# Learning When to Revise: Adaptive Revision Depth for Efficient Test-Time Reasoning

**Anonymous ACL submission**


## Abstract

Test time reasoning substantially improves large language model performance by using additional inference time computation. However, applying more reasoning uniformly to every input can add unnecessary cost and cause regressions, where iterative revision changes an initially correct response into an incorrect one. We formalize this problem as adaptive revision depth selection after CoT and propose Hegelian inspired Adaptive Dialectical Depth (`HegelADD`), a lightweight controller that chooses an appropriate refinement depth for each input. In a process with a frozen LLM verifier and reviser, `HegelADD` sequentially decides whether to continue refinement and controls revision depth with only 4,546 trainable parameters. Experiments show that `HegelADD` achieves a better accuracy and cost tradeoff than CoT and fixed depth refinement baselines. These results indicate that efficient LLM reasoning depends not on longer reasoning for all inputs, but on deciding when to revise or stop for each input.

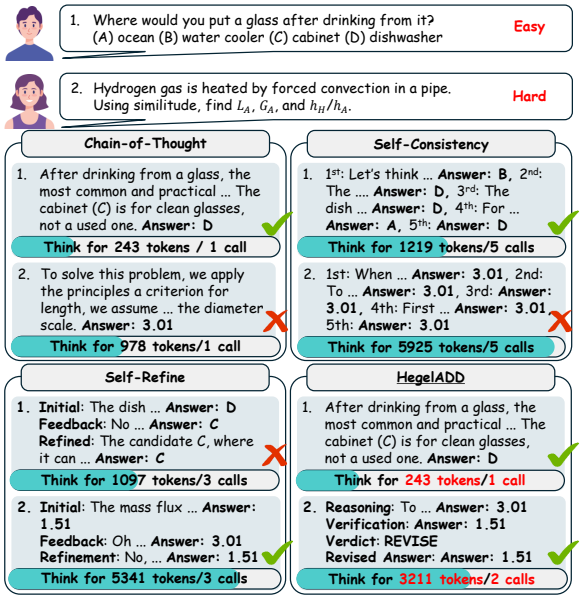




Figure 1: Post-CoT adaptive revision-depth selection. Rather than applying uniform refinement after the initial CoT answer, the method decides per input whether to continue and allocates only needed reasoning depth.

## 1 Introduction

LLMs perform strongly on many reasoning tasks, yet complex problems still expose unstable reasoning and errors. Test time methods including chain of thought (CoT) (Wei et al., 2022), self consistency (Wang et al., 2023), and self refinement (Madaan et al., 2023) address these limits with inference time computation. Recent work extends this direction with repeated sampling (Brown et al., 2024), verifier guided search (Snell et al., 2025), and adaptive budget allocation (Zhao et al., 2025). However, applying more reasoning uniformly to every input is not always effective. On easy problems, extra refinement only adds cost, and iterative self correction can cause regression, where a later revision changes an already correct response into an incorrect one (Huang et al., 2024). This suggests that the core question in test time reasoning is not simply how to increase reasoning depth, but **when to refine and when to stop for each input**.

This paper focuses on revision depth after CoT within test time compute. Figure 1 summarizes post CoT adaptive revision depth selection. The goal is to stop early on easy problems and revise difficult ones more deeply, rather than spend the same budget on all inputs. A frozen LLM first generates an initial CoT response, and the same LLM with verifier and reviser prompts can produce one pass and two pass revision candidates. Existing methods often use these depths as a fixed policy for all inputs, but appropriate depth varies with input difficulty and revision stability. Refinement is not monotonic, and an additional revision can damage a correct answer.

We propose **Hegelian inspired Adaptive Dialectical Depth** (HegelADD), a lightweight online controller inspired by thesis, antithesis, and synthesis. It decides whether to continue post CoT refinement using only observed information. The method does not fine tune the base LLM or train a new generator. Rather than selecting among generated candidates, it controls revision depth online on a frozen LLM. Experiments on in and out of domain reasoning benchmarks show that HegelADD achieves a better accuracy and cost tradeoff than fixed depth refinement while reducing unnecessary calls across several backbone LLMs. These results suggest that efficient test time reasoning depends on selectively allocating needed revision depth to each input rather than applying longer reasoning uniformly.

Our contributions are as follows.

- We formulate iterative verifier and reviser refinement after CoT as adaptive revision depth selection for each input, and analyze revision non monotonicity, where additional revision does not always improve accuracy.
- We propose HegelADD, a lightweight online controller that sequentially decides whether to continue refinement using only decision time visible information. It does not fine tune the base LLM and controls revision depth with only 4,546 trainable parameters.
- We empirically show that HegelADD provides a better accuracy and cost tradeoff than fixed depth refinement across reasoning regimes and backbone LLMs. Diagnostic analyses further characterize the need for adaptive stopping and the failure boundary of iterative refinement.

## 2 Related Work

**CoT and repeated inference.** CoT prompting and its variants improve reasoning by adding inference time computation through intermediate reasoning, repeated sampling, and majority voting (Wei et al., 2022; Wang et al., 2023). Test time scaling also shows that more inference compute can improve reasoning coverage and accuracy (Brown et al., 2024; Snell et al., 2025). In contrast, HegelADD makes only the revision calls needed after the initial CoT instead of globally increasing candidates.

**Self-refinement and verifier-guided reasoning.** Self refinement methods improve reasoning by having a model repeatedly critique and revise its outputs (Madaan et al., 2023; Shinn et al., 2023). However, prior work reports that intrinsic self correction can cause regression by changing an already correct response into an incorrect one (Huang et al., 2024). Verifier guided reasoning and process supervision highlight intermediate reasoning evaluation (Lightman et al., 2024; Uesato et al., 2022; Wang et al., 2024a; Chang et al., 2025), but often require an additional verifier or search procedure. HegelADD instead performs lightweight online stopping using only the generated revision state so far.

**Adaptive budget control.** Recent adaptive reasoning studies directly address input wise compute allocation. Yang et al. (2025b) use transition points and confidence in the reasoning trace for early exit, and Fang et al. (2025) learn control tokens for choosing short or long reasoning with reinforcement learning. Lou et al. (2025) also treats CoT triggering as a Pareto style accuracy and cost tradeoff and trains an RL policy. More broadly, confidence and self evaluation signals have been studied for deciding when model outputs are reliable (Kadavath et al., 2022; Manakul et al., 2023). Unlike these methods, HegelADD targets adaptive revision depth selection for iterative refinement after the initial CoT. It uses only two lightweight MLP gates to decide whether to continue refinement, without base LLM fine tuning, a large routing model, or an external verifier.

## 3 Method

We define the initial CoT response as $y_0$, the one step revision candidate as D1 ($y_1$), and the two step revision candidate as D2 ($y_2$). Figure 2 shows the overall structure of HegelADD. HegelADD is a lightweight online controller that decides whether to perform the next refinement on a revision trajectory generated by a frozen LLM, using only the state observable so far. Unlike a post hoc selector that first generates all candidates and then chooses one, it can skip unnecessary revision calls during inference.

### 3.1 Candidate States

Let $x$ be the input problem and $M$ be a frozen backbone LLM. We first generate an initial response with the CoT prompt $p_{\text{cot}}$:

$$y_0 = M(p_{\text{cot}}(x)). \tag{1}$$

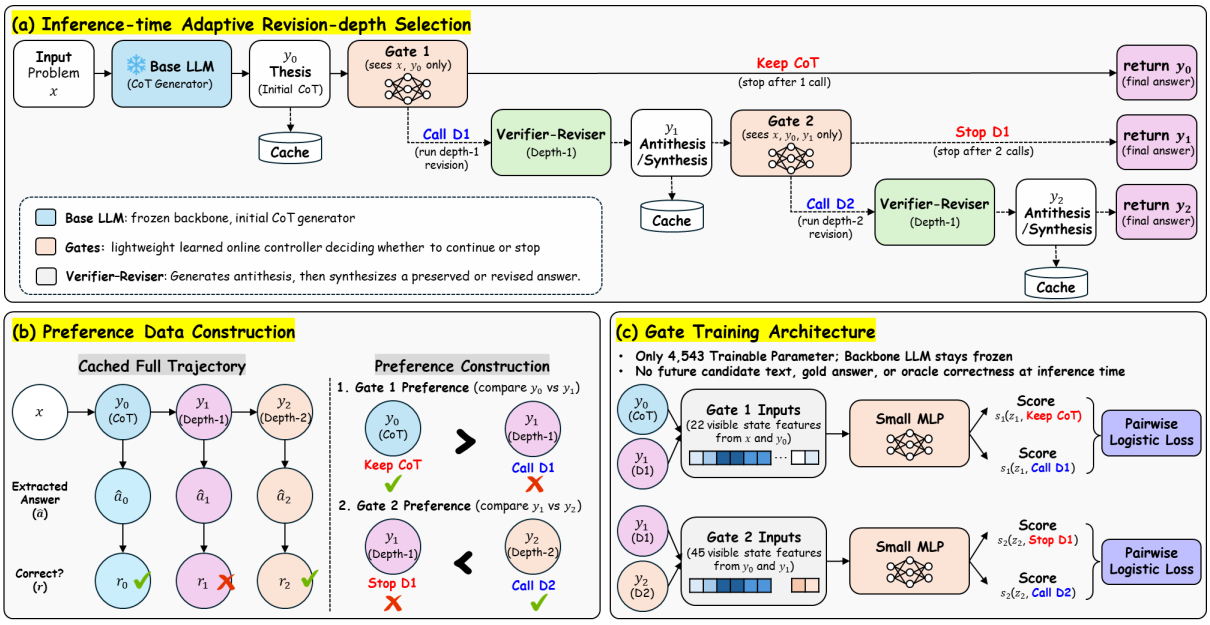



Figure 2: HegelADD framework. HegelADD consists of inference time online depth selection, offline preference data construction, and gate learning. At inference time, a frozen LLM generates CoT and revision candidates, and two gates sequentially decide whether to continue refinement using only the currently visible state. In the offline stage, gate training uses pairwise action preferences constructed from cached trajectories.

**Algorithm 1** HegelADD online inference

**Require:** input problem $x$, frozen LLM $M$, gates $G_1, G_2$
$y_0 \leftarrow M(p_{\text{cot}}(x))$
$a_1 \leftarrow G_1(x, y_0)$
**if** $a_1 =$ KEEP_COT **then**
  **return** $y_0$
**end if**
$y_1 \leftarrow M(p_{\text{rev}}(x, y_0))$
$a_2 \leftarrow G_2(x, y_0, y_1)$
**if** $a_2 =$ STOP_D1 **then**
  **return** $y_1$
**end if**
$y_2 \leftarrow M(p_{\text{rev}}(x, y_1))$
**return** $y_2$

We then repeatedly apply the verifier and reviser prompt $p_{\text{rev}}$ to generate revision candidates:

$$y_t = M(p_{\text{rev}}(x, y_{t-1})), \quad t \in \{1, 2\}. \tag{2}$$

While fixed depth baselines apply the same revision depth to all inputs, HegelADD selects an appropriate stopping depth for each input.

### 3.2 Online Revision-Depth Selection

Algorithm 1 summarizes the inference time online depth selection policy of HegelADD. HegelADD uses two binary gates sequentially. Gate1 decides whether to keep the initial CoT or perform one step revision based on $(x, y_0)$. Gate2 then decides whether to refine further based on $(x, y_0, y_1)$. Let $d_1, d_2 \in \{0, 1\}$ denote the gate decisions. The model call cost for an input is $C(x) = 1 + d_1 + d_1 d_2$. Thus, HegelADD uses between one and three LLM calls.

### 3.3 Offline Preference Construction

To train the gates, we generate the full trajectory $(y_0, y_1, y_2)$ offline on the training split. We extract the final answer from each candidate and compare it with the gold answer $g$ to compute correctness:

$$\begin{aligned} \hat{a}_t &= \text{Ans}(y_t), \\ r_t &= \mathbf{1}[\text{Eval}(\hat{a}_t, g)], \quad t \in \{0, 1, 2\}. \end{aligned} \tag{3}$$

where $\text{Ans}(\cdot)$ extracts the final answer from a candidate response, and $r_t$ denotes binary correctness for the candidate. The supervision for each gate is constructed to reflect whether additional revision is beneficial at the current state. For example, when correctness improves after revision, the continuation action is preferred. When regression occurs, the stopping action is preferred.

### 3.4 Gate Learning and Visibility Constraint

Let $\mathcal{D}_k = \{(z_i, a_i^+, a_i^-)\}_{i=1}^{N_k}$ be the preference dataset for Gate $k$. Here, $z_i$ is the state visible at the corresponding decision point, and $a_i^+$ and $a_i^-$ are the preferred and dispreferred actions. The gate defines the score margin between the two actions as

$$m_{k,i} = s_k(z_i, a_i^+) - s_k(z_i, a_i^-). \tag{4}$$

Each gate minimizes the pairwise logistic loss:

$$\mathcal{L}_k = \sum_{i=1}^{N_k} -\log \sigma(m_{k,i}). \tag{5}$$

The two gates are implemented as small MLP (Multi Layer Perceptron) controllers, while the backbone LLM remains frozen. Gates use only the state visible at decision time and cannot access future revision candidates or oracle correctness.

## 4 Experimental Setup

**Datasets.** We evaluate on a heterogeneous reasoning suite of 5,548 examples covering commonsense reasoning, mathematical reasoning, science question answering, and knowledge intensive multiple choice tasks (Talmor et al., 2019; Geva et al., 2021; Cobbe et al., 2021; Clark et al., 2018; Welbl et al., 2017; Clark et al., 2019; Khot et al., 2020; Mihaylov et al., 2018; Zellers et al., 2019; Hendrycks et al., 2021a; Wang et al., 2024b; Hendrycks et al., 2021b; Rein et al., 2024); see Appendix A. Reporting groups are constructed from dataset source and difficulty, measured by `Qwen3-8B` zero shot accuracy, and are used only as reporting diagnostics (Table 1). These groups are not used as controller input features, threshold selection inputs, or training labels. They are only aggregation units for interpreting results after final answer evaluation.

**Backbone Models.** The main experiment uses `Qwen3-8B` (Yang et al., 2025a) as the frozen backbone LLM, and the backbone is not fine tuned in any setting. We also evaluate backbone transfer by replaying the controller trained on `Qwen3-8B` over `Llama-3.1-8B-Instruct` and `Llama-3.2-3B-Instruct` (Dubey et al., 2024) trajectories. Finally, we add trajectories from the 24B scale backbone `Mistral-Small-3.2-24B-Instruct-2506` (Mistral AI, 2025) to test whether adaptive revision depth selection is maintained for a stronger instruction tuned backbone. For `Mistral-Small-3.2-24B-Instruct`, we generate $(y_0, y_1, y_2)$ trajectories with the same CoT and verifier and reviser prompt family and evaluate them under the same budget matched reporting protocol as the main table. Test labels are not used for controller training or decision threshold selection.

**Baselines.** We compare against both single pass and repeated inference baselines. Zero shot is a standard prompting baseline that uses only the problem and an answer format instruction (Brown et al., 2020). CoT generates one reasoning trace (Wei et al., 2022). SC@5 applies majority voting over five CoT samples (Wang et al., 2023). Self Refine is implemented as a fixed three call self feedback refinement pipeline (Madaan et al., 2023). D1 and D2 are fixed depth baselines that apply the same verifier and reviser prompt to every input for one pass and two pass revision, respectively. `HegelADD` uses the same candidate family but decides online whether to continue refinement for each input. Exact templates are in Appendix E.

Table 1: Reporting group definition. ID and OOD denote in-domain and out-of-domain evaluation groups, respectively. Difficulty levels are based on `Qwen3-8B` zero-shot accuracy on the final evaluation suite: Easy ($\geq 0.80$), Medium ($0.60 \leq \text{acc} < 0.80$), and Hard ($< 0.60$).

| Reporting group | Level | Datasets |
|---|---|---|
| ID Primary | Easy | `CSQA` |
| ID Primary | Medium | `SQA`, `GSM8K` |
| ID Aux. | Easy | `ARC-Easy`, `SciQ`, `ARC-Challenge`, `BoolQ`, `QASC`, `OpenBookQA` |
| ID Aux. | Medium | `HellaSwag` |
| OOD Medium | Medium | `MMLU` |
| OOD Hard | Hard | `MMLU-Pro`, `MATH-500`, `GPQA-Diamond` |

**Controller Training.** For controller training, we generate cached revision trajectories $(y_0, y_1, y_2)$ on the training split and construct pairwise preference examples for each gate from candidate correctness. Gate1 learns whether to keep the initial CoT response or call depth 1 revision, and Gate2 learns whether to refine further after depth 1 revision. The two gates are lightweight MLPs with hidden size 64. Gate1 uses 22 features and Gate2 uses 45 features. The total number of trainable parameters is 4,546, and the backbone LLM remains frozen in all experiments. Each gate uses only the state visible at decision time, and decision thresholds are selected on the dev split. In cached final suite replay, controller overhead is 0.839 ms per example, which is negligible relative to an additional LLM generation call. Thus, the efficiency gains of `HegelADD` come from reducing the average number of LLM calls, and the controller itself is not an inference time bottleneck.

**Evaluation.** We use three metrics. Accuracy is computed by comparing the extracted final answer against the gold answer. Average total tokens and model calls measure generated tokens and the number of LLM calls, respectively. Single trajectory

Table 2: Accuracy by reporting group and overall average cost. The *Overall* block reports average accuracy, total tokens, and model calls. In each accuracy column, the best result is shown in **bold** and the second best result is underlined.

| Backbone | Method | ID Primary | | | ID Aux. | | OOD Medium | OOD Hard | | | Overall | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CSQA | SQA | GSM8K | Aux.E | Aux.M | MMLU | MMLU-Pro | MATH-500 | GPQA-Diamond | Acc | Avg. Tokens | Calls |
| Llama-3.2-3B-Instruct | Zero-shot | **0.682** | **0.642** | 0.691 | **0.775** | **0.555** | **0.556** | 0.198 | 0.064 | 0.242 | **0.531** | 190.5 | 1.000 |
| | CoT | 0.533 | 0.553 | 0.476 | 0.574 | 0.420 | 0.422 | 0.159 | 0.094 | 0.237 | 0.410 | 326.9 | 1.000 |
| | SC@5 | 0.476 | 0.549 | **0.753** | 0.542 | 0.425 | 0.400 | 0.156 | 0.126 | 0.242 | 0.423 | 2322.0 | 5.000 |
| | Self-Refine | 0.464 | 0.529 | 0.689 | 0.524 | 0.390 | 0.401 | 0.160 | **0.252** | 0.253 | 0.421 | 2224.7 | 3.000 |
| | D1 | 0.415 | 0.482 | 0.651 | 0.469 | 0.365 | 0.350 | 0.149 | 0.076 | 0.258 | 0.369 | 1163.5 | 2.000 |
| | D2 | 0.415 | 0.484 | 0.651 | 0.468 | 0.360 | 0.351 | 0.150 | 0.082 | 0.263 | 0.369 | 2017.3 | 3.000 |
| | HegelADD | 0.573 | 0.527 | 0.651 | 0.610 | 0.370 | 0.441 | **0.229** | 0.110 | **0.283** | 0.451 | 1404.4 | 2.213 |
| Llama-3.1-8B-Instruct | Zero-shot | **0.769** | 0.684 | 0.842 | 0.837 | **0.620** | 0.563 | 0.217 | 0.144 | 0.212 | 0.584 | 198.3 | 1.000 |
| | CoT | 0.642 | 0.707 | 0.833 | 0.694 | 0.595 | 0.520 | 0.343 | 0.338 | 0.227 | 0.570 | 377.6 | 1.000 |
| | SC@5 | 0.765 | 0.678 | **0.885** | **0.850** | 0.490 | **0.628** | 0.235 | **0.452** | **0.318** | **0.632** | 2894.6 | 5.000 |
| | Self-Refine | 0.291 | 0.555 | 0.704 | 0.377 | 0.160 | 0.190 | 0.188 | 0.288 | 0.126 | 0.333 | 2962.3 | 3.000 |
| | D1 | 0.738 | 0.642 | 0.811 | 0.797 | 0.450 | 0.587 | 0.246 | 0.334 | 0.298 | 0.588 | 1414.5 | 2.000 |
| | D2 | 0.740 | 0.645 | 0.811 | 0.798 | 0.455 | 0.585 | 0.251 | 0.334 | 0.283 | 0.589 | 2454.5 | 3.000 |
| | HegelADD | 0.691 | **0.709** | 0.831 | 0.774 | 0.525 | 0.585 | **0.394** | 0.408 | 0.283 | 0.617 | 1436.5 | 1.866 |
| Qwen3-8B | Zero-shot | **0.829** | 0.658 | 0.787 | 0.895 | **0.780** | 0.642 | 0.228 | 0.226 | 0.248 | 0.625 | 205.1 | 1.000 |
| | CoT | 0.809 | 0.673 | 0.927 | 0.912 | 0.695 | 0.779 | 0.529 | 0.652 | 0.470 | 0.716 | 371.0 | 1.000 |
| | SC@5 | **0.829** | 0.685 | 0.938 | **0.917** | **0.780** | 0.773 | 0.496 | 0.670 | 0.434 | 0.725 | 2595.4 | 5.000 |
| | Self-Refine | 0.446 | 0.673 | 0.927 | 0.693 | 0.590 | 0.592 | 0.576 | 0.676 | 0.359 | 0.615 | 2002.9 | 3.000 |
| | D1 | 0.813 | 0.653 | **0.942** | 0.904 | 0.700 | **0.799** | 0.615 | 0.710 | 0.470 | 0.734 | 1281.9 | 2.000 |
| | D2 | 0.815 | 0.678 | 0.927 | 0.912 | 0.725 | **0.799** | 0.624 | 0.712 | 0.475 | 0.741 | 2141.4 | 3.000 |
| | HegelADD | 0.794 | **0.695** | 0.929 | 0.909 | 0.720 | 0.787 | **0.668** | **0.766** | **0.480** | **0.750** | 1058.6 | 1.705 |
| Mistral-Small-3.2-24B-Instruct | Zero-shot | 0.765 | 0.665 | 0.645 | 0.863 | 0.850 | 0.659 | 0.261 | 0.214 | 0.293 | 0.609 | 171.1 | 1.000 |
| | CoT | 0.833 | 0.785 | **0.953** | **0.930** | 0.850 | 0.799 | 0.545 | 0.576 | **0.490** | 0.779 | 355.0 | 1.000 |
| | SC@5 | 0.825 | 0.782 | 0.933 | 0.927 | **0.860** | 0.775 | 0.375 | 0.314 | 0.379 | 0.719 | 2660.2 | 5.000 |
| | Self-Refine | 0.815 | 0.624 | 0.942 | 0.881 | 0.800 | 0.763 | 0.451 | 0.258 | 0.318 | 0.693 | 2235.6 | 3.000 |
| | D1 | 0.809 | 0.787 | 0.909 | 0.922 | 0.800 | 0.762 | 0.450 | 0.230 | 0.303 | 0.711 | 1309.4 | 2.000 |
| | D2 | 0.811 | 0.785 | 0.907 | 0.922 | 0.800 | 0.764 | 0.450 | 0.230 | 0.313 | 0.711 | 2266.8 | 3.000 |
| | HegelADD | **0.836** | **0.795** | 0.920 | 0.929 | 0.835 | **0.805** | **0.655** | **0.596** | 0.424 | **0.792** | 1143.7 | 1.726 |

methods use deterministic decoding, while SC@5 uses temperature 0.7 and top $p$ 0.95 as a repeated sampling baseline. We use 5,000-sample paired bootstrap for reliability analysis, with details in Appendix D.

## 5 Main Results

Table 2 summarizes accuracy by reporting group and overall cost statistics. Figure 3 compares adaptive revision and fixed depth refinement on an example weighted basis. The main table centers on the controller trained on Qwen3-8B and also reports transfer to Llama and Mistral backbones. Appendices B and C provide the per dataset accuracy breakdown and the matched cost table.

### 5.1 Adaptive Revision Improves Accuracy and Cost Tradeoff

Table 2 and Figure 3 show that HegelADD consistently provides a better accuracy and cost tradeoff than fixed depth revision baselines. In the Qwen3-8B Overall setting, HegelADD reaches 0.750 accuracy with 1.705 average calls, yielding a more efficient performance combination than D1 at 0.734/2.000 and D2 at 0.741/3.000. In the OOD Hard regime, which includes MATH-500, MMLU-Pro, and GPQA-Diamond, HegelADD uses the repair potential of deeper revision while limiting harmful over refinement. This improvement is especially clear in heterogeneous OOD regimes where deeper revision has both potential gains and regression risk. The result suggests that adaptive stopping, which decides whether to continue revision for each input, is more effective than applying the same refinement depth to every input.

### 5.2 Generalization Across Backbone Models

We further evaluate whether the adaptive revision policy trained on Qwen3-8B generalizes to other backbone LLMs. On Llama-3.1-8B-Instruct, HegelADD outperforms CoT and fixed depth revision baselines while using far fewer model calls than SC@5. On Llama-3.2-3B-Instruct, it remains competitive with revision based baselines, although weaker backbones also show reduced stability in CoT style reasoning itself. A similar pattern

Table 3: Representative qualitative examples from matched `Qwen3-8B` evaluation traces. Case 1 shows answer preservation under unnecessary refinement, while Case 2 shows repair-preserving stopping that avoids harmful over-refinement.

**Case 1: Easy answer preservation.** `ARC-Easy` question. Samples are made of two or more minerals, and the gold answer is `B`, "rocks." The initial CoT is already correct, but iterative self-refinement changes the answer to `C`.

| Method | Ans. | Pred. | Decision / behavior | Cost |
|---|---|---|---|---|
| CoT | ✓ | B | Initial CoT directly identifies rocks as a mixture of minerals. | 1 call / 228 tok. |
| SC@5 | ✓ | B | All five samples vote for B. | 5 calls / 1154 tok. |
| Fixed D2 | ✓ | B | Both verifier and reviser passes keep the original answer. | 3 calls / 1234 tok. |
| Self-Refine | ✗ | C | Self-feedback incorrectly pushes the answer from rocks to elements. | 3 calls / 1095 tok. |
| `HegelADD` | ✓ | **B** | **`CALL_D1` → `STOP_D1`, keeps the correct answer and stops before deeper revision.** | 2 calls / 684 tok. |

**Case 2: Hard repair + stopping.** `MMLU-Pro` forced-convection similitude problem. The gold answer is `G`. The initial CoT is wrong, but the first verifier and reviser step recovers the correct answer. The main risk is that further revision can overwrite this repair.

| Method | Ans. | Pred. | Decision / behavior | Cost |
|---|---|---|---|---|
| CoT | ✗ | N/A | Initial reasoning fails to reach the correct similitude relation. | 1 call / 978 tok. |
| SC@5 | ✗ | A | Repeated sampling still selects an incorrect answer. | 5 calls / 5925 tok. |
| Fixed D2 | ✗ | A | D1 reaches G, but D2 changes it back to A. | 3 calls / 4885 tok. |
| Self-Refine | ✓ | G | Iterative refinement recovers the answer, but uses a full three-call path. | 3 calls / 5341 tok. |
| `HegelADD` | ✓ | **G** | **`CALL_D1` → `STOP_D1`, accepts the repair and avoids the harmful second revision.** | 2 calls / 3211 tok. |

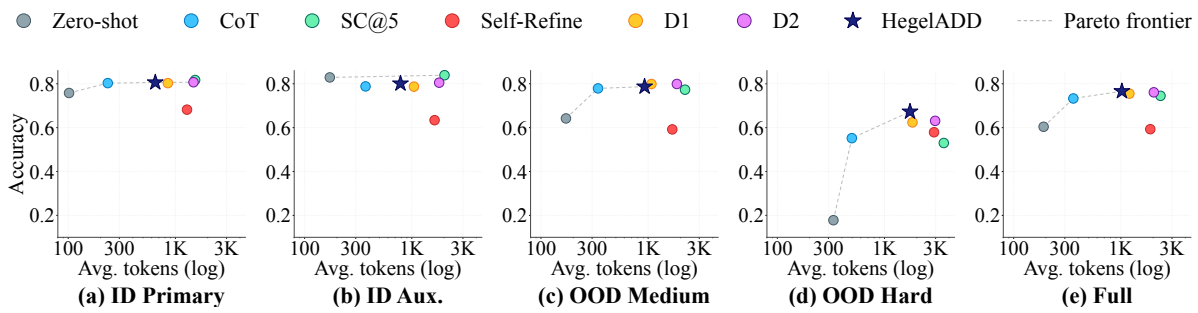



Figure 3: Comparison between adaptive depth and fixed depth baselines. On `Qwen3-8B`, `HegelADD` achieves higher accuracy than the fixed two step revision baseline with fewer average calls in the OOD Medium+Hard and Full settings.

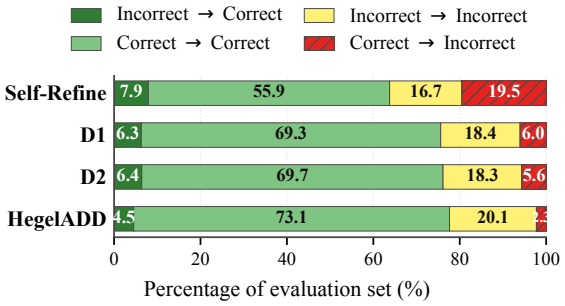



Figure 4: CoT to final transition breakdown for `HegelADD`. Each bar shows transitions from the initial CoT answer on the matched `Qwen3-8B` evaluation set. `HegelADD` preserves repair transitions while substantially reducing the regression region.

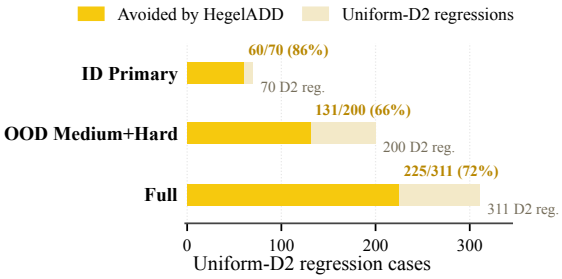
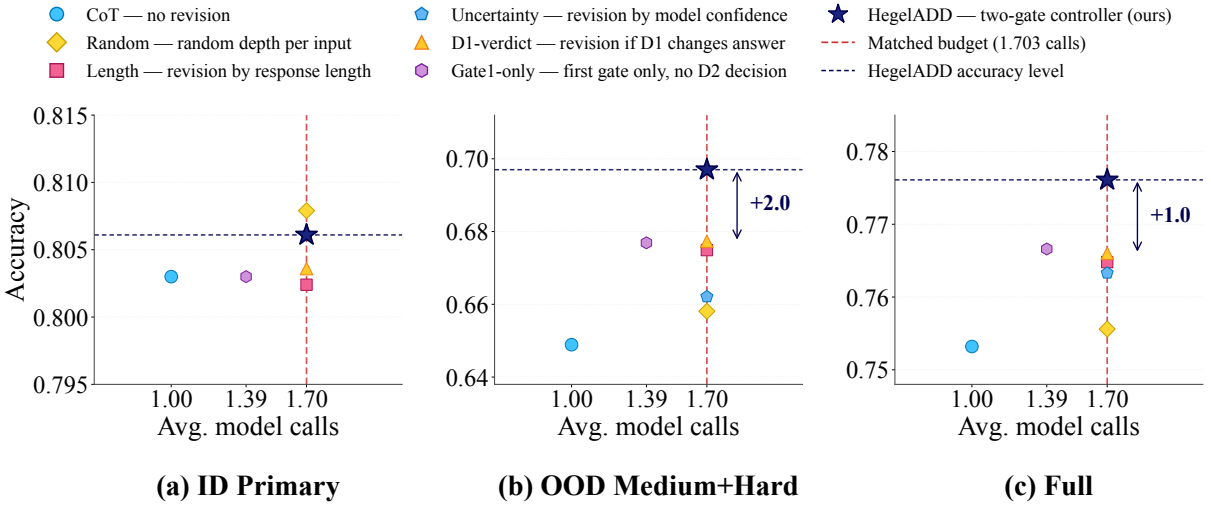



Figure 5: Uniform D2 regression avoided by `HegelADD` on matched `Qwen3-8B` evaluation traces. Light blue bars denote cases where fixed two step revision changes a correct CoT answer into an incorrect final answer, and dark yellow bars denote the subset avoided by early stopping in `HegelADD`.

appears on `Mistral-Small-3.2-24B-Instruct`. Specifically, `HegelADD` achieves 0.792 overall accuracy with 1.726 calls on `Mistral-Small-3.2-24B-Instruct`, exceeding the 0.779 accuracy of CoT and the 0.711 accuracy of fixed depth D1/D2. Compared with SC@5, it also achieves higher accuracy with far fewer than 5.000 calls. However, relative to CoT, accuracy improves while call count increases. The Mistral result should therefore be interpreted not




**(a) ID Primary** **(b) OOD Medium+Hard** **(c) Full**

Figure 6: Adaptive depth selection beyond call budget with the `Qwen3-8B` backbone. Same cost policies use the same cached candidate pool and depth budget, and differ only in the revision allocation policy.

as unconditional cost reduction, but as a better accuracy and cost tradeoff relative to fixed depth and repeated sampling baselines. Overall, these results show that `HegelADD` is not limited to a single backbone and can provide a stable accuracy and cost tradeoff across diverse LLM settings. Table **??** gives the corresponding per dataset transfer breakdown.

### 5.3 Qualitative Examples

Table 3 illustrates the adaptive stopping behavior of `HegelADD` in real reasoning traces. In the first case, the initial CoT is already correct and additional refinement causes incorrect regression, but `HegelADD` stops revision and preserves the correct answer. In the second case, `HegelADD` stops after a useful first revision and avoids unnecessary degradation. These cases show that `HegelADD` selectively controls refinement depth based on input specific revision utility. Appendix F reports additional matched outputs and full trace boxes in Appendix F.2.

## 6 Analysis and Ablations

This section analyzes the adaptive revision behavior and controller design of `HegelADD` through ablation and diagnostic experiments. Unless otherwise specified, the following diagnostic analyses are conducted on matched `Qwen3-8B` evaluation traces. Table 2 reports transfer across additional backbones.

### 6.1 Revision-Induced Regression Diagnostics

Figure 4 shows that the gains of `HegelADD` come from adaptive stopping rather than simply from deeper refinement. Fixed depth revision and Self Refine can repair incorrect CoT answers, but they also induce revision related regression by changing correct CoT answers into incorrect final answers. In contrast, `HegelADD` preserves repair transitions while sharply reducing the Correct $\rightarrow$ Incorrect transition rate, achieving both the smallest regression region and the highest net repair.

Figure 5 separates this effect under two step revision. A substantial fraction of regression cases caused by fixed D2 is avoided by early stopping in `HegelADD`, suggesting that the main source of improvement is harmful revision avoidance rather than additional refinement itself.

### 6.2 Adaptive Depth Selection Beyond Call Budget

Figure 6 compares policies while holding the same cached candidate pool and average depth budget fixed and changing only the revision allocation policy. The learned two gate policy consistently outperforms random, length based, and uncertainty based heuristics, with the largest improvement in the OOD Medium+Hard setting. This suggests that the gains of `HegelADD` arise from learned input wise depth selection rather than from simply increasing the number of calls.

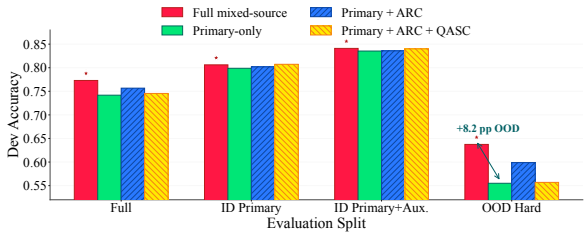



Figure 7: Effect of training source diversity. The `Qwen3-8B` controller shows the strongest OOD robustness under heterogeneous mixed source training.

Table 4: Held-out feature-interface diagnostic for the `Qwen3-8B` controller. All variants are evaluated under the same replay protocol. Δ columns denote accuracy change (pp) relative to the full 22/45 visible-state interface. OOD M+H denotes OOD Medium+Hard.

| Variant | Gate | Feats | Full | Δ (pp) | OOD M+H | Δ (pp) |
|---|---|---|---|---|---|---|
| Full interface | both | 22/45 | **0.7747** | N/A | **0.6958** | N/A |
| No action identity | G1 | 20/45 | 0.7556 | -1.91 | 0.6541 | -4.17 |
| No CoT/problem surface | G1 | 14/45 | 0.7540 | -2.07 | 0.6481 | -4.77 |
| No format checks | G1 | 15/45 | 0.7659 | -0.88 | 0.6761 | -1.97 |
| No D1 state | G2 | 22/38 | 0.7740 | -0.07 | 0.6946 | -0.12 |
| No action identity | G2 | 22/41 | 0.7743 | -0.04 | **0.6958** | 0.00 |

### 6.3 Effect of Training Source Diversity

Figure 7 compares the effect of the controller training source. Primary only training is competitive on ID Primary but reduces OOD robustness, whereas Full training with heterogeneous sources is most stable in the Full and OOD Hard settings. This shows that the adaptive revision policy generalizes better when it is learned from diverse reasoning regimes.

### 6.4 Feature Interface Analysis

Table 4 analyzes the contribution of each feature group under the online visibility constraint. The full 22/45 interface is the most stable. In feature removal ablations, removing Gate1 CoT/problem surface features and action identity features causes the largest drops. This suggests that initial CoT surface signals and Gate1 action level distinctions are important visible signals for depth selection. In contrast, removing Gate2 D1 state or action identity features has only limited impact, indicating that major depth selection signals are already visible at the first decision point. This ablation supports that `HegelADD` selects online revision depth using decision time visible states rather than future candidates or oracle correctness.

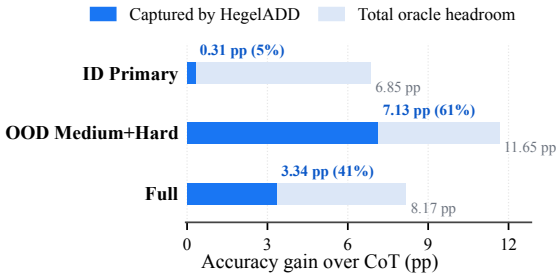



Figure 8: Candidate oracle headroom analysis on the matched `Qwen3-8B` evaluation set. Light bars denote total oracle headroom, and dark bars denote the accuracy gain actually recovered by `HegelADD`. Values in parentheses are oracle recovery rates.

### 6.5 Candidate-Oracle Headroom Analysis

Figure 8 shows the accuracy gain that `HegelADD` recovers relative to a candidate oracle. The candidate oracle is an upper bound that can select a correct candidate post hoc among CoT, D1, and D2 candidates. `HegelADD` achieves a much higher oracle recovery rate on OOD Medium+Hard than on ID Primary, and also recovers substantial oracle headroom in the Full aggregate. This shows that the benefits of adaptive revision are concentrated in harder reasoning regimes with larger revision headroom and greater input heterogeneity.

## 7 Conclusion

This paper formulates iterative refinement as adaptive revision depth selection for each input and proposes `HegelADD`, a lightweight controller framework that decides whether to continue post CoT refinement. `HegelADD` performs online revision stopping on top of a frozen base LLM and dynamically controls only the revision depth, without training an additional generator or fine tuning the backbone. Experiments show that `HegelADD` provides a better accuracy and cost tradeoff than fixed depth refinement across reasoning regimes and backbone LLMs. This pattern is also observed with the 24B scale `Mistral-Small-3.2-24B-Instruct` backbone, suggesting that adaptive stopping may not be limited to `Qwen3-8B` trajectories. Diagnostic analyses further confirm that iterative refinement can exhibit non monotonic behavior and revision induced regression, and that adaptive stopping is a key component of efficient test time reasoning.

## Limitations

This study has several limitations. First, HegelADD does not fine tune the base LLM, but it still requires offline revision trajectories for gate training. Second, the paper considers only a maximum two step revision setting, so extensions to deeper iterative refinement or other verifier and reviser prompting environments require further validation. Third, the evaluation mainly uses English reasoning benchmarks, and generalization to long form generation tasks or real user interactions remains future work. Fourth, although we include the 24B scale `Mistral-Small-3.2-24B-Instruct` backbone, we have not yet tested substantially larger frontier scale or closed source models. Finally, token counts depend on each backbone tokenizer, so cross backbone token comparisons should be interpreted as model specific efficiency measurements rather than a universal cost unit.

## Ethical Considerations

This work aims to improve the efficiency of test-time reasoning by avoiding unnecessary refinement calls. The proposed controller does not require new private or sensitive data, and the experiments use existing academic reasoning benchmarks. Reducing redundant LLM calls may also lower computational cost and associated environmental impact.

However, adaptive stopping is not a reliability guarantee. The controller may stop before a useful correction or allow a harmful revision, leading to incorrect final answers. Therefore, the method should not be used without external validation or human oversight in high-stakes domains such as medical, legal, financial, or safety-critical applications.

## References


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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*.

Kaiyan Chang, Yonghao Shi, Chenglong Wang, Hang Zhou, Chi Hu, Xiaoqian Liu, Yingfeng Luo, Yuan Ge, Tong Xiao, and Jingbo Zhu. 2025. Step-level verifier-guided hybrid test-time scaling for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. In *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Thinkless: LLM learns when to think. *Preprint*, arXiv:2505.13379.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did Aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. In *Transactions of the Association for Computational Linguistics*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. *Preprint*, arXiv:2310.01798.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Dodds, Nova DasSarma, Eli Tran-Johnson, and 1 others. 2022. Language models (mostly) know what they know. In *arXiv preprint arXiv:2207.05221*.

Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. 2020. QASC: A dataset for question answering via sentence composition. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*.

Chenwei Lou, Zewei Sun, Xinnian Liang, Meng Qu, Wei Shen, Wenqi Wang, Yuntao Li, Qingping Yang, and Shuangzhi Wu. 2025. AdaCoT: Pareto-optimal adaptive chain-of-thought triggering via reinforcement learning. *Preprint*, arXiv:2505.11896.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*.

Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. 2023. SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Mistral AI. 2025. Mistral-Small-3.2-24B-Instruct-2506. Hugging Face model card.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. *Preprint*, arXiv:2311.12022.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process- and outcome-based feedback. In *arXiv preprint arXiv:2211.14275*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024a. 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*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, and 1 others. 2024b. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *Preprint*, arXiv:2406.01574.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. 2025b. Dynamic early exit in reasoning models. *Preprint*, arXiv:2504.15895.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Zhengyi Zhao, Shubo Zhang, Zezhong Wang, Huimin Wang, Yutian Zhao, Bin Liang, Yefeng Zheng, Binyang Li, Kam-Fai Wong, and Xian Wu. 2025. $T^2$: An adaptive test-time scaling strategy for contextual question answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3731–3756.

Table 5: Dataset composition of the paper-facing evaluation suite. The table follows the exact grouping used by Table 2. Aux.E aggregates six auxiliary datasets, while Aux.M is the HellaSwag auxiliary column. Dataset identity is used only for reporting and analysis.

| Reporting block | Main column | Underlying dataset(s) | $N$ | Answer format |
|---|---|---|---|---|
| ID Primary | CSQA | CommonsenseQA | 550 | Multiple choice |
| ID Primary | SQA | StrategyQA | 550 | Yes/no |
| ID Primary | GSM8K | GSM8K | 550 | Numeric |
| ID Aux. | Aux.E | ARC-Easy, SciQ | 400 | Multiple choice |
| ID Aux. | Aux.E | BoolQ, QASC, OpenBookQA, ARC-Challenge | 800 | Boolean / multiple choice |
| ID Aux. | Aux.M | HellaSwag | 200 | Multiple choice |
| OOD Medium | MMLU | MMLU | 1000 | Multiple choice |
| OOD Hard | MMLU-Pro | MMLU-Pro | 800 | Multiple choice |
| OOD Hard | MATH-500 | MATH-500 | 500 | Math expression / numeric |
| OOD Hard | GPQA-Diamond | GPQA-Diamond | 198 | Multiple choice |

## A Evaluation Suite

This appendix follows the reporting structure used by Table 2. All auxiliary tables use the same columns as the main table: ID Primary, ID Aux., OOD Medium, and OOD Hard. These groups are reporting units only; the controller never receives the reporting group, dataset name, problem id, gold answer, or correctness label as an input feature. Table 5 gives the dataset composition behind the main table.

## B Full Dataset-Level Results

This section unpacks the grouped accuracy columns in Table 2. The goal is not to introduce a second evaluation protocol, but to make clear how each paper-facing reporting block is constructed. We therefore keep the same grouping as the main table: CSQA, SQA, and GSM8K form ID Primary; Aux.E and Aux.M are ID auxiliary aggregates; MMLU is the OOD Medium block; and MMLU-Pro, MATH-500, and GPQA-Diamond form OOD Hard.

Table 6 first gives a fine-grained Qwen3-8B decomposition. We include it because the Qwen3-8B evaluation records include subject-level metadata for MMLU-Pro; this table is useful for diagnosing where the aggregate OOD Hard gain comes from.

Table 6 gives the fine-grained dataset decomposition for all four backbones in the main table. The column order matches the reporting groups used in Table 2, while expanding Aux.E, Aux.M, and MMLU-Pro into their source dataset or subject-level components. This view is useful for checking whether an aggregate gain comes from broad behavior or from a small number of subgroups. Table **??** then reports the same results in the compact grouped form used in the main paper.

**Interpretation.** The cross-backbone table shows that the benefit of adaptive revision depth is concentrated in settings where revision candidates contain recoverable headroom. On Qwen3-8B, HegelADD gives the best overall result (0.750) and is strongest on the hard columns MMLU-Pro, MATH-500, and GPQA. On Mistral-Small-3.2-24B-Instruct, the same pattern becomes stronger: HegelADD reaches the best overall accuracy (0.792), improves the OOD Medium and two of the three OOD Hard columns, and avoids the large cost of SC@5. The Llama backbones are more mixed. Llama-3.1-8B-Instruct still benefits from adaptive depth relative to CoT, D1, and D2 overall, but SC@5 remains stronger on several ID columns. Llama-3.2-3B-Instruct is a useful lower-capacity stress test: HegelADD improves over the fixed revision baselines, but the direct zero-shot format remains strongest on many easy ID columns. This supports the main claim that the controller is most useful when the base model can produce repairable revision candidates, rather than when the first answer is already the dominant strategy.

Table 6: Fine-grained dataset-level accuracy decomposition across backbones. D1 and D2 denote one-pass and two-pass verifier–reviser revision. The MP-* columns are subject-level MMLU-Pro components retained in the source artifacts. This table is a diagnostic expansion of Table 2.

| Backbone | Method | Dataset-level columns used to construct Table 2 | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CSQA | ARC-E | SciQ | ARC-C | BoolQ | QASC | OBQA | GSM8K | StratQA | HSwag | MP-Bio | MMLU | MP-Phys | MP-CS | GPQA | MP-Math | MP-Chem | MP-Eng | MATH-500 |
| Llama-3.2-3B-Instruct | Zero-shot | 0.682 | 0.860 | 0.850 | 0.715 | 0.815 | 0.715 | 0.695 | 0.691 | 0.642 | 0.555 | 0.535 | 0.556 | 0.163 | 0.292 | 0.242 | 0.113 | 0.146 | 0.116 | 0.000 |
| | CoT | 0.533 | 0.600 | 0.760 | 0.455 | 0.770 | 0.405 | 0.455 | 0.476 | 0.553 | 0.420 | 0.327 | 0.422 | 0.120 | 0.246 | 0.237 | 0.113 | 0.146 | 0.116 | 0.094 |
| | SC@5 | 0.476 | 0.615 | 0.495 | 0.515 | 0.725 | 0.410 | 0.490 | 0.753 | 0.549 | 0.425 | 0.525 | 0.400 | 0.127 | 0.154 | 0.242 | 0.072 | 0.125 | 0.070 | 0.126 |
| | Self-Refine | 0.464 | 0.595 | 0.685 | 0.440 | 0.620 | 0.370 | 0.435 | 0.689 | 0.529 | 0.390 | 0.307 | 0.401 | 0.145 | 0.215 | 0.253 | 0.133 | 0.097 | 0.147 | 0.252 |
| | D1 | 0.415 | 0.565 | 0.380 | 0.495 | 0.675 | 0.315 | 0.385 | 0.651 | 0.482 | 0.365 | 0.455 | 0.350 | 0.108 | 0.215 | 0.258 | 0.062 | 0.090 | 0.124 | 0.076 |
| | D2 | 0.415 | 0.565 | 0.380 | 0.495 | 0.665 | 0.315 | 0.385 | 0.651 | 0.484 | 0.360 | 0.465 | 0.351 | 0.108 | 0.215 | 0.263 | 0.062 | 0.090 | 0.124 | 0.082 |
| | HegelADD | 0.573 | 0.655 | 0.755 | 0.545 | 0.765 | 0.470 | 0.470 | 0.651 | 0.527 | 0.370 | 0.396 | 0.443 | 0.187 | 0.277 | 0.283 | 0.185 | 0.229 | 0.202 | 0.110 |
| Llama-3.1-8B-Instruct | Zero-shot | 0.769 | 0.910 | 0.890 | 0.785 | 0.865 | 0.805 | 0.765 | 0.842 | 0.684 | 0.620 | 0.574 | 0.563 | 0.157 | 0.323 | 0.212 | 0.113 | 0.208 | 0.132 | 0.000 |
| | CoT | 0.642 | 0.690 | 0.705 | 0.745 | 0.860 | 0.580 | 0.585 | 0.833 | 0.707 | 0.595 | 0.594 | 0.520 | 0.259 | 0.462 | 0.227 | 0.369 | 0.319 | 0.178 | 0.338 |
| | SC@5 | 0.765 | 0.935 | 0.890 | 0.830 | 0.900 | 0.755 | 0.790 | 0.885 | 0.678 | 0.490 | 0.574 | 0.628 | 0.229 | 0.400 | 0.318 | 0.133 | 0.174 | 0.116 | 0.452 |
| | Self-Refine | 0.291 | 0.300 | 0.355 | 0.265 | 0.635 | 0.410 | 0.295 | 0.704 | 0.555 | 0.160 | 0.257 | 0.190 | 0.163 | 0.185 | 0.126 | 0.200 | 0.201 | 0.132 | 0.288 |
| | D1 | 0.738 | 0.890 | 0.880 | 0.755 | 0.795 | 0.710 | 0.750 | 0.811 | 0.642 | 0.450 | 0.505 | 0.587 | 0.253 | 0.338 | 0.298 | 0.174 | 0.201 | 0.147 | 0.334 |
| | D2 | 0.740 | 0.895 | 0.875 | 0.760 | 0.785 | 0.720 | 0.755 | 0.811 | 0.645 | 0.455 | 0.515 | 0.585 | 0.259 | 0.354 | 0.283 | 0.179 | 0.201 | 0.147 | 0.334 |
| | HegelADD | 0.691 | 0.830 | 0.815 | 0.780 | 0.865 | 0.685 | 0.670 | 0.831 | 0.709 | 0.525 | 0.624 | 0.585 | 0.337 | 0.538 | 0.283 | 0.426 | 0.354 | 0.209 | 0.408 |
| Qwen3-8B | Zero-shot | **0.829** | 0.970 | 0.935 | 0.905 | **0.895** | 0.820 | 0.845 | 0.787 | 0.658 | **0.780** | 0.653 | 0.642 | 0.229 | 0.323 | 0.247 | 0.108 | 0.174 | 0.085 | 0.226 |
| | CoT | 0.809 | **0.985** | 0.940 | 0.945 | 0.885 | **0.850** | 0.865 | 0.927 | 0.673 | 0.695 | 0.762 | 0.779 | 0.446 | 0.692 | 0.470 | 0.590 | 0.493 | 0.318 | 0.652 |
| | SC@5 | **0.829** | **0.985** | **0.945** | 0.940 | 0.870 | 0.845 | **0.920** | 0.938 | 0.685 | **0.780** | 0.782 | 0.773 | 0.416 | 0.677 | 0.434 | 0.544 | 0.479 | 0.233 | 0.670 |
| | Self-Refine | 0.446 | 0.770 | 0.760 | 0.705 | 0.830 | 0.585 | 0.510 | 0.927 | 0.673 | 0.590 | 0.653 | 0.592 | 0.536 | 0.585 | 0.359 | 0.626 | 0.576 | 0.488 | 0.676 |
| | D1 | 0.813 | **0.985** | 0.940 | **0.960** | 0.840 | 0.835 | 0.865 | **0.942** | 0.653 | 0.700 | 0.802 | **0.799** | 0.554 | 0.739 | 0.470 | 0.661 | 0.569 | 0.465 | 0.710 |
| | D2 | 0.815 | **0.985** | **0.945** | 0.950 | 0.860 | **0.850** | 0.885 | 0.927 | 0.678 | 0.725 | **0.812** | **0.799** | 0.560 | 0.739 | 0.475 | 0.672 | 0.549 | 0.512 | 0.712 |
| | HegelADD | 0.794 | 0.980 | 0.935 | 0.950 | 0.865 | 0.845 | 0.880 | 0.929 | **0.695** | 0.720 | 0.792 | 0.787 | **0.596** | **0.785** | **0.480** | **0.723** | **0.604** | **0.589** | **0.766** |
| Mistral-Small-3.2-24B-Instruct | Zero-shot | 0.765 | 0.940 | 0.920 | 0.870 | 0.815 | 0.770 | 0.865 | 0.645 | 0.665 | 0.850 | 0.653 | 0.659 | 0.241 | 0.385 | 0.293 | 0.159 | 0.132 | 0.217 | 0.000 |
| | CoT | 0.833 | 0.985 | 0.955 | 0.955 | 0.900 | 0.860 | 0.925 | 0.953 | 0.785 | 0.850 | 0.802 | 0.799 | 0.470 | 0.692 | 0.490 | 0.605 | 0.521 | 0.302 | 0.576 |
| | SC@5 | 0.825 | 0.975 | 0.955 | 0.940 | 0.910 | 0.855 | 0.930 | 0.933 | 0.782 | 0.860 | 0.713 | 0.775 | 0.337 | 0.554 | 0.379 | 0.379 | 0.285 | 0.163 | 0.314 |
| | Self-Refine | 0.815 | 0.970 | 0.950 | 0.920 | 0.665 | 0.865 | 0.915 | 0.942 | 0.624 | 0.800 | 0.762 | 0.763 | 0.404 | 0.554 | 0.318 | 0.508 | 0.382 | 0.209 | 0.258 |
| | D1 | 0.809 | 0.975 | 0.945 | 0.915 | 0.930 | 0.855 | 0.915 | 0.909 | 0.787 | 0.800 | 0.673 | 0.762 | 0.422 | 0.538 | 0.303 | 0.497 | 0.375 | 0.279 | 0.230 |
| | D2 | 0.811 | 0.970 | 0.945 | 0.910 | 0.930 | 0.860 | 0.915 | 0.907 | 0.785 | 0.800 | 0.673 | 0.764 | 0.416 | 0.538 | 0.313 | 0.497 | 0.375 | 0.287 | 0.230 |
| | HegelADD | 0.836 | 0.985 | 0.960 | 0.935 | 0.905 | 0.860 | 0.925 | 0.920 | 0.795 | 0.835 | 0.802 | 0.805 | 0.548 | 0.723 | 0.424 | 0.651 | 0.611 | 0.380 | 0.596 |

## C Reporting Group-Level Cost

Accuracy alone does not show the central trade-off in this paper, because several baselines buy accuracy by making every input pay for additional generations. Table 7 therefore reports the same reporting groups as the main table, but with average total generated tokens and average model calls.

Table 7: Backbone-specific reporting-group inference cost. Each cell reports average total tokens and average model calls in the format "tokens / calls." Lower cost is better. HegelADD rows are shaded.

| Backbone | Method | ID Primary | | | ID Aux. | | OOD Medium | OOD Hard | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | CSQA | SQA | GSM8K | Aux.E | Aux.M | MMLU | MMLU-Pro | MATH-500 | GPQA-Diamond |
| Llama-3.2-3B-Instruct | Zero-shot | 104.1 / 1.000 | 62.4 / 1.000 | 204.7 / 1.000 | 125.6 / 1.000 | 250.0 / 1.000 | 177.1 / 1.000 | 352.5 / 1.000 | 217.8 / 1.000 | 424.9 / 1.000 |
| | CoT | 241.6 / 1.000 | 200.3 / 1.000 | 226.0 / 1.000 | 271.7 / 1.000 | 405.5 / 1.000 | 332.6 / 1.000 | 499.2 / 1.000 | 326.3 / 1.000 | 727.3 / 1.000 |
| | SC@5 | 1439.9 / 5.000 | 1587.9 / 5.000 | 1840.2 / 5.000 | 1631.3 / 5.000 | 2430.9 / 5.000 | 2121.2 / 5.000 | 3355.6 / 5.000 | 3505.3 / 5.000 | 6075.5 / 5.000 |
| | Self-Refine | 1441.0 / 3.000 | 1638.4 / 3.000 | 1986.5 / 3.000 | 1581.7 / 3.000 | 2073.7 / 3.000 | 1956.0 / 3.000 | 3147.1 / 3.000 | 3329.1 / 3.000 | 5581.7 / 3.000 |
| | D1 | 720.1 / 2.000 | 869.2 / 2.000 | 1026.4 / 2.000 | 805.9 / 2.000 | 1142.7 / 2.000 | 1026.1 / 2.000 | 1693.2 / 2.000 | 1733.5 / 2.000 | 2897.6 / 2.000 |
| | D2 | 1239.3 / 3.000 | 1582.3 / 3.000 | 1799.5 / 3.000 | 1385.6 / 3.000 | 1978.6 / 3.000 | 1776.2 / 3.000 | 2964.5 / 3.000 | 2967.9 / 3.000 | 4850.1 / 3.000 |
| | HegelADD | 832.6 / 2.204 | 897.1 / 1.967 | 1552.1 / 2.640 | 818.1 / 2.018 | 1489.4 / 2.425 | 1159.0 / 2.136 | 2056.5 / 2.179 | 2468.4 / 2.620 | 3377.9 / 2.192 |
| Llama-3.1-8B-Instruct | Zero-shot | 102.5 / 1.000 | 62.4 / 1.000 | 219.0 / 1.000 | 124.9 / 1.000 | 244.5 / 1.000 | 184.2 / 1.000 | 363.3 / 1.000 | 241.1 / 1.000 | 479.9 / 1.000 |
| | CoT | 245.7 / 1.000 | 264.2 / 1.000 | 311.1 / 1.000 | 289.5 / 1.000 | 440.9 / 1.000 | 363.7 / 1.000 | 541.6 / 1.000 | 468.7 / 1.000 | 890.6 / 1.000 |
| | SC@5 | 1857.5 / 5.000 | 2210.1 / 5.000 | 2291.5 / 5.000 | 2185.3 / 5.000 | 3043.3 / 5.000 | 2663.5 / 5.000 | 3651.1 / 5.000 | 4395.8 / 5.000 | 7819.7 / 5.000 |
| | Self-Refine | 2058.2 / 3.000 | 2391.2 / 3.000 | 2226.8 / 3.000 | 2253.6 / 3.000 | 3015.8 / 3.000 | 2555.2 / 3.000 | 3823.0 / 3.000 | 4550.1 / 3.000 | 7912.6 / 3.000 |
| | D1 | 853.0 / 2.000 | 1102.6 / 2.000 | 995.4 / 2.000 | 1018.8 / 2.000 | 1517.9 / 2.000 | 1235.2 / 2.000 | 2008.9 / 2.000 | 2119.7 / 2.000 | 4020.6 / 2.000 |
| | D2 | 1517.5 / 3.000 | 1967.0 / 3.000 | 1733.1 / 3.000 | 1795.8 / 3.000 | 2606.1 / 3.000 | 2168.4 / 3.000 | 3513.2 / 3.000 | 3602.3 / 3.000 | 6522.6 / 3.000 |
| | HegelADD | 889.8 / 2.011 | 828.1 / 1.675 | 817.9 / 1.738 | 844.1 / 1.718 | 1351.5 / 1.820 | 1093.3 / 1.771 | 2468.2 / 2.095 | 2472.4 / 2.114 | 4987.3 / 2.232 |
| Qwen3-8B | Zero-shot | 79.0 / 1.000 | 38.3 / 1.000 | 188.0 / 1.000 | 103.7 / 1.000 | 223.0 / 1.000 | 171.1 / 1.000 | 351.4 / 1.000 | 230.0 / 1.000 | 461.8 / 1.000 |
| | CoT | 247.9 / 1.000 | 191.5 / 1.000 | 262.2 / 1.000 | 262.5 / 1.000 | 451.3 / 1.000 | 340.6 / 1.000 | 501.7 / 1.000 | 406.2 / 1.000 | 675.4 / 1.000 |
| | SC@5 | 1391.3 / 5.000 | 1566.5 / 5.000 | 1623.5 / 5.000 | 1482.5 / 5.000 | 2422.4 / 5.000 | 2208.3 / 5.000 | 3292.4 / 5.000 | 3517.7 / 5.000 | 5854.0 / 5.000 |
| | Self-Refine | 1235.9 / 3.000 | 1302.5 / 3.000 | 1310.2 / 3.000 | 1287.3 / 3.000 | 1889.9 / 3.000 | 1679.5 / 3.000 | 2823.1 / 3.000 | 2808.1 / 3.000 | 3689.3 / 3.000 |
| | D1 | 793.5 / 2.000 | 876.1 / 2.000 | 883.1 / 2.000 | 827.8 / 2.000 | 1213.3 / 2.000 | 1073.3 / 2.000 | 1770.4 / 2.000 | 1787.4 / 2.000 | 2312.6 / 2.000 |
| | D2 | 1420.3 / 3.000 | 1463.2 / 3.000 | 1541.8 / 3.000 | 1462.6 / 3.000 | 2056.6 / 3.000 | 1853.2 / 3.000 | 2895.6 / 3.000 | 2894.6 / 3.000 | 3684.3 / 3.000 |
| | HegelADD | 624.4 / 1.640 | 647.9 / 1.689 | 674.3 / 1.698 | 594.5 / 1.531 | 926.9 / 1.600 | 926.2 / 1.729 | 1724.3 / 1.862 | 1788.9 / 1.980 | 1619.8 / 1.616 |
| Mistral-Small-3.2-24B-Instruct | Zero-shot | 87.6 / 1.000 | 48.1 / 1.000 | 195.1 / 1.000 | 110.4 / 1.000 | 234.4 / 1.000 | 154.5 / 1.000 | 316.7 / 1.000 | 224.8 / 1.000 | 342.6 / 1.000 |
| | CoT | 275.9 / 1.000 | 218.8 / 1.000 | 262.4 / 1.000 | 290.0 / 1.000 | 450.8 / 1.000 | 367.7 / 1.000 | 515.5 / 1.000 | 422.0 / 1.000 | 626.2 / 1.000 |
| | SC@5 | 1756.6 / 5.000 | 2106.3 / 5.000 | 1997.6 / 5.000 | 1930.0 / 5.000 | 3004.2 / 5.000 | 2560.3 / 5.000 | 3623.0 / 5.000 | 3877.4 / 5.000 | 6168.3 / 5.000 |
| | Self-Refine | 1594.5 / 3.000 | 1651.5 / 3.000 | 1667.3 / 3.000 | 1645.4 / 3.000 | 2347.6 / 3.000 | 2172.0 / 3.000 | 3311.0 / 3.000 | 3242.1 / 3.000 | 4114.7 / 3.000 |
| | D1 | 898.8 / 2.000 | 1008.7 / 2.000 | 1013.3 / 2.000 | 962.0 / 2.000 | 1386.2 / 2.000 | 1245.0 / 2.000 | 1971.6 / 2.000 | 1857.2 / 2.000 | 2403.2 / 2.000 |
| | D2 | 1616.9 / 3.000 | 1783.5 / 3.000 | 1821.9 / 3.000 | 1704.4 / 3.000 | 2369.9 / 3.000 | 2168.5 / 3.000 | 3330.2 / 3.000 | 3131.0 / 3.000 | 3973.3 / 3.000 |
| | HegelADD | 739.5 / 1.691 | 935.0 / 1.913 | 874.7 / 1.835 | 684.7 / 1.528 | 860.6 / 1.430 | 1046.2 / 1.691 | 2019.8 / 1.885 | 1839.3 / 1.848 | 1858.2 / 1.732 |

## D Paired Bootstrap Reliability

We estimate confidence intervals for `Qwen3-8B` by problem-id paired bootstrap. For each reporting split, we resample examples with replacement 5,000 times, compute the accuracy difference between `HegelADD` and each baseline on the same resampled examples, and report the 2.5th and 97.5th percentiles. Positive values indicate that `HegelADD` is more accurate than the compared baseline.

Table 8: Paired bootstrap accuracy differences between HegelADD and each baseline across retained backbone runs. Positive values favor HegelADD. Intervals are percentile confidence intervals from 5,000 paired bootstrap resamples.

| Backbone | Split | $N$ | vs. CoT | vs. SC@5 | vs. D1 | vs. D2 | Interpretation |
|---|---|---|---|---|---|---|---|
| Qwen3-8B | ID Primary | 1650 | +0.0030 [-0.0061,+0.0121] | -0.0115 [-0.0248,+0.0018] | +0.0036 [-0.0103,+0.0176] | -0.0006 [-0.0158,+0.0139] | competitive, not uniformly dominant |
| | OOD Medium+Hard | 2498 | +0.0713 [+0.0580,+0.0845] | +0.0833 [+0.0661,+0.1001] | +0.0240 [+0.0104,+0.0372] | +0.0204 [+0.0072,+0.0336] | stable hard-regime gain |
| | Full | 5548 | +0.0333 [+0.0263,+0.0402] | +0.0301 [+0.0209,+0.0391] | +0.0137 [+0.0056,+0.0218] | +0.0081 [+0.0000,+0.0160] | stable aggregate gain |
| Llama-3.2-3B-Instruct | ID Primary | 1650 | +0.0255 [+0.0024,+0.0485] | -0.0467 [-0.0685,-0.0255] | +0.0303 [+0.0164,+0.0448] | +0.0297 [+0.0158,+0.0430] | improves over CoT/fixed revision, below SC@5 |
| | OOD Medium+Hard | 2498 | -0.0200 [-0.0348,-0.0044] | -0.0172 [-0.0344,-0.0004] | +0.0140 [+0.0012,+0.0268] | +0.0116 [-0.0008,+0.0240] | helps over fixed revision only |
| | Full | 5548 | -0.0160 [-0.0274,-0.0045] | -0.0294 [-0.0415,-0.0175] | +0.0254 [+0.0164,+0.0344] | +0.0247 [+0.0159,+0.0333] | positive versus D1/D2, weaker than direct CoT-style baselines |
| Llama-3.1-8B-Instruct | ID Primary | 1650 | +0.0158 [+0.0012,+0.0303] | -0.0333 [-0.0527,-0.0139] | +0.0127 [-0.0036,+0.0285] | +0.0109 [-0.0042,+0.0261] | improves over CoT, below SC@5 |
| | OOD Medium+Hard | 2498 | +0.0336 [+0.0204,+0.0468] | -0.0052 [-0.0260,+0.0156] | +0.0328 [+0.0160,+0.0488] | +0.0332 [+0.0168,+0.0500] | stable against CoT and fixed revision |
| | Full | 5548 | +0.0342 [+0.0256,+0.0429] | -0.0278 [-0.0400,-0.0157] | +0.0160 [+0.0054,+0.0263] | +0.0151 [+0.0049,+0.0256] | aggregate gain over CoT and fixed revision |

# E Prompt Templates

This section records the prompt templates used by the evaluation protocol. We preserve the English template text because these are the strings passed to the generator and verifier.

## E.1 Shared Problem Rendering

Multiple-choice problems are rendered as follows.

**Shared Problem Rendering with Choices**

```
{question}
Choices:
{label}. {text}
...
```

Problems without choices use only the question text.

**Shared Problem Rendering without Choices**

```
{question}
```

## E.2 Answer-format Strings

**Answer Format Strings**

```
math: Return a single final mathematical
expression or value in the requested
format.
numeric or GSM8K/SVAMP: Return a single
final numeric answer, with units only if
needed.
boolean or StrategyQA: Return exactly yes
or no.
ECQA choice: Return the selected option as
1, 2, 3, 4, or 5.
choice with labels: Return exactly one
selected option letter from: {labels}.
choice fallback: Return the selected
option letter, such as A, B, C, D, or E.
fallback: Return the final answer in the
task's required format.
```

## E.3 Error-checking Rubric Strings

**Error Rubric Strings**

```
math: Check algebra, case conditions,
simplification, units or requested
format, and whether the final mathematical
expression directly answers the question.
numeric or GSM8K/SVAMP: Check arithmetic,
operation choice, unit consistency, copied
numbers, and whether the final numeric
answer directly answers the question.
boolean or StrategyQA: Check yes/no
polarity, unsupported assumptions,
contradictions with the question, and
whether the final answer is exactly yes
or no.
choice: Check whether the selected option
is best supported, whether a stronger
option was ignored, whether the reasoning
eliminates distractors correctly, and
whether the final answer is a valid
option.
fallback: Check for concrete arithmetic,
logical, factual, or answer-format errors
that change the final answer.
```

## E.4 Zero-shot Format Prompt

**Zero-shot Format Prompt**

```
Question:
{render_problem_text(problem)}
Answer format:
{task_answer_format(problem)}
Answer:
```

## E.5 CoT Format Prompt

**CoT Format Prompt**

```
<TASK_SYSTEM>
Solve step by step with visible reasoning
only. Do not emit <think> tags.
Return reasoning inside <reasoning> tags
and the answer inside <final> tags.
<ANSWER_FORMAT>
{task_answer_format(problem)}
</ANSWER_FORMAT>
</TASK_SYSTEM>
<PROBLEM>
{render_problem_text(problem)}
</PROBLEM>
```

```
<RESPONSE>
```

## E.6 SC@5 Prompt and Voting Policy

**SC@5 Prompt**

```
<TASK_SYSTEM>
Solve step by step with visible reasoning
only. Do not emit <think> tags.
Return reasoning inside <reasoning> tags
and the answer inside <final> tags.
</TASK_SYSTEM>
<PROBLEM>
{render_problem_text(problem)}
</PROBLEM>
<RESPONSE>
```

**SC@5 Voting Policy**

```
Generate 5 samples from the same CoT
prompt.
Extract a final answer from each sample.
Normalize answers by verifier type.
Select the most frequent normalized
answer.
Break ties by the first sample index.
```

## E.7 Self-Refine Prompt

**Self-Refine Feedback Prompt**

```
<SELF_REFINE_FEEDBACK_SYSTEM>
Provide self-feedback on a candidate
solution.
Focus on concrete answer-changing
arithmetic, logical, factual, and
answer-format errors.
Do not solve from scratch unless needed to
identify an issue.
Return only the requested tags.
<ANSWER_FORMAT>
{task_answer_format(problem)}
</ANSWER_FORMAT>
<ERROR_RUBRIC>
{task_error_rubric(problem)}
</ERROR_RUBRIC>
Return exactly:
<feedback>...</feedback>
</SELF_REFINE_FEEDBACK_SYSTEM>
<PROBLEM>
{render_problem_text(problem)}
</PROBLEM>
<CANDIDATE_SOLUTION>
{candidate_solution.strip()}
</CANDIDATE_SOLUTION>
<CANDIDATE_ANSWER>
{candidate_answer.strip()}
</CANDIDATE_ANSWER>
<SELF_REFINE_FEEDBACK_RESPONSE>
```

**Self-Refine Refinement Prompt**

```
<SELF_REFINE_REFINEMENT_SYSTEM>
Refine a candidate solution using the
provided self-feedback.
Preserve correct reasoning, fix concrete
errors, and return the final answer in the
required format.
Do not emit <think> tags.
Return only the requested tags.
<ANSWER_FORMAT>
{task_answer_format(problem)}
</ANSWER_FORMAT>
Return exactly:
<refined_reasoning>...</refined_reasoning>
<final>...</final>
</SELF_REFINE_REFINEMENT_SYSTEM>
<PROBLEM>
{render_problem_text(problem)}
</PROBLEM>
<CANDIDATE_SOLUTION>
{candidate_solution.strip()}
</CANDIDATE_SOLUTION>
<CANDIDATE_ANSWER>
{candidate_answer.strip()}
</CANDIDATE_ANSWER>
<SELF_FEEDBACK>
{feedback.strip()}
</SELF_FEEDBACK>
<SELF_REFINE_REFINEMENT_RESPONSE>
```

## E.8 One-pass Revision Prompt

**One-pass Revision Prompt**

```
<DAD_VERIFIER_SYSTEM>
Verify a candidate solution
conservatively.
Keep the candidate answer unless the
answer format and error rubric identify
a concrete error that changes the final
answer.
Return only the requested tags.
<ANSWER_FORMAT>
{task_answer_format(problem)}
</ANSWER_FORMAT>
<ERROR_RUBRIC>
{task_error_rubric(problem)}
</ERROR_RUBRIC>
Return exactly:
<verification>...</verification>
<verdict>KEEP|REVISE</verdict>
<revised_answer>...</revised_answer>
</DAD_VERIFIER_SYSTEM>
<PROBLEM>
{render_problem_text(problem)}
</PROBLEM>
<CANDIDATE_SOLUTION>
{cot_output.strip()}
</CANDIDATE_SOLUTION>
<CANDIDATE_ANSWER>
{candidate_answer.strip()}
</CANDIDATE_ANSWER>
<DAD_VERIFIER_RESPONSE>
```

**One-pass Final-answer Policy**

```
If <verdict> is REVISE and
<revised_answer> is non-empty, return
<revised_answer>.
Otherwise, keep the extracted CoT answer.
```

### E.9 Two-pass Revision Prompt

Two-pass revision uses the same initial CoT call and one-pass verifier and reviser prompt. The second verifier and reviser call appends the first verifier output to the candidate solution.

**Depth-2 Candidate Construction**

```
{cot_text.strip()}
<FIRST_DAD_REVISION>
{verifier_text.strip()}
</FIRST_DAD_REVISION>
```

**Two-pass Final-answer Policy**

```
If second-pass <verdict> is REVISE and
second-pass <revised_answer> is non-empty,
return second-pass <revised_answer>.
Otherwise, return the depth-1 answer.
```

### E.10 `HegelADD` Gate Inference

`HegelADD` does not introduce a new LLM prompt. It applies the learned gates over the CoT, D1, and D2 candidate states generated by the prompts above.

**`HegelADD` Gate Inference**

```
Gate1 visible state: x, y0
Decision: KEEP_COT or CALL_D1
Gate2 visible state: x, y0, y1
Decision: STOP_D1 or RUN_D2
If Gate1 returns KEEP_COT, return y0.
If Gate1 returns CALL_D1 and Gate2 returns
STOP_D1, return y1.
If Gate2 returns RUN_D2, return y2.
```

## F Qualitative Trace Excerpts

This section reports concrete logged examples from `Mistral-Small-3.2-24B-Instruct`. We use this backbone because the final main-table run contains all baseline outputs and the completed `HegelADD` replay. Each example satisfies a simple selection rule: `HegelADD` is correct and at least one baseline on the same problem is incorrect. The examples cover early stopping, exact-format preservation, one-step repair, and depth-2 repair. We report extracted final answers and inference cost as stored in the logs, while replacing internal identifiers with dataset-level descriptions.

Decision-count aggregates are reported in Table **??**.

### F.1 Matched Baseline Outputs Across Reporting Groups

Table 9 gives the extracted final answer for every baseline on three shared problems. These rows make the controller behavior explicit: `HegelADD` keeps the concise CoT answer when later revision changes the answer, calls one verifier and reviser pass when D1 repairs the initial answer, and spends the full depth-2 budget when the first revision still fails to produce a valid option.

### F.2 Full Matched Trace Boxes

**In StrategyQA, HegelADD Stops Before Harmful Revision (GT = yes)**

**Question**
*Can monkeys use QWERTY keyboards?*
**Rejected Baseline Outputs**
*Zero-shot: no* (35 tokens / 1 call).
*Self-Refine: no* (1807 tokens / 3 calls).
*D1: no* (1119 tokens / 2 calls).
*D2: no* (1803 tokens / 3 calls).
**Accepted HegelADD Output**
*The CoT trace notes that monkeys can be trained to press keys or buttons and concludes that they can use QWERTY keyboards in a rudimentary way.*
**Final Answer: yes**
**Gate path:** `KEEP_COT`. **Cost:** 235 tokens / 1 call.

**In ARC-Challenge, HegelADD Preserves the Exact Choice Format (GT = B)**

**Question**
*If an atom of hydrogen gives up one electron, what is its overall charge?*
**Choices:** A: -1, B: +1, C: -2, D: +2.
**Rejected Baseline Outputs**
*Zero-shot: +1* (75 tokens / 1 call).
*SC@5: B. +1* (1285 tokens / 5 calls).
*Self-Refine: B. +1* (1170 tokens / 3 calls).
*D1/D2: B. +1* (714/1365 tokens; 2/3 calls).
**Accepted HegelADD Output**
*A hydrogen atom has one proton and one electron. When it gives up one electron, it loses a negatively charged particle, leaving a net posi-*

*tive charge.*
**Final Answer: B**
**Gate path:** `KEEP_COT`. **Cost:** 188 tokens / 1 call.

### In QASC, HegelADD Avoids Multi-Answer Drift (GT = G)

**Question**
*What can cause fire in forests?*
**Choices:** A: Chemicals, B: a metal, C: wildfire, D: Type O, E: acids, F: heat, G: storms, H: oaks.
**Rejected Baseline Outputs**
*Zero-shot: C* (90 tokens / 1 call).
*SC@5: A, F, G* (2220 tokens / 5 calls).
*Self-Refine/D1/D2: C, F, G* (1785/1076/1871 tokens; 3/2/3 calls).
**Accepted HegelADD Output**
*The CoT trace distinguishes causes from fire types and selects storms because lightning from storms can ignite forest fires.*
**Final Answer: G**
**Gate path:** `KEEP_COT`. **Cost:** 386 tokens / 1 call.

### In MMLU, HegelADD Repairs a Ring-Theory Example at D1 (GT = D)

**Question**
*Let $R$ be a ring and let $U$ and $V$ be two-sided ideals of $R$. Which sets must also be ideals of $R$? I. $U + V$; II. $\{uv : u \in U, v \in V\}$; III. $U \cap V$.*
**Choices:** A: II only, B: III only, C: I and II only, D: I and III only.
**Rejected Baseline Outputs**
*Zero-shot: B* (320 tokens / 1 call).
*CoT: A* (559 tokens / 1 call).
*SC@5: A* (3440 tokens / 5 calls).
*Self-Refine: A* (3891 tokens / 3 calls).
**Accepted HegelADD Output**
*The verifier detects that the initial conclusion over the three candidate sets is inconsistent with the required option set and revises the final answer to the option that keeps the sum and intersection but excludes the pairwise-product set.*
**Final Answer: D**
**Gate path:** `CALL_DAD` → `STOP_D1`. **Cost:** 2278 tokens / 2 calls.

### In MMLU-Pro, HegelADD Repairs a Physics Unit Example at D1 (GT = F)

**Question**
*Find the mass and weight of the air at $20°C$ in a living room with a $4.0m \times 5.0m$ floor and a ceiling 3.0m high, and the mass and weight of an equal volume of water.*
**Gold option:** F: 5.9.
**Rejected Baseline Outputs**
*Zero-shot: A* (334 tokens / 1 call).
*CoT: A* (497 tokens / 1 call).
*SC@5: A* (3645 tokens / 5 calls).
*Self-Refine: A* (4042 tokens / 3 calls).
**Accepted HegelADD Output**
*The first verifier and reviser pass recognizes that the weight of the equal volume of water is approximately $5.886 \times 10^5$ N and revises the answer to the closest listed option.*
**Final Answer: F**
**Gate path:** `CALL_DAD` → `STOP_D1`. **Cost:** 2572 tokens / 2 calls.

### In MMLU-Pro, HegelADD Uses Depth 2 for a Mass-Transfer Example (GT = I)

**Question**
*Carbon dioxide from an aqueous solution has to be removed by an air stream in a wetted-wall column. Select the gas-phase mass transfer coefficient and mass flux at the stated operating point.*
**Gold option:** I: 89 ft/hr and 1.83 lb mole/hr ft$^2$.
**Rejected Baseline Outputs**
*Zero-shot: J* (374 tokens / 1 call).
*CoT: A* (672 tokens / 1 call).
*SC@5: A* (4555 tokens / 5 calls).
*Self-Refine: A* (5029 tokens / 3 calls).
*D1: no valid option* (2951 tokens / 2 calls).
**Accepted HegelADD Output**
*Gate1 calls revision because the CoT answer is unreliable. The first revision still cannot map the calculation to an option, so Gate2 runs depth 2. The second verifier and reviser pass selects the plausible option matching the expected coefficient and flux.*
**Final Answer: I**
**Gate path:** `CALL_DAD` → `RUN_D2`. **Cost:** 4707 tokens / 3 calls.

Table 9: `Mistral-Small-3.2-24B-Instruct` qualitative examples. Each block uses one shared problem and reports the extracted final answer, correctness, cost, and `HegelADD` gate path.

| Regime | Example | Method | Extracted final answer | OK | Tok./Calls | Gate path or note |
|---|---|---|---|---|---|---|
| ID Primary early stop | StrategyQA: keyboard use | Zero-shot | no | ✗ | 35 / 1 | N/A |
| | | CoT | yes | ✓ | 235 / 1 | N/A |
| | | SC@5 | no | ✗ | 2500 / 5 | verbose no-answer |
| | | Self-Refine | no | ✗ | 1807 / 3 | over-revision |
| | | D1 | no | ✗ | 1119 / 2 | revision flips answer |
| | | D2 | no | ✗ | 1803 / 3 | revision remains wrong |
| | | `HegelADD` | yes | ✓ | 235 / 1 | `KEEP_COT` |
| OOD Medium D1 repair | MMLU: ring ideals | Zero-shot | B | ✗ | 320 / 1 | incomplete extraction |
| | | CoT | A | ✗ | 559 / 1 | wrong ideal set |
| | | SC@5 | A | ✗ | 3440 / 5 | majority repeats wrong option |
| | | Self-Refine | A | ✗ | 3891 / 3 | refinement keeps wrong option |
| | | D1 | D | ✓ | 2278 / 2 | first revision repairs answer |
| | | D2 | D | ✓ | 3513 / 3 | correct but extra call |
| | | `HegelADD` | D | ✓ | 2278 / 2 | `CALL_DAD` → `STOP_D1` |
| OOD Hard D2 repair | MMLU-Pro: mass transfer | Zero-shot | J | ✗ | 374 / 1 | N/A |
| | | CoT | A | ✗ | 672 / 1 | incomplete calculation |
| | | SC@5 | A | ✗ | 4555 / 5 | majority repeats wrong option |
| | | Self-Refine | A | ✗ | 5029 / 3 | revision remains wrong |
| | | D1 | no valid option | ✗ | 2951 / 2 | first revision cannot map to choices |
| | | D2 | I | ✓ | 4707 / 3 | depth-2 repair |
| | | `HegelADD` | I | ✓ | 4707 / 3 | `CALL_DAD` → `RUN_D2` |

Table 10: Additional matched success cases from the `Mistral-Small-3.2-24B-Instruct` logs. Each cell reports "extracted answer; tokens/calls." `HegelADD` is correct in every row, while at least one baseline fails on the same input.

| Qualitative bucket | Example | Gold | Zero-shot | CoT | SC@5 | Self-Refine | D1 | D2 | `HegelADD` |
|---|---|---|---|---|---|---|---|---|---|
| ID Primary early stop | StrategyQA: keyboard use | yes | no ✗; 35/1 | yes ✓; 235/1 | no ✗; 2500/5 | no ✗; 1807/3 | no ✗; 1119/2 | no ✗; 1803/3 | yes ✓; 235/1; `KEEP` |
| ID Aux. format guard | ARC-Challenge: hydrogen ion | B | +1 ✗; 75/1 | B ✓; 188/1 | B.+1 ✗; 1285/5 | B.+1 ✗; 1170/3 | B.+1 ✗; 714/2 | B.+1 ✗; 1365/3 | B ✓; 188/1; `KEEP` |
| ID Aux. early stop | QASC: forest fire cause | G | C ✗; 90/1 | G ✓; 386/1 | A,F,G ✗; 2220/5 | C,F,G ✗; 1785/3 | C,F,G ✗; 1076/2 | C,F,G ✗; 1871/3 | G ✓; 386/1; `KEEP` |
| OOD Medium D1 repair | MMLU: ring ideals | D | B ✗; 320/1 | A ✗; 559/1 | A ✗; 3440/5 | A ✗; 3891/3 | D ✓; 2278/2 | D ✓; 3513/3 | D ✓; 2278/2; `D1` |
| OOD Hard D1 repair | MMLU-Pro: air and water weight | F | A ✗; 334/1 | A ✗; 497/1 | A ✗; 3645/5 | A ✗; 4042/3 | F ✓; 2572/2 | F ✓; 4270/3 | F ✓; 2572/2; `D1` |
| OOD Hard D2 repair | MMLU-Pro: mass transfer | I | J ✗; 374/1 | A ✗; 672/1 | A ✗; 4555/5 | A ✗; 5029/3 | none ✗; 2951/2 | I ✓; 4707/3 | I ✓; 4707/3; `D2` |