slow. Penguin, London.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023b. Let's verify step by step. ArXiv, abs/2305.20050.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, et al. 2024. Improve mathematical reasoning in language models by automated process supervision. arXiv preprint arXiv:2406.06592.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. arXiv preprint arXiv:2501.12599.

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

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

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. arXiv preprint arXiv:2409.12122.

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

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

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. arXiv preprint arXiv:2501.07301.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2024. Processbench: Identifying process errors in mathematical reasoning. arXiv preprint arXiv:2412.06559.

A Appendix

A.1 Training Details

Our model processes inputs with a maximum token length of 2048, ensuring robust contextual understanding. To further enhance efficiency, we employ Low-Rank Adaptation (LoRA) configured with a rank of 16, an alpha value of 16, and a dropout rate of 0.1. The training regimen spans three epochs, using a per-device batch size of 2 and leveraging gradient accumulation over 8 steps. The learning rate is set to 2×10^{-5} and a weight decay of 0.01 is applied. Training is executed with mixed precision (fp16), optimizing computational resources without sacrificing performance.

Inference During inference, our model leverages a multi-step reasoning process to evaluate each problem instance. The procedure begins by formulating a sequence of conversational prompts that encapsulate both the problem statement and its progressive steps. At each step, the Dyve model is queried via its custom chat interface, and the generated response is examined for specific response patterns — such as the presence of a "+" symbol signaling a correct evaluation. This iterative mechanism continues until a response fails to meet the designated correctness criteria, at which point the process halts. To ensure efficiency, the inference is executed concurrently using a pool of 32 parallel workers, processing various configurations from the ProcessBench dataset (including `gsm8k`, `math`, `olympiadbench`, and `omnimath`). For every evaluated problem, all intermediate responses (or generations) and the final step classification are recorded. These results are then systematically saved in JSON Lines format, facilitating subsequent analysis and serving as a robust foundation for further evaluation.

A.2 Benchmark Details

ProcessBench (Zheng et al., 2024) serves as our main benchmark for evaluating step-wise error identification. It comprises four challenging test sets derived from GSM8K (Cobbe et al., 2021a), MATH (Hendrycks et al., 2021), OlympiadBench (He et al., 2024), and OmniMATH (Gao et al., 2024). ProcessBench includes 3,400 test cases that span from high-school to Olympiad-level mathematical problems. Each case provides a problem, a step-by-step solution, and expert annotations indicating the location of the first error, if any. For this benchmark, models are tasked with sequentially verifying a given reasoning trace $\{s_1, s_2, \ldots, s_T\}$. For each step s_t, the model must determine if an error has occurred in $s_{1:t}$. If an error is identified, the process halts, and the index of the erroneous step is reported; otherwise, the model confirms the correctness of all steps. Our primary metric for ProcessBench is the F1 score, calculated as the harmonic mean of the accuracies on identifying erroneous samples and correct samples (Zheng et al., 2024). This metric effectively balances a model's ability to detect errors without being overly critical.

MATH-500 (Lightman et al., 2023b) is employed to assess Dyve's effectiveness when integrated as a verifier with Proposer LLMs. This dataset consists of 500 problems from the MATH dataset. We measure performance using the pass@k (maj@k and rm@k) metrics as defined in (Yang et al., 2024), applying a Best-of-N (BoN) decoding strategy where N varies (1, 2, 4, 8). Due to known inconsistencies with automated evaluation tools for complex mathematical reasoning, all reported outcomes on MATH-500 were manually verified for accuracy.

A.3 Efficient Estimation of MCTS

In this section, we detail our approach to efficiently utilize Monte Carlo Tree Search (MCTS) for sampling rollouts, which are crucial for training process-based verifiers.

Overview

Our method leverages MCTS to construct a state-action tree representing detailed reasoning paths for a given question. This approach allows us to collect Process-based Reward Model (PRM) training examples by exploring various reasoning paths and identifying errors efficiently.

State-Action Tree Construction

Each state s in the tree corresponds to a question and its preceding reasoning steps, with the root state being the question without any reasoning steps. An action a is a potential next step, and the state transition function is defined as $s' = \text{Concatenate}(s, a)$. Each node s stores the visit count $N(s)$, Monte Carlo estimation $MC(s)$, and rollout value function $Q(s, r)$.

Table 5: Cost Comparison of the Automated Pipeline vs. Manual Annotation.

Cost Component	Our Automated Pipeline	Manual Annotation Path
Rollout Generation	1,344 A800-hrs	1,344 A800-hrs
Label Generation Method	DS-32B Model Inferencing	Human Expert Labeling
API / Labeling Cost	$2,500 (DeepSeek V3)	$585,000 (5 dollar each)

MCTS Process

Selection We maintain a pool of rollouts with $0 < MC(s) < 1$. During selection, a rollout is chosen based on tree statistics using a variant of the PUCT algorithm:

$$U(s) = c_{\text{puct}} \frac{\sqrt{\sum_i N(s_i)}}{1 + N(s)}$$

This strategy initially favors rollouts with low visit counts, gradually shifting preference towards those with high rollout values.

Binary Search A binary search identifies the first error location in the selected rollout. Rollouts with $0 < MC(s) < 1$ are added to the selection pool. The search process divides and examines rollouts to pinpoint errors, updating the tree with new states.

Maintenance After binary search, update the statistics $N(s)$, $MC(s)$, and $Q(s, r)$. Increment $N(s)$ for the selected state-action pair and update $MC(s)$ and $Q(s, r)$ based on the binary search results.

Tree Construction Repeat the above process to construct the state-action tree. The process ends when the search count reaches a predetermined limit or no additional rollouts are available.

A.4 Training Data Example

A.5 On the Computational Overhead of the Supervision Pipeline

The computational cost is a one-time, offline investment to create a reusable dataset. Crucially, the expensive "Rollout Generation" step is a prerequisite for *both* automated and manual annotation. Our method provides a >99% cost reduction by replacing the slow, expensive manual labeling with an efficient inference step.

Table 5 clarifies this comparison.

A.6 Examples of "non-trivially correct" steps triggering System 2

A System 2 response can be triggered for steps that are correct but logically complex, such as verifying the correctness of a multi-step plan. In these cases, a simple "correct" token is insufficient; the model must reason through the logic of the plan itself:

As shown in 9, Dyve uses its System 2 thinking process to analyze the logical flow of the proposed plan before concluding that it is correct. This is distinct from a simple arithmetic check and highlights the deeper verification our model learns to perform.

A.7 How is the S1/S2 determination made during data processing?

Thank you for the opportunity to clarify. The decision is not made by a hard-coded rule, but is an emergent property of our data pipeline that Dyve *learns*.

In our pipeline (Section 3.2, "System 1 and System 2 Target Output Generation"), we use a powerful teacher model (**DeepSeek-R1-Distill-Qwen-32B**) to analyze each step of a filtered reasoning trace.

- If this teacher model deems a step to be straightforward and correct, we generate a simple + token as the target output (**System 1 signal**).
- If the teacher model identifies an error or a complex step that requires explanation, it generates a detailed thought process within `<think>` tags followed by a verdict (**System 2 signal**).

Dyve is then trained via SFT on this data. It learns to associate different kinds of reasoning steps with either a short, confirmatory output or a longer, analytical one. The choice at inference time is therefore a learned behavior of the model, not a predefined rule.

Example Data Sample

LLM Proposer:

```
Step: Cowboy Mickey and cowgirl Minnie train horses. Mickey mounts six less than twice as
many horses per day as Minnie, while Minnie mounts three more horses per day than there are days in
a week. How many horses does Mickey mount per week?
Step 1: Minnie mounts 7+3=<<7+3=10>>10 horses per day.
Is this step correct? Answer with '+' for correct or '-' for incorrect.
```

Dyver:

```
<think>
</think> +
```

LLM Proposer:

```
Step 2: Mickey mounts 12-6=<<12-6=6>>6 horses per day.
Is this step correct? Answer with '+' for correct or '-' for incorrect.
```

Dyver:

```
<think>
Let's verify Step 2 ....
Mickey mounts six less than twice Minnie's count. Minnie mounts 10 horses, so Mickey should mount
2*10 - 6 = 14. Step 2 is incorrect.
</think>
Answer: -
```

Figure 8: A example of the training data.

Standardized Prompt Format

```
step: To determine how much James made from selling the cars, we need to f
ollow these steps: First, calculate the total cost of the cars James sold.
Second, determine the total amount the company paid for the cars. Third, f
ind the profit from selling the cars. Fourth, calculate the commission Jam
es received based on his profit.

response: <think>Okay, so I need to figure out if the steps provided are c
orrect for determining how much James made from selling the cars. Let me b
reak it down step by step.

The steps given are:
1. Calculate the total cost of the cars James sold.
2. Determine the total amount the company paid for the cars.
3. Find the profit from selling the cars.
4. Calculate the commission James received based on his profit.

Let me analyze each step to see if they make sense.

[omitted for brevity.....]
</think>

The steps provided are correct as they accurately follow the necessary cal
culations to determine James's earnings.
+
```

Figure 9: An example of "non-trivially correct" steps triggering System 2.