THINK SIMPLER TO SOLVE HARDER: DIFFICULTY-AWARE ADAPTIVE REASONING VIA TOKEN ENTROPY

Anonymous authors
Paper under double-blind review

ABSTRACT

Decomposing complex problems into subproblems can improve the reasoning capabilities of large language models (LLMs). However, choosing an appropriate degree of decomposition based on subproblem difficulty and limiting the impact of intermediate-answer errors on final results remain important challenges. We find that answer-token entropy, defined as the mean entropy of token distributions across positions in a generated answer, can reflect subproblem difficulty at suitable granularities. Building on this observation, we propose **DART**, a training-free algorithm for **D**ifficulty-aware **A**daptive **R**easoning through **T**oken entropy. At suitable granularities identified through one-time probing, DART uses answer-token entropy as a routing signal to adaptively choose between direct solving and further decomposition, balancing solution quality and inference overhead. We also design a confidence-aligned self-critique mechanism that jointly constrains candidate evaluation scores and score-token confidence to guide the evaluation and refinement of intermediate answers, restraining the impact of errors on subsequent reasoning. Results aggregated across three tasks show that, compared to the baseline with the lowest average error on each backbone, DART reduces average error by 22%–57% and average online inference cost by 51%–72%.

1 INTRODUCTION

LLMs have demonstrated strong performance across a wide range of complex reasoning tasks (Zhu et al., 2026; Shen et al., 2026; Chen et al., 2026). Recent work on DeepSeek-R1 (Guo et al., 2025) demonstrates that scaling inference-time computation can markedly enhance model performance, particularly on problems requiring sustained multi-step deliberation, thereby driving the development of advanced reasoning algorithms (Liu et al., 2026; Qin et al., 2025).

Existing methods improve problem solving primarily by organizing reasoning structures and adjusting reasoning computation. Fixed-structure methods extend reasoning from chains to trees and graphs, tackling complex problems through predefined exploration, decomposition, and aggregation procedures (Wei et al., 2022; Yao et al., 2023; Besta et al., 2024). Dynamic methods further adjust computational effort or recursively expand and update problem structures based on the current reasoning state, increasing the flexibility of the solution process (Han et al., 2025; Pandey et al., 2025; Zhang et al., 2025; Teng et al., 2025). However, for decomposable tasks, insufficient decomposition may compromise solution quality, whereas excessive decomposition increases overhead, and errors in intermediate answers may affect subsequent aggregation. Therefore, deciding whether to decompose further based on the difficulty of each subproblem and effectively correcting intermediate candidate answers are key to balancing solution quality and inference cost.

Through our exploration, we uncover an intrinsic signal: The answer-token entropy naturally produced by an LLM during problem solving reflects its perceived problem difficulty. This signal enables fine-grained routing at the subproblem level: A subproblem is solved directly when the model is sufficiently certain and further decomposed when uncertainty remains high. Linking internal uncertainty to explicit structural decisions enables structurally adaptive reasoning, which requires an adaptable representation. GoT (Besta et al., 2024) shows that graph structures naturally represent nonlinear thought relations such as branching and aggregation, but it neither dynamically adapts the topology to model-perceived difficulty nor supports backtracking. Cognitive science suggests that humans decompose problems when they recognize difficulty and reflect on solutions after solv-

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

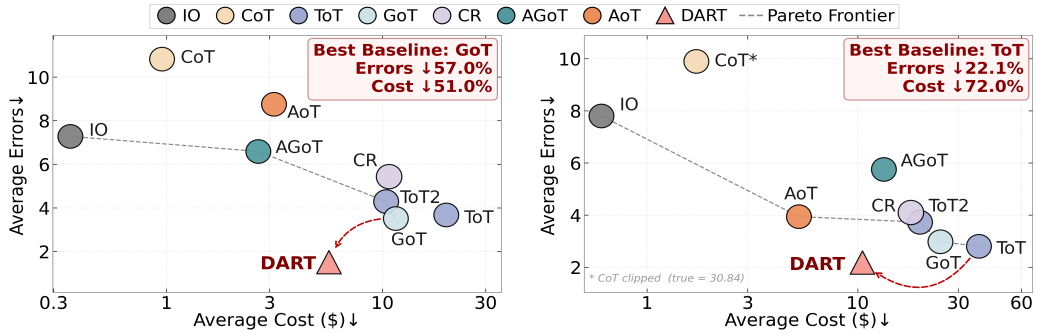


Figure 1: Average cost–performance scatter plots for Qwen3-32B (left) and Llama3.1-70B-FP8 (right) on three tasks: sorting 64 numbers, intersecting two sets of 64 numbers, and keyword counting. Lower values are better for both metrics. DART lies on the Pareto frontier, substantially reducing both error and cost relative to the best-performing baseline on each backbone.

ing them (Sweller, 1988; Correa et al., 2025). Together, these observations motivate a graph-based reasoning framework that can adaptively expand its structure and revise solutions when needed.

Accordingly, we propose DART, a training-free algorithm for difficulty-aware reasoning that combines entropy-guided routing with confidence-aligned self-critique. The router selects direct solving or further decomposition for each subproblem, while self-critique jointly constrains evaluation scores and score-token confidence to improve intermediate-answer assessment and refinement. The three-task averages in Figure 1 show that DART outperforms the best baseline on each backbone in terms of both error and inference cost.

Our contributions are summarized as follows:

- **Difficulty-aware routing.** We uncover that answer-token entropy reflects difficulty perceived by an LLM and use it for subproblem-level routing. Reusing token probabilities from the initial attempt, the router adapts the reasoning structure without training.
- **Confidence-aligned self-critique.** We combine score magnitude with the confidence of the score token to assess solution quality more reliably.
- **Extensive empirical validation.** Extensive experiments validate the advantages of DART, offering a reliable approach for LLMs to handle complex reasoning tasks.

2 Related Work

2.1 Static Reasoning

Static reasoning methods use fixed chain, tree, or graph topologies. Chain-based reasoning was pioneered by CoT (Wei et al., 2022), followed by methods incorporating multi-path voting (Wang et al., 2023), progressive decomposition (Zhou et al., 2023), modular scheduling (Khot et al., 2023), and complexity-based selection (Fu et al., 2023). Tree-based methods, represented by ToT (Yao et al., 2023), explore multiple reasoning paths using BFS or DFS, while FoT (Bi et al., 2025) extends this paradigm to a forest of parallel trees. Graph-based methods, exemplified by GoT (Besta et al., 2024), enable aggregation and refinement across thought nodes. Later work enhances expressiveness through template reuse (Yang et al., 2024), analogical propagation (Yu et al., 2024), and cross-model communication (Yin et al., 2023). Although these methods improve reasoning performance, their predefined topologies cannot adapt to variations in subproblem difficulty.

2.2 Dynamic Reasoning

Dynamic reasoning methods aim to adapt the reasoning process to problem difficulty. At the topology level, AGoT (Pandey et al., 2025) dynamically switches among chain, tree, and graph topologies; CR (Zhang et al., 2025) and AoT (Teng et al., 2025) progressively construct or simplify

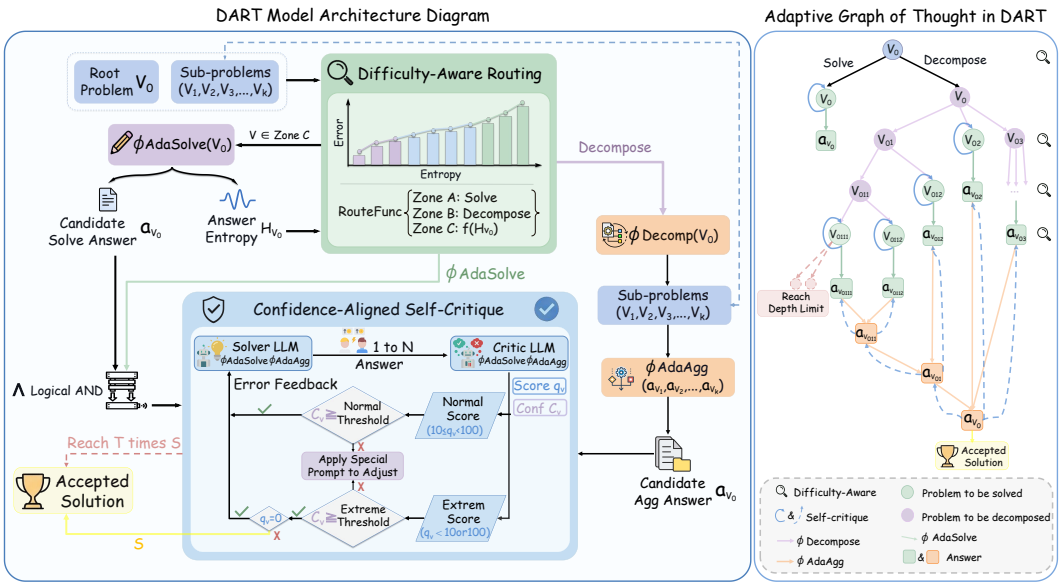


Figure 2: Overview of DART. *Left*: Model architecture. DART routes each subproblem by zone: Zone A solves directly, Zone B directly decomposes it, and Zone C first attempts solving, then uses answer-token entropy to decide whether to retain the candidate answer or further decompose the subproblem. Both solving and aggregation outputs undergo confidence-aligned self-critique. *Right*: The adaptive graph of thoughts constructed by DART during inference. Decomposition depth varies with subproblem difficulty, and loops represent iterative self-reflection triggered by self-critique.

reasoning graphs; XoT (Ding et al., 2024) and RLoT (Hao et al., 2026) select reasoning actions through tree search and reinforcement learning. At the computation level, Token-Budget (Han et al., 2025), BudgetThinker (Wen et al., 2025), CoT-Valve (Ma et al., 2025), and SoT (Aytes et al., 2025) regulate reasoning length through budget constraints or step compression, whereas Confidence Tokens (Chuang et al., 2025) routes requests to different models. These methods improve reasoning flexibility, but adapting the degree of decomposition to subproblem difficulty and reliably correcting intermediate answers remain challenging. DART balances solution quality and inference cost through entropy-guided decomposition decisions and confidence-aligned self-critique.

2.3 LLM Self-Critique

Self-critique methods improve solution quality through iterative evaluation and refinement (Shinn et al., 2023; Madaan et al., 2023; Chen et al., 2025). However, they typically treat evaluation scores as deterministic values, overlooking systematic biases in LLM-based evaluation (Gu et al., 2026; Yang et al., 2025). ConFix (Yuan et al., 2024) and RASC (Wan et al., 2025) incorporate confidence only for factual correction and sample filtering, respectively. In contrast, DART jointly constrains score magnitude and evaluator confidence, directly addressing this limitation.

3 Method

DART constructs an adaptive graph of thoughts whose topology evolves with problem difficulty. It uses answer-token entropy to route each subproblem to either direct solving or further decomposition, thereby adapting the reasoning structure at the subproblem level, and applies confidence-aligned self-critique to improve answer quality. Figure 2 presents the overall framework.

3.1 Thought Graph Definition and Basic Transformation Operations

Formally, DART is defined by the tuple $(\mathcal{G}, \Phi, \mathcal{S}, \mathcal{R}, \mathcal{E})$, where $\mathcal{G}$ is the adaptive thought graph, $\Phi = \{\phi_{\text{Decomp}}, \phi_{\text{AdaSolve}}, \phi_{\text{AdaAgg}}\}$ is the set of transformation operations, $\mathcal{S}$ is the scoring and selection

function, $\mathcal{R}$ is the difficulty-aware router (Section 3.2), and $\mathcal{E}$ is the confidence-aligned self-critique mechanism (Section 3.3). During inference, $\mathcal{R}$ selects the operation for each subproblem, Φ drives the dynamic expansion of $\mathcal{G}$, $\mathcal{E}$ iteratively refines candidates, and $\mathcal{S}$ selects the best among them.

Reasoning Process. We model $\mathcal{G} = (V, E)$ as a directed cyclic graph. V contains problem nodes v, representing subproblems at different granularities, and result nodes a_v, storing their answers. E includes decomposition edges from a parent problem to its subproblems, aggregation edges from subproblem answers to the parent answer, and self-critique feedback edges forming iterative loops. During inference, DART adaptively constructs the topology of $\mathcal{G}$ according to the difficulty of each subproblem being solved, as shown on the right of Figure 2.

Transformation Operations. Φ consists of three transformation operations, each performing a distinct type of structural update on $\mathcal{G}$. We use V^+ and E^+ to denote the sets of nodes and edges added after an operation is applied.

- **Decomposition** ϕ_{Decomp}**.** Problem v is split into k subproblems at a finer granularity, where $V^+ = \{v_1, \ldots, v_k\}$ and $E^+ = \{(v, v_1), \ldots, (v, v_k)\}$.
- **Adaptive Solving** ϕ_{AdaSolve}**.** The solver LLM iteratively solves subproblem v via $\mathcal{E}$, producing h candidate solutions; the self-critique loop is modeled by $E^+ = \{(v, v)\}$. $\mathcal{S}$ then selects the best candidate to form the result node, adding $V^+ = \{a_v\}$ and $E^+ = E^+ \cup \{(v, a_v)\}$. The answer-token entropy computed during the first solution attempt is directly reused as the routing input to $\mathcal{R}$, as described in Section 3.2.
- **Adaptive Aggregation** ϕ_{AdaAgg}**.** The solver LLM aggregates the sub-results $\{a_{v_1}, \ldots, a_{v_k}\}$ into candidate solutions for the parent problem and iteratively refines them using critic feedback via $\mathcal{E}$, producing h candidates. Here $V^+ = \{a_v\}$ and $E^+ = \{(a_{v_i}, a_v), (a_v, a_{v_i}) \mid i = 1, \ldots, k\}$, where the forward and feedback edges form the self-critique loop, and $\mathcal{S}$ selects the best candidate as the final content of a_v.

Scoring and Selection. $\mathcal{S}(\mathcal{G}, v, h)$ selects the best of h candidates at node v as result node a_v, and is incorporated into the pipeline of $\mathcal{E}$. Following GoT and ToT, $\mathcal{S}$ can be implemented as either an LLM judge or a local scoring function.

3.2 Difficulty-Aware Routing

To motivate difficulty-aware routing, we examine the relationship between answer-token entropy and solution error on set intersection, sorting, and keyword counting.

Specifically, for a subproblem v, let $(y_1, \ldots, y_L)$ be the answer sequence generated when the model solves it directly. At generation position i, the Shannon entropy of the model's output distribution is

$$H_i(v) = -\sum_{u \in \mathcal{V}} p_\theta(u \mid x_v, y_{<i}) \log p_\theta(u \mid x_v, y_{<i}), \tag{1}$$

where $\mathcal{V}$ is the model vocabulary, θ denotes the model parameters, x_v is the subproblem input, and $p_\theta(u \mid x_v, y_{<i})$ is the conditional probability assigned to token u given the subproblem input and generation prefix. We then take the arithmetic mean of the position-level token entropies over all generated positions to obtain the answer-token entropy:

$$\bar{H}(v) = \frac{1}{L} \sum_{i=1}^{L} H_i(v). \tag{2}$$

At each granularity, we filter 1,000 probes and sort the retained samples by answer-token entropy into 8 equal-frequency bins, computing each bin's mean entropy and error. Figure 3 shows Llama3.1-70B-FP8 on set intersection: the first set contains 128 elements, while the second decreases from 128 to 2 across GIDs 0–6, which represent successive decomposition granularities along the same reasoning chain.

The entropy–error relationship varies with granularity. At the harder GIDs 0–1, errors are high and correlations are negative; GIDs 2–5 exhibit strong positive correlations. However, correlation alone does not determine the need for entropy gating: as problems become easier, low absolute

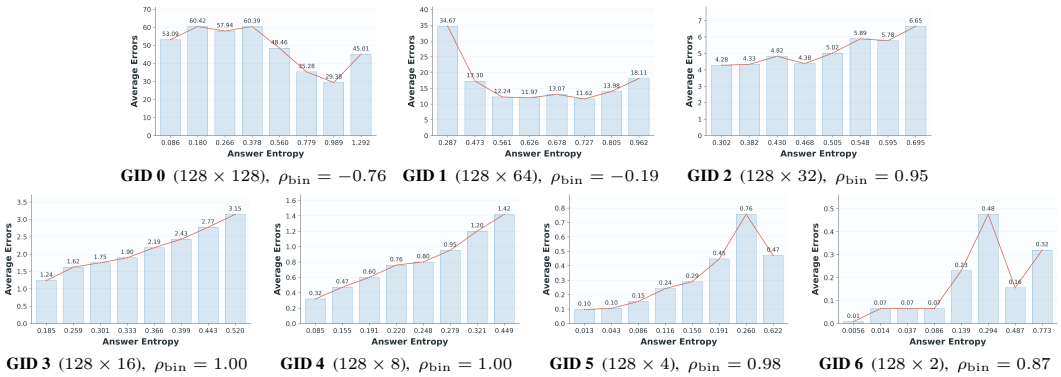


Figure 3: Answer-token entropy and average error for Llama3.1-70B-FP8 on set intersection. Each panel gives the GID, input-set sizes $a \times b$, and Spearman correlation ρ_{bin} between the eight bin-average entropies and errors after filtering 1,000 probes per GID. The first set has 128 elements; the second decreases from 128 to 2.

error limits the value of further decomposition. DART therefore combines mean error with entropy–error correlation to identify routing regions. Complete results across tasks and models, including granularities assigned to fixed solving and decomposition paths, appear in Appendix C.4.

One-time region calibration. For each model–granularity pair (m, d), single-attempt direct-solving probes yield the mean error $\mu_{m,d}$ and binned Spearman correlation $\rho_{m,d}$ between entropy and error. Within each task t, a Gaussian mixture model (GMM) jointly clusters these pairs to assign regions:

$$\begin{aligned} Z_{m,d} &= \mathcal{M}_{K_t^\star}\left(\mathrm{GMM}_{K_t^\star}\left[\mathrm{Std}_t\big(\log(1+\mu_{m,d})\big),\ \tfrac{1}{2}\rho_{m,d}\right]\right), \\ K_t^\star &= \arg\min_{K\in\{1,2,3\}} \mathrm{BIC}_t(K). \end{aligned} \tag{3}$$

Here, Std_t denotes within-task standardization. The Bayesian information criterion (BIC) selects the component count by balancing fit against model complexity; lower values are preferred. Mapping $\mathcal{M}_K$ orders components by mean error: three components map to A, C, B in ascending order, and two map to A, B. The resulting $Z_{m,d}$ is the router's fixed region label. Appendix B.1 details the procedure, single-component case, and experimental results.

Difficulty routing function. After each split, the task-specific granularity level d increases by one for child subproblems. For the current model m, DART uses the calibrated label $Z_{m,d}$ to select an operation for each subproblem, reusing the offline calibration throughout graph construction:

$$R(v,d) = \begin{cases} \text{Solve}, & Z_{m,d} = \text{A}, \\ \text{Decompose}, & Z_{m,d} = \text{B}, \\ f_{\text{H}}\big(\bar{H}(v)\big), & Z_{m,d} = \text{C}. \end{cases} \tag{4}$$

Zones A and B invoke direct solving and decomposition, respectively; Zone C uses the instance-level entropy decision below to retain the initial answer or trigger further subproblem decomposition.

Entropy decision in Zone C. In Zone C, DART computes $\bar{H}(v)$ from an initial direct-solving attempt and decides whether to retain its answer or decompose:

$$f_{\text{H}}\big(\bar{H}(v)\big) = \begin{cases} \text{Solve}, & \bar{H}(v) < \tau_d, \\ \text{Decompose}, & \bar{H}(v) \geq \tau_d. \end{cases} \tag{5}$$

Here, τ_d is the mean answer-token entropy after filtering 1,000 probes at granularity d; Appendix C.6 studies threshold sensitivity. Below τ_d, DART reuses the initial answer; otherwise, it decomposes. Calibrated regions and thresholds are reused across requests.

3.3 Confidence-Aligned Self-Critique

To further improve answer accuracy during adaptive solving (ϕ_{AdaSolve}) and adaptive aggregation (ϕ_{AdaAgg}), we design a confidence-aligned self-critique mechanism for DART.

Algorithm 1 DART Algorithm Process

Require: Root problem v_0, difficulty router R with thresholds $\{\tau_d\}$
Require: Atomic granularity $d_{\min}$, maximum self-critique rounds T
Ensure: Final solution a^*

```
function RecSolve(v)
    Determine d and read the current model's region label Z_{m,d} from R
    if v has reached d_min or Z_{m,d} = A then
        return φ_AdaSolve(v; T)                          ▷ Atomic or direct solve with self-critique
    else if Z_{m,d} = C then
        (ã_v, p_v) ← SolveOnce(v)                        ▷ Probe answer and token probabilities
        H̄(v) ← AnswerEntropy(p_v)
        if f_H(H̄(v); τ_d) = Solve then
            return φ_AdaSolve(v; a^(0) = ã_v, T)         ▷ Reuse probe answer
        end if
    end if
    {v_1, ..., v_k} ← φ_Decomp(v)                        ▷ Zone B or high-entropy Zone C
    a_{v_i} ← RecSolve(v_i), i = 1, ..., k
    return φ_AdaAgg(v, {a_{v_1}, ..., a_{v_k}}; T)       ▷ Aggregate with self-critique
end function
return RecSolve(v_0)
```

Confidence Alignment Principle. As shown in the blue region at the bottom of Figure 2, at each iteration the critic LLM outputs a quality score $q_v \in [0, 100]$ and a confidence c_v, defined as the probability assigned to the scoring token. Extreme scores (near perfect or very low) require higher confidence thresholds, while intermediate scores allow lower ones. Early exit requires $q_v > 99$ and $c_v \geq \tau_{\text{ext}}$. If the score format or confidence is inadequate, the critic reevaluates the same candidate; if the candidate is still not accepted, the solver refines it using the final feedback. Once the acceptance condition is met or the iteration limit $T_{\max}$ is reached, candidate generation stops, and $\mathcal{S}$ selects the final answer from the retained candidates. The complete procedure is provided in Appendix B.2.

Proximal Feedback. The solving LLM receives only the current candidate and feedback from the preceding iteration, rather than the full history information, to avoid redundant or contradictory signals that amplify evaluation bias. Section 4.3 further validates the effectiveness of this design.

Overall, the difficulty-aware routing and confidence-aligned self-critique mechanisms work synergistically to improve overall reasoning performance.

3.4 Adaptive Graph-of-Thought Construction

Algorithm 1 integrates these modules. Starting from v_0, DART recursively solves or decomposes each subproblem, stopping decomposition when direct solving is selected or atomic granularity $d_{\min}$ is reached. The graph adapts vertically through difficulty-aware decomposition depth and horizontally through local self-critique loops governed by confidence alignment.

4 Experiments

We evaluate DART through four research questions:

- ***RQ1*** (Section 4.2): Does DART improve accuracy and efficiency over baselines across tasks, models, and scales?
- ***RQ2*** (Section 4.3): How do routing and self-critique contribute individually and jointly, and how do confidence alignment and feedback strategy affect performance?
- ***RQ3*** (Section 4.4): Can entropy guidance improve random routing's error–cost trade-off?
- ***RQ4*** (Section 4.5): How sensitive is DART to the number of self-critique iterations?

4.1 Experimental Setup

Tasks and datasets. Following GoT (Besta et al., 2024), we evaluate three canonical benchmarks for decomposable reasoning: (1) set intersection, finding common elements in two integer sets; (2)

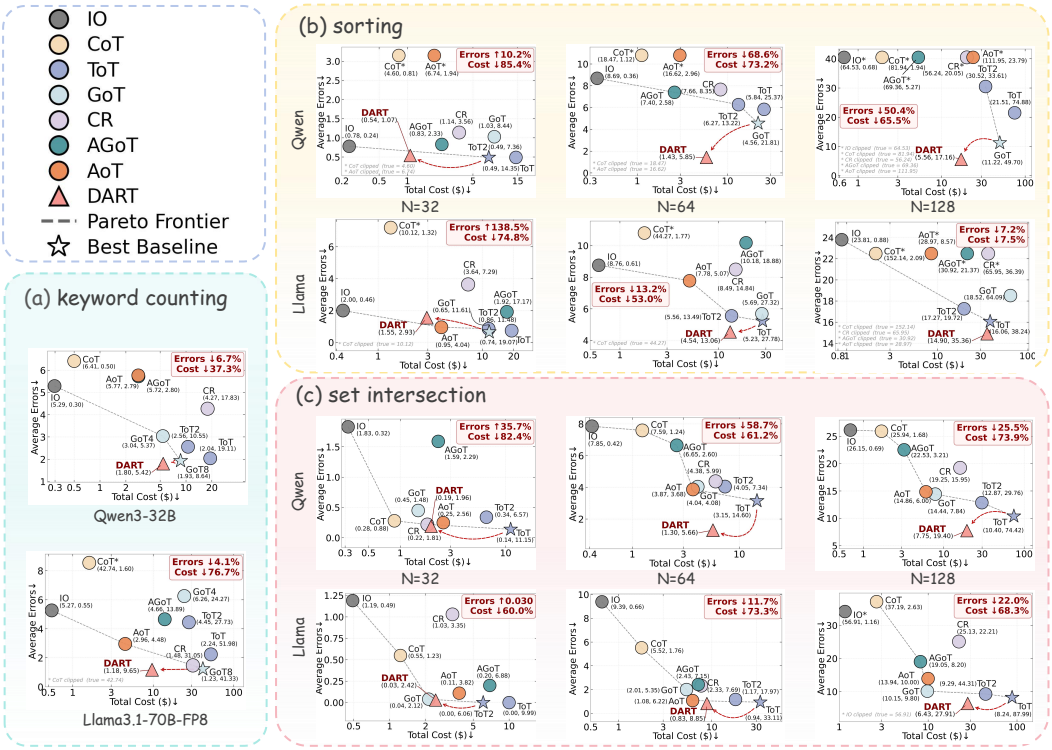


Figure 4: Main results and cost–performance comparison of DART across tasks. The vertical axis shows mean errors per instance over 100 samples (see Appendix C.1.1), and the horizontal axis shows total inference cost in US dollars. Point annotations give average error and total cost.

sorting, arranging a numerical list in ascending order; and (3) keyword counting, counting specified keywords in text with varying sentence-level keyword density. For set intersection and sorting, $N \in \{32, 64, 128\}$ denotes the size of each input set and the list length, respectively. These tasks capture fundamental matching, ordering, and counting operations underlying applications such as document comparison, pattern matching, and text analysis. These benchmarks challenge LLMs to maintain accuracy as input size grows and subproblem difficulty varies, testing whether additional inference effort can improve reasoning quality without unnecessary computational cost. Each task-scale-model combination is evaluated on 100 randomly sampled instances.

Baselines. Static baselines include IO for direct answers, CoT (Wei et al., 2022) for step-by-step reasoning, ToT (Yao et al., 2023) and ToT2 for wide-branch shallow and narrow-branch deep search, respectively, and GoT (Besta et al., 2024) for graph-based reasoning, with two GoT4/GoT8 granularity variants for the keyword counting task. Dynamic baselines include CR (Zhang et al., 2025) for cumulative verification of intermediate reasoning steps, AGoT (Pandey et al., 2025) for adaptive graph expansion, and AoT (Teng et al., 2025) for progressive problem simplification.

Implementation details. We use Qwen3-32B (Qwen) and Llama3.1-70B-FP8 (Llama) as solvers and critics across all evaluated tasks. Baseline implementations, configurations, task adaptations, hardware, and sampling temperatures are given in Appendix C.3. Following GoT (Besta et al., 2024), we report inference cost in US dollars at fixed token rates across methods using the same backbone; token rates and detailed evaluation metrics are provided in Appendix C.1.

4.2 Main Results (*RQ1*)

Figure 4(a)–(c) show keyword counting, sorting, and set-intersection results (tabulated in Appendix C.5). DART achieves the lowest average error in 10 out of 14 task–size–model configurations, with lower inference cost than the best-performing baseline in each.

Table 1: Routing and self-critique ablation. Header ↓ means lower is better; percentage ↑/↓ indicates increases/decreases from full DART within each task–model setting.

Variant	Routing	Self-Critique	Keyword Counting				Set Intersection			
			Qwen		Llama		Qwen		Llama	
			Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓
DART	✓	✓	**1.80**	5.42	**1.18**	9.65	**1.30**	5.66	**0.83**	8.85
Decomp. once	×	✓	2.89↑60.6%	3.73↓31.2%	2.11↑78.8%	8.54↓11.5%	2.54↑95.4%	4.45↓21.4%	1.67↑101.2%	7.57↓14.5%
Decomp. twice	×	✓	2.02↑12.2%	5.54↑2.2%	1.78↑50.8%	10.23↑6.0%	1.97↑51.5%	6.32↑11.7%	1.15↑38.6%	9.19↑3.8%
DART	✓	×	2.55↑41.7%	4.31↓20.5%	1.96↑66.1%	8.82↓8.6%	2.37↑82.3%	4.93↓12.9%	1.52↑83.1%	7.89↓10.8%
Decomp. once	×	×	3.48↑93.3%	**2.08**↓61.6%	2.73↑131.4%	**8.19**↓15.1%	2.94↑126.2%	**4.34**↓23.3%	2.39↑188.0%	**7.42**↓16.2%
Decomp. twice	×	×	2.44↑35.6%	4.03↓25.6%	2.01↑70.3%	8.89↓7.9%	2.10↑61.5%	5.86↑3.5%	1.63↑96.4%	7.94↓10.3%

Table 2: Confidence alignment ablation.

Task	Conf. Align.	Qwen		Llama	
		Error↓	Cost↓	Error↓	Cost↓
Keyword Counting	✓	**1.80**	**5.42**	**1.18**	9.65
	×	1.97	5.49	1.43	**9.60**
Set Intersection	✓	**1.30**	**5.66**	**0.83**	**8.85**
	×	1.66	5.78	1.04	8.98
Sorting	✓	**1.43**	**5.85**	**4.54**	**13.06**
	×	1.68	5.97	4.79	13.14

Table 3: Feedback-strategy ablation.

Task	Model	Proximal		Historical	
		Error↓	Cost↓	Error↓	Cost↓
Keyword Counting	Qwen	**1.80**	**5.42**	1.97	6.05
	Llama	**1.18**	**9.65**	1.21	9.89
Set Intersection	Qwen	**1.30**	**5.66**	1.34	6.09
	Llama	**0.83**	**8.85**	0.89	9.27
Sorting	Qwen	**1.43**	**5.85**	1.44	6.41
	Llama	**4.54**	**13.06**	4.65	14.70

DART versus GoT and ToT. Across both backbones, DART achieves lower error than GoT and ToT on sorting and set intersection at $N = 64/128$, and lower cost than ToT in all 14 configurations. DART costs less than GoT on sorting and achieves lower error and cost than GoT8 on keyword counting. ToT achieves minimum error in three $N = 32$ configurations, but incurs 4.1–13.4 times DART's cost. These results show that DART combines adaptive reasoning structures with self-critique to achieve a favorable error–cost trade-off relative to fixed-topology baselines.

DART versus CR, AGoT, and AoT. DART achieves lower average error than CR and AGoT in all 14 configurations and than AoT in 13, with lower inference cost than CR in 10 out of these configurations. Across the four $N = 128$ configurations for sorting and set intersection, these dynamic baselines produce 1.9–20.1 times as many errors as DART. These results show that DART achieves lower error on the evaluated decomposable tasks across the two backbone models.

DART versus IO and CoT. Although IO and CoT have the lowest computational costs, their accuracy is substantially lower than that of DART.

Scaling behavior. Although DART does not attain the lowest error at $N = 32$, it achieves the lowest error in all eight sorting and set-intersection configurations at $N = 64/128$. This suggests that larger problems may benefit more from decomposition and self-critique, which improve solution quality by reducing subproblem complexity and correcting intermediate errors.

4.3 Ablation Study (*RQ2*)

Ablations use both backbones (sorting/set intersection: $N = 64$). Routing/self-critique and random routing (Section 4.4) cover keyword counting and set intersection. Sorting's one-time BIC calibration selects $K = 2$ (Zones A/B): fixed solve/decompose routes without instance-level entropy gating, not a task-specific exception. Confidence-alignment/feedback ablations cover all tasks.

Routing and self-critique ablation. Table 1 compares six configurations to assess difficulty-aware routing, self-critique, and their complementarity. Decomp. once/twice uses one/two fixed decomposition levels, splitting the input text or the set being decomposed into two/four parts. All percentage changes are relative to full DART: (1) *Routing enables a better error–cost trade-off.* With self-critique retained, fixed one-level decomposition reduces cost by 11.5%–31.2% but increases error by 60.6%–101.2%; fixed two-level decomposition increases both error (12.2%–51.5%) and cost (2.2%–11.7%) in all four settings. (2) *Self-critique improves the reliability of routed solutions.* Removing self-critique while retaining routing reduces cost by 8.6%–20.5% but increases error by 41.7%–83.1%. (3) *Self-critique complements routing.* Removing both mechanisms increases error in every setting. Routing alone offers inconsistent benefits, but with self-critique it lowers both error

Table 4: Entropy-guided versus random routing (p: Zone C decomposition probability; set intersection: $N = 64$). Percentages show changes from full DART (↑: increase; ↓: decrease).

Variant	Keyword Counting				Set Intersection			
	Qwen		Llama		Qwen		Llama	
	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓
DART	1.80	5.42	1.18	9.65	1.30	5.66	0.83	8.85
Random ($p = 0.25$)	2.55↑41.7%	5.21↓3.9%	1.81↑53.4%	9.54↓1.1%	1.61↑23.8%	5.35↓5.5%	0.97↑16.9%	8.81↓0.5%
Random ($p = 0.50$)	2.32↑28.9%	5.61↑3.5%	1.48↑25.4%	10.27↑6.4%	1.69↑30.0%	5.96↑5.3%	0.92↑10.8%	9.24↑4.4%
Random ($p = 0.75$)	1.88↑4.4%	7.00↑29.2%	0.94↓20.3%	12.71↑31.7%	1.31↑0.8%	6.82↑20.5%	0.89↑7.2%	9.94↑12.3%

and cost versus fixed two-level decomposition with self-critique in all four settings. Full DART's lowest error in all four settings supports self-critique's complementary role.

Confidence alignment ablation. Table 2 compares full DART with critic scoring without confidence alignment. Removing alignment increases error across both backbones and all three tasks, with little cost change, supporting score-token confidence for reliable self-critique.

Feedback-strategy ablation. Table 3 compares proximal feedback, which uses only the latest critique, with historical feedback, which retains the accumulated history. Proximal feedback achieves lower error and cost across both backbones and all three tasks, supporting the use of recent feedback for effective and efficient solution refinement.

4.4 Entropy-Guided versus Random Routing (*RQ3*)

We compare DART with Bernoulli random routing ($p \in \{0.25, 0.50, 0.75\}$) on keyword counting and set intersection ($N = 64$) across both backbones. Only Zone C decisions change: each subproblem independently decomposes with probability p or retains its initial answer otherwise. All other settings match DART. Sorting has no Zone C and is excluded.

Table 4 shows: (1) $p = 0.25$ cuts cost 0.5%–5.5% but raises error 16.9%–53.4%; (2) $p = 0.50$ worsens both in all four settings; (3) $p = 0.75$ lowers only Llama keyword-counting error (20.3%) at 31.7% higher cost, worsening both elsewhere. Overall, no tested random rate improves both metrics over DART. Appendix C.7 finds larger error reduction per dollar for high- than low-entropy splits; equal-count random routing also has higher expected error and cost than entropy routing. Together, these results support entropy routing's error–cost benefit.

4.5 Hyperparameter Sensitivity (*RQ4*)

Figure 5 shows that increasing T from 1 to 3 reduces error from 2.73 to 1.80, while cost rises from 4.56 to 5.42. The $T = 3$ point uses the main-result run under identical settings. Most improvement occurs from $T = 1$ to $T = 2$, reducing error by 0.78 for 0.67 additional cost; $T = 3$ adds a 0.15 reduction for 0.19. For $T > 3$, error fluctuates between 1.85 and 1.92, while cost reaches 5.55, with no further error reduction. We choose $T = 3$ for its lowest observed error at moderate cost.

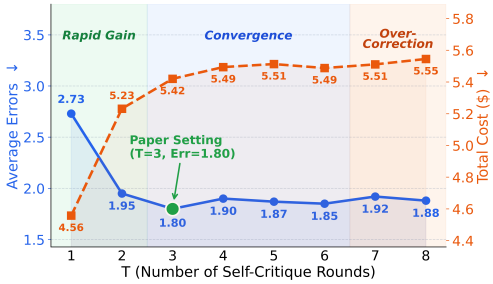


Figure 5: Effect of self-critique iterations T on Qwen3-32B for keyword counting.

5 Conclusion

We propose DART, a training-free algorithm that uses answer-token entropy to guide the choice between direct solving and further decomposition. Confidence-aligned self-critique complements routing by jointly considering evaluation scores and score-token confidence to assess and refine intermediate answers. Experiments across three tasks demonstrate favorable error–cost trade-offs, while ablations support the contributions and complementarity of the two mechanisms. Future work will explore more efficient reasoning mechanisms for a better quality–cost trade-off.

REFERENCES

Simon A. Aytes, Jinheon Baek, and Sung Ju Hwang. Sketch-of-thought: Efficient LLM reasoning with adaptive cognitive-inspired sketching. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 24296–24320, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1236. URL https://aclanthology.org/2025.emnlp-main.1236/.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and et al. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, Mar. 2024. doi: 10.1609/aaai.v38i16.29720. URL https://ojs.aaai.org/index.php/AAAI/article/view/29720.

Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing LLM reasoning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=BMJ3pyYxu2.

Justin Chen, Archiki Prasad, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. MAgICoRe: Multi-agent, iterative, coarse-to-fine refinement for reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 32663–32686, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1660. URL https://aclanthology.org/2025.emnlp-main.1660/.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. Towards reasoning era: a survey of long chain-of-thought for reasoning large language models. *SCIENCE CHINA Information Sciences*, 69(6):161101–, 2026. URL http://www.sciengine.com/publisher/ScienceChinaPress/journal/SCIENCECHINAInformationSciences/69/6/10.1007/s11432-025-4665-8.

Yu-Neng Chuang, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki, Xia Hu, and Helen Zhou. Learning to route LLMs with confidence tokens. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=U08mUogGDM.

Carlos G. Correa, Sophia Sanborn, Mark K. Ho, Frederick Callaway, Nathaniel D. Daw, and Thomas L. Griffiths. Exploring the hierarchical structure of human plans via program generation. *Cognition*, 255:105990, 2025. ISSN 0010-0277. doi: https://doi.org/10.1016/j.cognition.2024.105990. URL https://www.sciencedirect.com/science/article/pii/S0010027724002762.

Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Everything of thoughts: Defying the law of penrose triangle for thought generation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 1638–1662, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.95. URL https://aclanthology.org/2024.findings-acl.95/.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=yf1icZHC-l9.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Zhouchi Lin, Bowen Zhang, Lionel Ni, Wen Gao, Yuanzhuo Wang, and Jian Guo. A survey on llm-as-a-judge. *The Innovation*, pp. 101253, 2026. ISSN 2666-6758. doi: https://doi.org/10.1016/j.xinn.2025.101253. URL https://www.sciencedirect.com/science/article/pii/S2666675825004564.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: https://doi.org/10.1038/s41586-025-09422-z. URL https://www.nature.com/articles/s41586-025-09422-z.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1274. URL https://aclanthology.org/2025.findings-acl.1274/.

Qianyue Hao, Sibo Li, Jian Yuan, and Yong Li. RL of thoughts: Navigating LLM reasoning with inference-time reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Dw034qKrP5.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=_nGgzQjzaRy.

Zihan Liu, Zhuolin Yang, Yang Chen, Chankyu Lee, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron 1.1: Advancing math and code reasoning through SFT and RL synergy. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IaEqjWXd1d.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. CoT-valve: Length-compressible chain-of-thought tuning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6025–6035, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.300. URL https://aclanthology.org/2025.acl-long.300/.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Tushar Pandey, Ara Ghukasyan, Oktay Goktas, and Santosh Kumar Radha. Adaptive graph of thoughts: Test-time adaptive reasoning unifying chain, tree, and graph structures, 2025. URL https://arxiv.org/abs/2502.05078.

Libo Qin, Qiguang Chen, Yuhang Zhou, Zhi Chen, Yinghui Li, Lizi Liao, Min Li, Wanxiang Che, and Philip S. Yu. A survey of multilingual large language models. *Patterns*, 6(1):101118, 2025. ISSN 2666-3899. doi: https://doi.org/10.1016/j.patter.2024.101118. URL https://www.sciencedirect.com/science/article/pii/S2666389924002903.

Xiangqing Shen, Fanfan Wang, Zinong Yang, Bing Wang, Wenli Du, Chengqing Zong, and Rui Xia. Reason-align-respond: Aligning llm reasoning with knowledge graphs for kgqa. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–14, 2026. doi: 10.1109/TPAMI.2026.3665645.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

John Sweller. Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2):257–285, 1988. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(88)90023-7. URL https://www.sciencedirect.com/science/article/pii/0364021388900237.

Fengwei Teng, Quan Shi, Zhaoyang Yu, Jiayi Zhang, Yuyu Luo, Chenglin Wu, and Zhijiang Guo. Atom of thoughts for markov llm test-time scaling. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 74010–74040. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/6b1018f844177088c380e1143d55cdfb-Paper-Conference.pdf.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. Reasoning aware self-consistency: Leveraging reasoning paths for efficient LLM sampling. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3613–3635, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.184. URL https://aclanthology.org/2025.naacl-long.184/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Hao Wen, Xinrui Wu, Yi Sun, Feifei Zhang, Liye Chen, Jie Wang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. Budgetthinker: Empowering budget-aware llm reasoning with control tokens, 2025. URL https://arxiv.org/abs/2508.17196.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 113519–113544. Curran Associates, Inc., 2024. doi: 10.52202/079017-3607. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/cde328b7bf6358f5ebb91fe9c539745e-Paper-Conference.pdf.

Zhe Yang, Yichang Zhang, Yudong Wang, Ziyao Xu, Junyang Lin, and Zhifang Sui. Confidence v.s. critique: A decomposition of self-correction capability for LLMs. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3998–4014, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.203. URL https://aclanthology.org/2025.acl-long.203/.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuanjing Huang, and Xipeng Qiu. Exchange-of-thought: Enhancing large language model capabilities through cross-model communication. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023*

Conference on Empirical Methods in Natural Language Processing, pp. 15135–15153, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.936. URL https://aclanthology.org/2023.emnlp-main.936/.

Junchi Yu, Ran He, and Zhitao Ying. THOUGHT PROPAGATION: AN ANALOGICAL APPROACH TO COMPLEX REASONING WITH LARGE LANGUAGE MODELS. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=SBoRhRCzM3.

Yige Yuan, Bingbing Xu, Hexiang Tan, Fei Sun, Teng Xiao, Wei Li, Huawei Shen, and Xueqi Cheng. Fact-level confidence calibration and self-correction, 2024. URL https://arxiv.org/abs/2411.13343.

Yifan Zhang, Jingqin Yang, Yang Yuan, and Andrew C Yao. Cumulative reasoning with large language models. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=grW15p4eq2.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WZH7099tgfM.

Mingye Zhu, Yi Liu, Zheren Fu, Quan Wang, and Yongdong Zhang. In-token rationality optimization: Towards accurate and concise llm reasoning via self-feedback. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(41):35195–35203, Mar. 2026. doi: 10.1609/aaai.v40i41.40826. URL https://ojs.aaai.org/index.php/AAAI/article/view/40826.

A STATEMENT

A.1 AI USE STATEMENT

We used an AI assistant to improve the clarity of the writing and assist with code implementation and testing. The AI-assisted writing and code were reviewed by the human authors, who take responsibility for the final manuscript, code, and results.

A.2 REPRODUCIBILITY STATEMENT

Calibration, evaluation settings, and prompt templates are detailed in Appendices B–D. The supplementary package includes the keyword-counting implementation, benchmark data, dependencies, and execution instructions.

A.3 ETHICS STATEMENT

This work does not involve human-subject experiments. LLM outputs may still contain errors or biases; deployment in high-stakes settings requires independent validation.

B SUPPLEMENTARY METHOD

B.1 ONE-TIME REGION CALIBRATION WITH GAUSSIAN MIXTURE MODELS

This section specifies the region-calibration procedure in Eq. 3 and reports its results using 1,000 direct-solving probes per model–task–GID combination. A single attempt is made for each probe; the resulting answers and entropies are aggregated to assign one region label to each model–GID pair. Thus, calibration requires reference answers to measure error, whereas subsequent routing reuses the calibrated labels without reference answers.

Probe statistics and filtering. For a probe, let h be its mean answer-token entropy, e its task-specific error count, and b its structural reference size. We retain probes with valid output format, finite h and e, $0 \le e < 1000$, $b > 0$, and $e \le 1.25b$. Here, b is the smaller input-set size for set intersection, the list length for sorting, and the total number of country-name mentions for keyword counting. The $b > 0$ condition excludes legitimate zero-mention keyword samples. The 1,000 probes are counted before filtering, and retained sample counts can differ across GIDs.

For each model–GID pair, $\mu_{m,d}$ is the mean retained error. We sort probes by entropy into eight approximately equal-frequency bins and compute $\rho_{m,d}$ as the Spearman correlation between the bins' mean entropies and mean raw errors. If either sequence is constant, we set $\rho_{m,d} = 0$.

Task-specific feature space. Within each task t, we pool both backbones and represent every model–GID pair by $\mathbf{z}_{m,d} = (x_{m,d}, y_{m,d})$, where

$$x_{m,d} = \frac{\log(1 + \mu_{m,d}) - a_t}{s_t}, \qquad y_{m,d} = \frac{\rho_{m,d}}{2}. \tag{6}$$

Here, a_t and s_t are the mean and population standard deviation of $\log(1 + \mu)$ across that task's model–GID pairs. Each pair has equal weight, irrespective of its retained probe count. The three tasks are fitted separately, with $N_t = 36$, 14, and 8 points for set intersection, sorting, and keyword counting, respectively. The logarithm compresses the error scale; the fixed factor $1/2$ controls the relative contribution of correlation without amplifying small differences when correlations are nearly constant.

GMM fitting and selection. For each $K \in \{1, 2, 3\}$, we fit a mixture with component weights π_k, centers $\mathbf{c}_k$, and spherical covariances $\sigma_k^2 I_2$:

$$p_K(\mathbf{z}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\mathbf{z}; \mathbf{c}_k, \sigma_k^2 I_2), \qquad \sum_{k=1}^{K} \pi_k = 1. \tag{7}$$

Each component has its own variance, shared across both feature directions. We initialize from every partition into K nonempty contiguous segments of points sorted by (μ, ρ), then fit by expectation–maximization (EM). Covariance regularization is $\epsilon_t = \max\{10^{-6}, \min[\mathrm{Var}(x), \mathrm{Var}(y)]\}$; EM uses

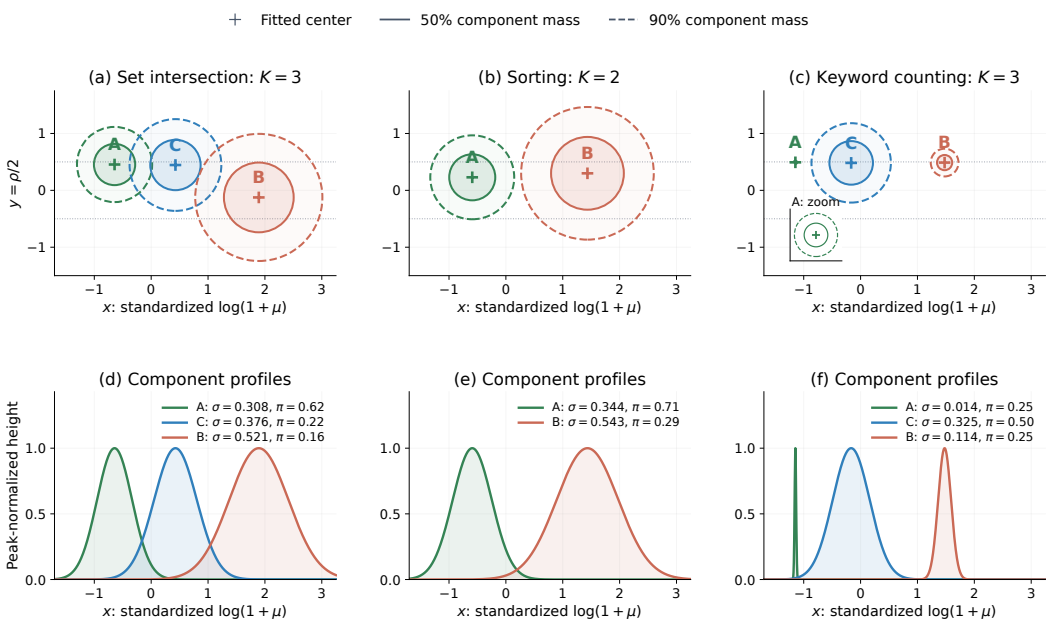


Figure 6: Gaussian components learned from 1,000 probes per model–GID pair using eight entropy bins, jointly fitting both backbones within each task. **Top:** centers (crosses) and contours enclosing 50% (solid) and 90% (dashed) of component probability mass. Colors denote A (green), C (blue), and B (orange); the inset enlarges the keyword-counting A component. Dotted lines mark $y \in [-0.5, 0.5]$. **Bottom:** marginal profiles along x, normalized to unit peak height; legends report standard deviation σ and mixture weight π.

at most 500 iterations, tolerance 10^{-5}, and seed 0. Assignments use maximum posterior probability. For each K, we retain the highest-likelihood converged fit with at least two assigned points per component.

The Bayesian information criterion (BIC) chooses the number of components:

$$\mathrm{BIC}_t(K) = -2\ell_{t,K} + (4K - 1)\log N_t, \qquad K_t^\star = \arg\min_{K \in \{1,2,3\}} \mathrm{BIC}_t(K), \tag{8}$$

where $\ell_{t,K}$ is the total fitted log-likelihood and $4K - 1$ is the number of free parameters. N_t counts model–GID points, not individual probes. Lower BIC balances fit against complexity; ties favor smaller K. The candidate set allows no partition and at most three routing regions.

Mapping components to routing regions. We order components by the average untransformed $\mu_{m,d}$ of their hard-assigned points. For $K = 3$, the ascending order maps to A, C, B; for $K = 2$, it maps to A, B. These labels select direct solving, entropy gating, and direct decomposition as defined in Eq. 4. A single component indicates no partition and has no defined routing action; no evaluated task selects it. Calibration supplies fixed labels $Z_{m,d}$, while the instance-level threshold τ_d is determined separately.

Rationale for the routing policy. Mean error measures direct-solving reliability, while correlation measures entropy's ability to distinguish higher-error groups. In Zone A, low error limits the potential improvement: eliminating all errors on the retained probes would reduce their mean by at most $\mu_{m,d}$, motivating direct solving without additional calls. In Zone C, intermediate error and positive entropy–error association motivate selective decomposition of less reliable answers. In Zone B, high error motivates decomposition under the premise that smaller subproblems are easier. The partition uses both features; action labels follow error order, with no explicit correlation cutoff. This is a probe-informed policy, not a direct estimate of decomposition benefit; its value is assessed through end-to-end error and cost.

Calibration results and interpretation. Table 5 reports the BIC values, and Table 6 lists the resulting region assignments for every GID and both backbones. The selected component counts are

three for set intersection, two for sorting, and three for keyword counting. Across 58 model–GID pairs, the assignments comprise 35 A, 11 C, and 12 B labels.

Table 5: BIC model selection using 1,000 probes per model–GID pair before filtering. Bold values denote the lowest BIC within each task; $K_t^\star$ is the selected component count.

Task	N_t	BIC(1)	BIC(2)	BIC(3)	$K_t^\star$
Set intersection	36	169.813	116.045	**110.648**	3
Sorting	14	69.850	**49.587**	54.925	2
Keyword counting	8	40.556	20.428	**12.759**	3

Figure 6 visualizes the selected mixtures. Set intersection has components separated by error level and, for the hardest component, lower correlation. Sorting is represented by two components and therefore has no C region. Keyword counting has similarly high correlations across its components, so their separation mainly follows the error feature; its two GID 3 points form a particularly narrow A component. Thus, positive correlation alone does not imply a C label. For example, Llama set-intersection GID 3 has $(\mu, \rho) = (2.129, 1.000)$ and maps to C, whereas GID 4 has $(0.814, 1.000)$ and maps to A despite its strong positive correlation.

Appendix C.8 evaluates the stability of this calibration under bootstrap resampling of the probes.

B.2 Confidence-Aligned Self-Critique Algorithm

This section provides the complete procedure for the confidence-aligned self-critique mechanism described in Section 3.3 of the main paper.

Algorithm 2 Confidence-Aligned Self-Critique

Require: Initial candidate answer $a^{(0)}$, maximum number of iterations $T_{\max}$, confidence threshold for extreme scores τ_{ext}, confidence threshold for normal scores τ_{norm}
Ensure: Final answer a^*

for $t = 0$ to $T_{\max} - 1$ **do**
  Evaluate $a^{(t)}$ with the critic, obtaining $q^{(t)}$, $c^{(t)}$, and feedback
  Set $\tau \leftarrow \tau_{\text{norm}}$; if $q^{(t)} < 10$ or $q^{(t)} = 100$, set $\tau \leftarrow \tau_{\text{ext}}$
  if the score format is invalid **or** $c^{(t)} < \tau$ **then**
    Reevaluate the same $a^{(t)}$ once, replacing $q^{(t)}$, $c^{(t)}$, and feedback
  end if
  Save $a^{(t)}$ and its final evaluation
  if $q^{(t)} > 99$ **and** $c^{(t)} \geq \tau_{\text{ext}}$ **then**
    break ▷ Stop generating candidates
  end if
  if $t < T_{\max} - 1$ **then**
    The solver generates $a^{(t+1)}$ from $a^{(t)}$ and the final critique feedback
  end if
end for
return the best answer selected by $\mathcal{S}$ from all saved candidates

C Supplementary Experiments

C.1 Evaluation Metric Definitions

C.1.1 Average Error

Set intersection. Let A and B denote the ground-truth intersection and the set output by the model, respectively. We define the set of missing elements as $M = A \setminus B$ and the set of false elements as $F = B \setminus A$. The total error is

$$\varepsilon_{\text{SI}} = |M| + |F|. \tag{9}$$

Sorting. Let $a = (a_1, \ldots, a_n)$ be the correct ordering and $b = (b_1, \ldots, b_m)$ be the model output. The error has two components:

$$X = \sum_{i=1}^{m-1} \operatorname{sgn}(\max(b_i - b_{i+1}, 0)), \tag{10}$$

$$Y = \sum_{i=0}^{9} \left| |\{b_p : b_p = i\}| - |\{a_q : a_q = i\}| \right|, \tag{11}$$

where X counts adjacent pairs that violate the ordering, and Y measures the total deviation between the frequency of each value in the output and its ground-truth frequency. The total error is $\varepsilon_{\text{Sort}} = X + Y$, thereby accounting for both ordering and multiplicity errors.

Keyword counting. Let K be the total number of keywords, and let c_k and $\hat{c}_k$ denote the ground-truth and model-predicted frequencies of keyword k, respectively. The error is the sum of the counting deviations over all keywords:

$$\varepsilon_{\text{KW}} = \sum_{k=1}^{K} |c_k - \hat{c}_k|. \tag{12}$$

All results report the mean over 100 instances—the average error per sample, $\bar{\varepsilon} = \frac{1}{100} \sum_{j=1}^{100} \varepsilon^{(j)}$.

C.1.2 Inference Cost

Inference cost is measured in US dollars ($) and denotes the total expense of all LLM calls in one experiment. Input- and output-token costs are accumulated separately using fixed, backbone-specific rates across methods. For Qwen, input and output tokens cost $0.002 and $0.008 per 1K tokens, respectively; for Llama, they cost $0.004 and $0.012 per 1K tokens, respectively. This metric reflects both reasoning-path length and the number of LLM calls; lower cost indicates greater reasoning efficiency.

C.2 Granularity Identifiers and Routing Configuration

This section summarizes the granularity partitions and routing configurations obtained in the one-time probing phase. GID denotes a granularity-group identifier in DART's decomposition chain, not an entropy-bin index. Table 6 lists each GID's granularity and zone for both backbones. Granularity denotes the set-size pair (N_1, N_2) for set intersection, subarray length for sorting, and sentences per segment for keyword counting.

Zone A directly solves the subproblem without further decomposition, while retaining the configured self-critique. **Zone B** directly decomposes the subproblem. **Zone C** uses answer-token entropy: if $\bar{H}(v) < \tau_d$, DART follows the solving path, reusing the probe answer; otherwise, it further decomposes the subproblem. Table 7 gives the 1,000-probe thresholds used in the reported experiments for all Zone C configurations. The two backbones share zone assignments except at set-intersection GID 13, which belongs to Zone C for Qwen and Zone A for Llama. Sorting has no Zone C configurations.

Table 6: Granularity and zone for each GID. A: solve; B: decompose; C: entropy-gated.

Set intersection

GID	Granularity	Zone	
		Qwen	Llama
0	(128,128)	B	B
1	(128,64)	B	B
2	(128,32)	C	C
3	(128,16)	C	C
4	(128,8)	A	A
5	(128,4)	A	A
6	(128,2)	A	A
7	(64,64)	B	B
8	(64,32)	C	C
9	(64,16)	A	A
10	(64,8)	A	A
11	(64,4)	A	A
12	(64,2)	A	A
13	(32,32)	C	A
14	(32,16)	A	A
15	(32,8)	A	A
16	(32,4)	A	A
17	(32,2)	A	A

Sorting (same zones for both models)

GID	0	1	2	3	4	5	6
Granularity	128	64	32	16	8	4	2
Zone	B	B	A	A	A	A	A

Keyword counting (same zones for both models)

GID	0	1	2	3
Granularity	16	8	4	2
Zone	B	C	C	A

Table 7: Entropy thresholds for Zone C. Qwen and Llama denote the two backbones.

Model	Task / GID / Granularity	τ_d
Qwen	Set intersection GID 2 (128×32)	0.118094
Qwen	Set intersection GID 3 (128×16)	0.082937
Qwen	Set intersection GID 8 (64×32)	0.089339
Qwen	Set intersection GID 13 (32×32)	0.078192
Llama	Set intersection GID 2 (128×32)	0.490262
Llama	Set intersection GID 3 (128×16)	0.350520
Llama	Set intersection GID 8 (64×32)	0.450059
Qwen	Keyword counting GID 1 (8 sentences)	0.020876
Qwen	Keyword counting GID 2 (4 sentences)	0.012714
Llama	Keyword counting GID 1 (8 sentences)	0.048096
Llama	Keyword counting GID 2 (4 sentences)	0.042625

C.2.1 One-Time Probing Cost and Reuse

Table 8 reports recorded token usage for 1,000 probes per GID and model, before filtering, using the model-specific rates in Appendix C.1. It covers all set-intersection chains, all seven sorting GIDs, and keyword-counting GIDs 0–3, totaling 58,000 probes across both models. The same 1,000-probe cohorts are used for the region-calibration results in Appendix B.1.

Table 8: One-time probing token usage and costs in US dollars for 1,000 probes per GID and model before filtering. Costs use recorded token counts and the paper's fixed rates. GID ranges include both endpoints; totals are computed before rounding.

Task / Scale	GIDs	Samples	Model	Input tokens	Output tokens	Total tokens	Cost ($)
Set intersection, 128	0–6	7,000	Qwen	12,982,498	495,793	13,478,291	29.93
Set intersection, 128	0–6	7,000	Llama	9,596,000	451,524	10,047,524	43.80
Set intersection, 64	7–12	6,000	Qwen	8,735,317	233,816	8,969,133	19.34
Set intersection, 64	7–12	6,000	Llama	6,798,000	180,386	6,978,386	29.36
Set intersection, 32	13–17	5,000	Qwen	6,373,362	115,882	6,489,244	13.67
Set intersection, 32	13–17	5,000	Llama	5,056,000	94,674	5,150,674	21.36
Sorting, 128	0	1,000	Qwen	1,116,000	567,445	1,683,445	6.77
Sorting, 128	0	1,000	Llama	1,139,000	336,856	1,475,856	8.60
Sorting, 64	1	1,000	Qwen	924,000	216,091	1,140,091	3.58
Sorting, 64	1	1,000	Llama	947,000	190,606	1,137,606	6.08
Sorting, 32	2	1,000	Qwen	828,000	98,116	926,116	2.44
Sorting, 32	2	1,000	Llama	851,000	97,324	948,324	4.57
Sorting, 16	3	1,000	Qwen	780,000	49,015	829,015	1.95
Sorting, 16	3	1,000	Llama	803,000	49,054	852,054	3.80
Sorting, 8	4	1,000	Qwen	756,000	24,997	780,997	1.71
Sorting, 8	4	1,000	Llama	779,000	25,042	804,042	3.42
Sorting, 4	5	1,000	Qwen	744,000	13,000	757,000	1.59
Sorting, 4	5	1,000	Llama	767,000	13,000	780,000	3.22
Sorting, 2	6	1,000	Qwen	738,000	7,000	745,000	1.53
Sorting, 2	6	1,000	Llama	761,000	7,000	768,000	3.13
Keyword counting	0–3	4,000	Qwen	2,730,109	239,602	2,969,711	7.38
Keyword counting	0–3	4,000	Llama	2,817,685	276,504	3,094,189	14.59
Total (all tasks)		29,000	Qwen	36,707,286	2,060,757	38,768,043	89.90
Total (all tasks)		29,000	Llama	30,314,685	1,721,970	32,036,655	141.92

Under the same model and configuration, the calibrated router is reused for all subsequent requests for the covered tasks and granularities. Probing costs are thus amortized over continued service and reported separately from inference costs. Table 7 reports τ_d, the mean entropy after filtering the 1,000-probe cohort and the threshold used in the reported online experiments.

C.3 Implementation Details

We deploy Qwen3-32B and Llama3.1-70B-FP8 on $2\times$ NVIDIA H100 (80GB) GPUs. Both the solver and critic in DART use a sampling temperature of 0.8; IO, CoT, ToT, ToT2, and the GoT variants also use 0.8. AGoT uses 0.3, consistent with the official experiment scripts, while our AoT task adaptation uses 0.2. CR uses 0.7 for the proposer and 0.1 for both the verifiers and reporter.

Each online evaluation result reported in the main text is averaged over three independent runs, each comprising 100 instances. Within each run, mean error is computed over those 100 instances and inference cost is accumulated over the same 100 instances; the two run-level metrics are then averaged separately across the three runs.

The depth L and width k of ToT and ToT2 follow the GoT-based configuration described in the main paper and are set separately for each task and scale. For sorting, ToT uses $(L, k) \in \{(2, 20), (4, 20), (4, 20)\}$ and ToT2 uses $\{(3, 10), (7, 10), (10, 10)\}$. For set intersection, ToT uses $\{(2, 20), (4, 25), (3, 10)\}$ and ToT2 uses $\{(7, 10), (9, 10), (4, 20)\}$. These settings correspond, in the listed order, to input scales 32, 64, and 128, respectively. For keyword counting, ToT uses $(4, 20)$ and ToT2 uses $(6, 10)$. GoT uses two decomposition granularities for keyword counting: GoT4 divides the text into four segments, whereas GoT8 divides it into eight segments; subresults are then aggregated pairwise until the final answer is obtained. CR follows the original configuration, with a score threshold of 90 and a maximum of 3 retries. For DART, the extreme-score confidence threshold is $\tau_{\text{ext}} = 0.90$ for all three tasks; the normal-score confidence threshold is $\tau_{\text{norm}} = 0.6$.

When selection among retained candidates is required, DART's $\mathcal{S}$ performs local consistency checks on verification evidence provided by the critic and retains the candidate with the smallest estimated discrepancy; ties within a self-critique loop are resolved by the critic score and then generation order. The ToT, ToT2, and GoT baselines use the task-specific local error functions corresponding to the

measures in Appendix C.1.1 in their `Score–KeepBestN` steps, while retaining their respective search structures and selection stages.

AGoT uses the official `multi-agent-llm` implementation with maximum nested-graph depth 1, at most 2 graph layers, 3 new tasks per layer, and nested-graph layer-depth reduction. It terminates after solving a final task, generated either during expansion or at the layer limit. AoT follows the official implementation: direct and decomposition-based solving, dependency labeling, contraction, contracted-question solving, and ensemble selection, then terminates without additional depth or width parameters. Both methods use a $4{,}096$-token output limit per LLM call, with task-specific descriptions, output formats, and answer parsing.

In implementation, DART follows the zone assignments in Table 6, applying entropy-based decisions only in Zone C with the 1,000-probe thresholds τ_d in Table 7. Zones A and B follow fixed direct-solving and decomposition paths, respectively. Based on the observed error levels and entropy–error relationships in one-time probing, both backbones use Zone B at sorting GIDs 0–1 and Zone A at GIDs 2–6, without requiring instance-level entropy gating.

C.4 Relationship between Answer-Token Entropy and Solution Error

This section systematically analyzes the relationship between answer-token entropy and solution error across two models (Qwen3-32B and Llama3.1-70B-FP8) and three tasks (set intersection, sorting, and keyword counting). The entropy–error plots and the region calibration in Appendix B.1 use 1,000 samples per model–task–GID combination before filtering. Probe-cost accounting and calibration reuse are discussed in Appendix C.2.1. For each model–task–GID combination, we sort retained samples by answer-level mean Shannon entropy and divide them into 8 equal-frequency bins. The horizontal axis shows the mean answer entropy of each bin, and the vertical axis shows its average error. Here, GID denotes a task-scale group in DART's decomposition chain rather than an entropy-bin index. The specific input scale appears below each subplot and is summarized in the corresponding caption.

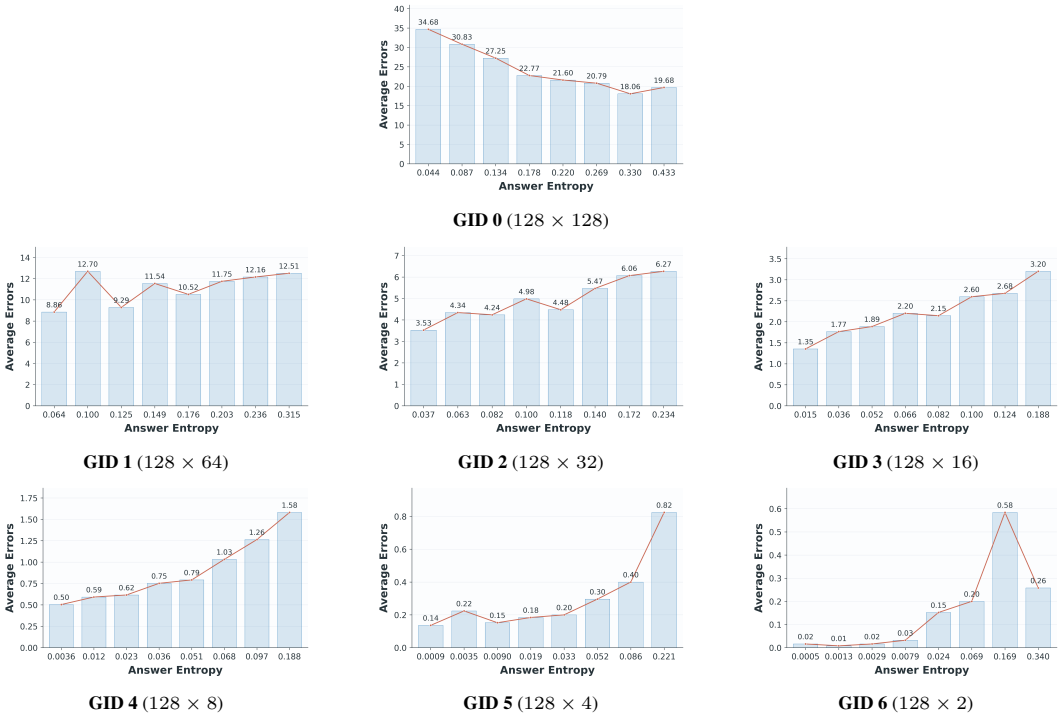
Overall observation. The entropy–error relationship varies across tasks and granularities. Region calibration therefore uses both absolute error level and entropy–error correlation, as detailed in Appendix B.1. Strong positive correlation can persist in easy or difficult regions, notably in keyword counting, and does not by itself imply a need for entropy gating.

Set intersection. Figures 7–12 report results for all GIDs, grouped by the size of the first set so that each figure presents one continuous granularity sequence. Table 7 summarizes the combinations that use entropy-based decisions. At GID 13 (32×32), Qwen uses entropy gating in Zone C, whereas Llama directly solves problems in Zone A.

Sorting. Figures 13 and 14 show that correlation varies across granularities and can be strongly positive in both A and B. The GMM selects two components by BIC, so sorting uses fixed solving and decomposition routes without a C region.

Keyword counting. Figures 15 and 16 show a more stable positive entropy–error relationship for GIDs 1 and 2, which therefore use entropy-based decisions in Zone C. GID 0 directly decomposes in Zone B, while GID 3 directly solves in Zone A on both backbones.

Implication for routing design. One-time probing combines error level and entropy–error correlation to identify solving regimes. Low- and high-error regions receive fixed direct-solving and decomposition routes, respectively; in Zone C, instance-level entropy determines whether to retain the initial answer or decompose.



GID 0 (128×128)

GID 1 (128×64)

GID 2 (128×32)

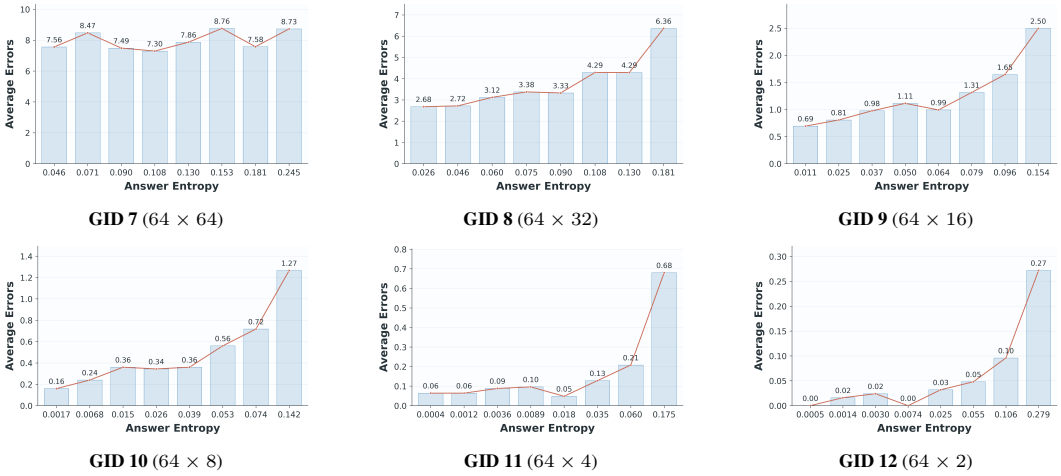
GID 3 (128×16)

GID 4 (128×8)

GID 5 (128×4)

GID 6 (128×2)

Figure 7: Answer-token entropy and average error along the $128 \times \{128, 64, 32, 16, 8, 4, 2\}$ granularity sequence for Qwen3-32B on set intersection (GIDs 0–6).



GID 7 (64×64)

GID 8 (64×32)

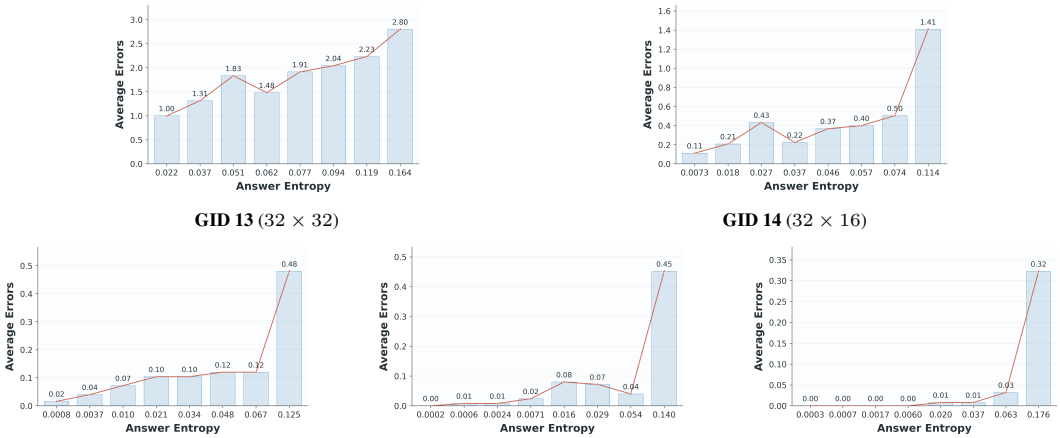
GID 9 (64×16)

GID 10 (64×8)

GID 11 (64×4)

GID 12 (64×2)

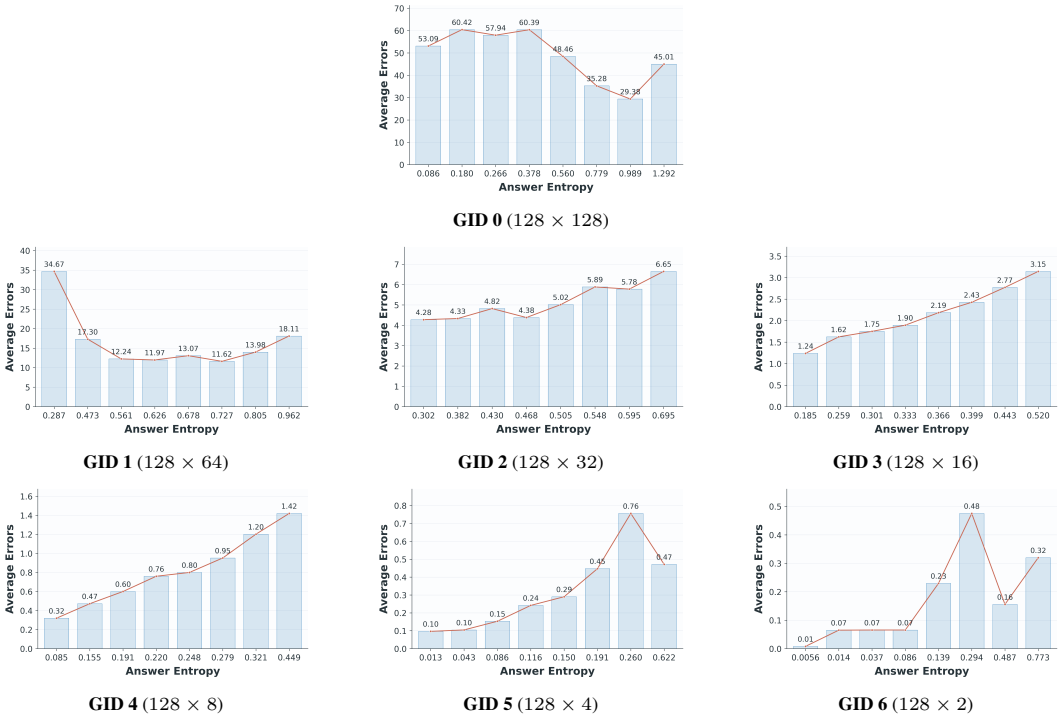
Figure 8: Answer-token entropy and average error along the $64 \times \{64, 32, 16, 8, 4, 2\}$ granularity sequence for Qwen3-32B on set intersection (GIDs 7–12).



GID 13 (32×32) **GID 14** (32×16)

GID 15 (32×8) **GID 16** (32×4) **GID 17** (32×2)

Figure 9: Answer-token entropy and average error along the $32 \times \{32, 16, 8, 4, 2\}$ granularity sequence for Qwen3-32B on set intersection (GIDs 13–17). GID 13 uses entropy gating.

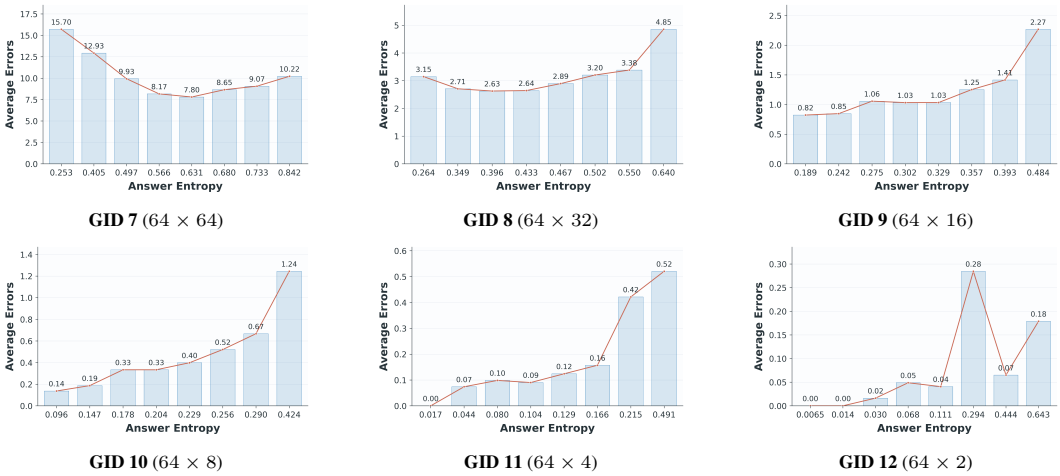


GID 0 (128×128)

GID 1 (128×64) **GID 2** (128×32) **GID 3** (128×16)

GID 4 (128×8) **GID 5** (128×4) **GID 6** (128×2)

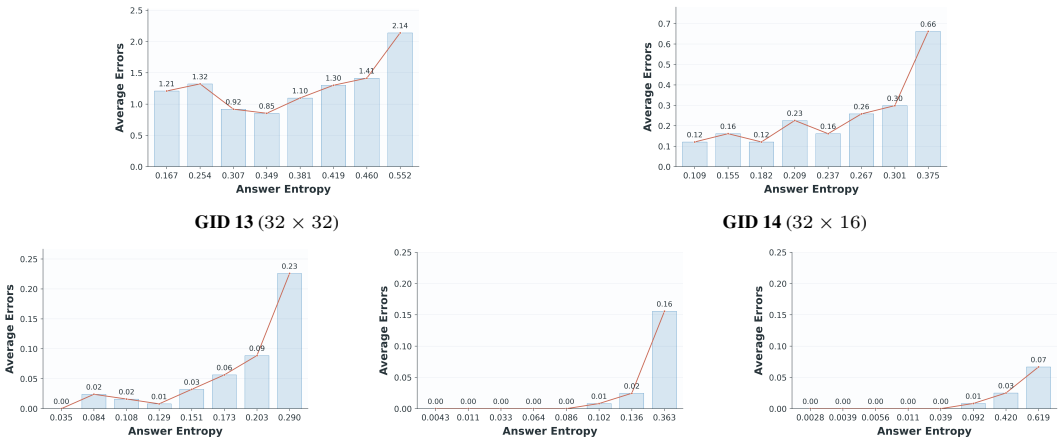
Figure 10: Answer-token entropy and average error along the $128 \times \{128, 64, 32, 16, 8, 4, 2\}$ granularity sequence for Llama3.1-70B-FP8 on set intersection (GIDs 0–6).



GID 7 (64×64) **GID 8** (64×32) **GID 9** (64×16)

GID 10 (64×8) **GID 11** (64×4) **GID 12** (64×2)

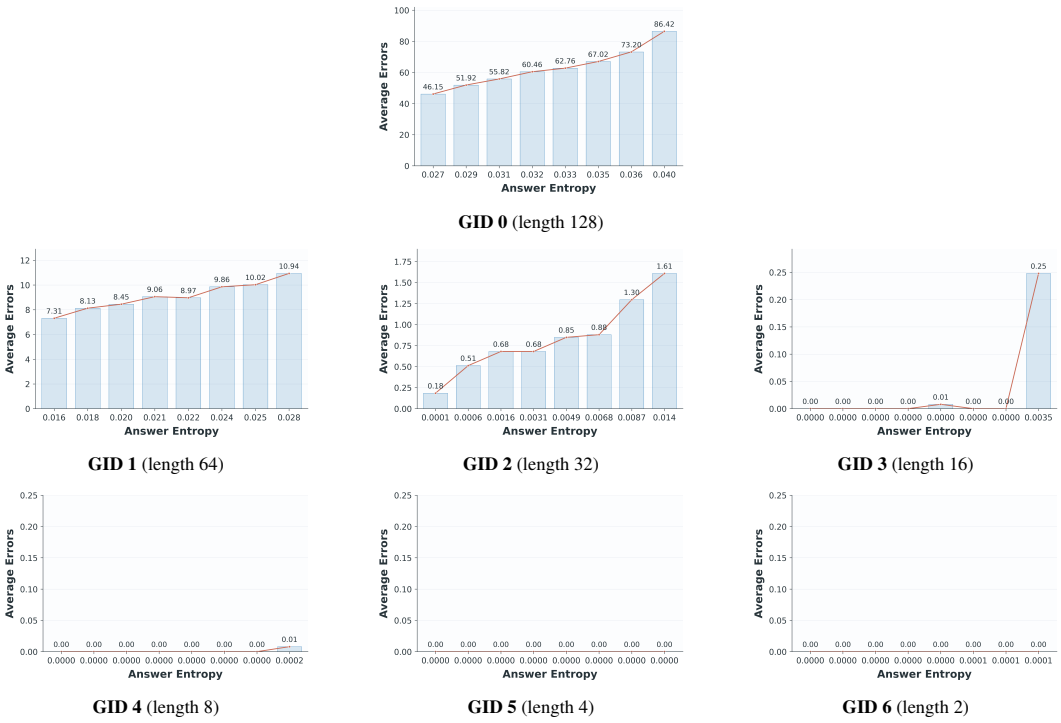
Figure 11: Answer-token entropy and average error along the $64 \times \{64, 32, 16, 8, 4, 2\}$ granularity sequence for Llama3.1-70B-FP8 on set intersection (GIDs 7–12).



GID 13 (32×32) **GID 14** (32×16)

GID 15 (32×8) **GID 16** (32×4) **GID 17** (32×2)

Figure 12: Answer-token entropy and average error along the $32 \times \{32, 16, 8, 4, 2\}$ granularity sequence for Llama3.1-70B-FP8 on set intersection (GIDs 13–17). GID 13 follows direct solving without an entropy decision.



GID 0 (length 128)

GID 1 (length 64)

GID 2 (length 32)

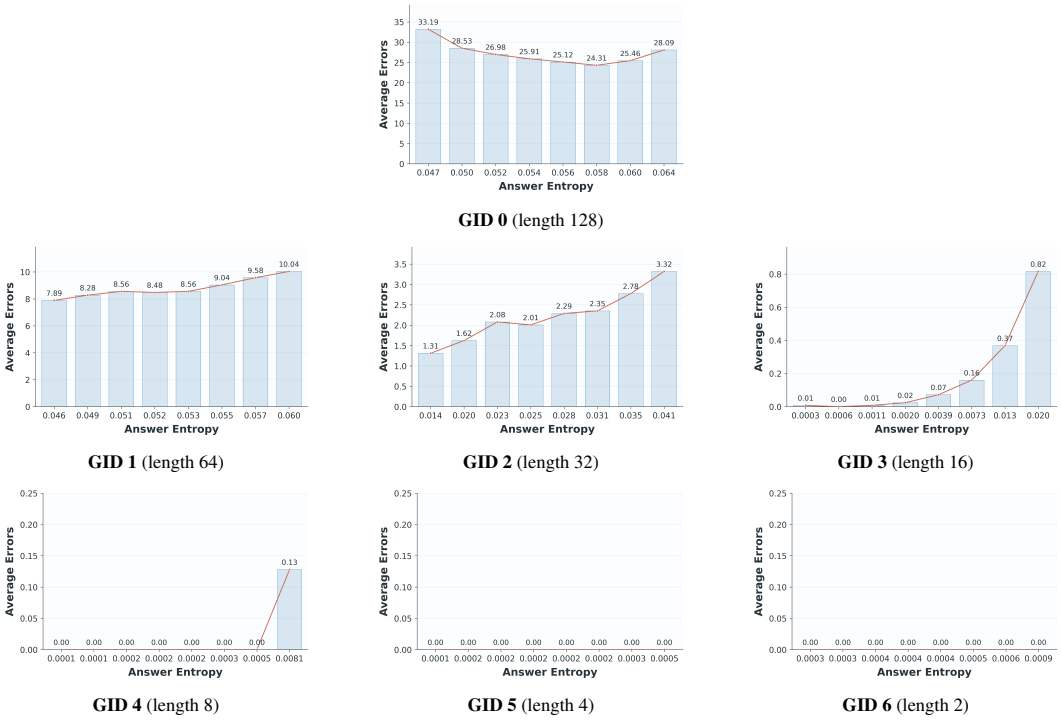
GID 3 (length 16)

GID 4 (length 8)

GID 5 (length 4)

GID 6 (length 2)

Figure 13: Relationship between answer-token entropy and average error for Qwen3-32B on sorting. The subplots correspond to GIDs 0–6, with list lengths 128, 64, 32, 16, 8, 4, and 2. The final DART configuration does not use entropy gating for sorting.



GID 0 (length 128)

GID 1 (length 64)

GID 2 (length 32)

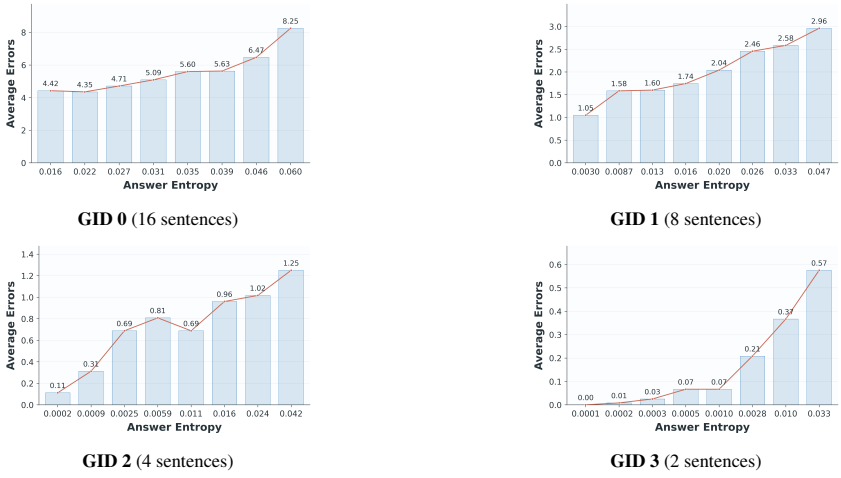
GID 3 (length 16)

GID 4 (length 8)

GID 5 (length 4)

GID 6 (length 2)

Figure 14: Relationship between answer-token entropy and average error for Llama3.1-70B-FP8 on sorting. The subplots correspond to GIDs 0–6, with list lengths 128, 64, 32, 16, 8, 4, and 2. BIC selects two components, assigned to fixed solving and decomposition routes.



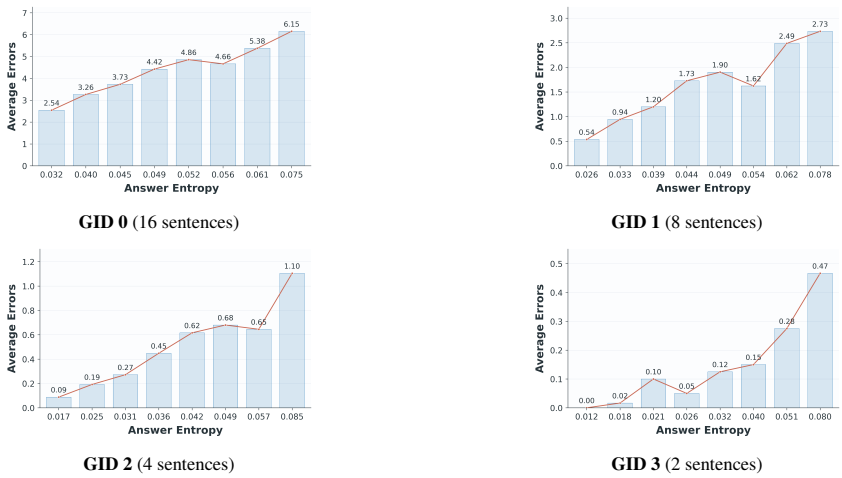
GID 0 (16 sentences)

GID 1 (8 sentences)

GID 2 (4 sentences)

GID 3 (2 sentences)

Figure 15: Relationship between answer-token entropy and average error for Qwen3-32B on keyword counting. The subplots correspond to GIDs 0–3, with input texts containing 16, 8, 4, and 2 sentences. Positive correlations persist across GIDs; the joint feature partition assigns GIDs 1–2 to entropy gating.



GID 0 (16 sentences)

GID 1 (8 sentences)

GID 2 (4 sentences)

GID 3 (2 sentences)

Figure 16: Relationship between answer-token entropy and average error for Llama3.1-70B-FP8 on keyword counting. The subplots correspond to GIDs 0–3, with input texts containing 16, 8, 4, and 2 sentences.

C.5 Complete Tables of Main Results

This appendix provides complete tables of the main results discussed in Section 4.2 of the main paper: set intersection (Table 9), sorting (Table 10), and keyword counting (Table 11).

Table 9: Set-intersection results. Error↓: average error per instance over 100 samples, reported as a decimal (see Appendix C.1.1); Cost↓: total inference cost in US dollars ($).

Method	Qwen3-32B						Llama3.1-70B-FP8					
	$N = 32$		$N = 64$		$N = 128$		$N = 32$		$N = 64$		$N = 128$	
	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓
IO	1.83	0.32	7.85	0.42	26.15	0.69	1.19	0.49	9.39	0.66	56.91	1.16
CoT	0.28	0.88	7.59	1.24	25.94	1.68	0.55	1.23	5.52	1.76	37.19	2.63
ToT	0.14	11.15	3.15	14.60	10.40	74.42	0.00	9.99	0.94	33.11	8.24	87.99
ToT2	0.34	6.57	4.05	7.34	12.87	29.76	0.00	6.06	1.17	17.97	9.29	44.31
GoT	0.45	1.48	4.04	4.08	14.44	7.84	0.04	2.12	2.01	5.35	10.15	9.80
AGoT	1.59	2.29	6.65	2.60	22.53	3.21	0.20	6.88	2.43	7.15	19.05	8.20
AoT	0.25	2.56	3.87	3.68	14.86	6.00	0.11	3.82	1.08	6.22	13.94	10.00
CR	0.22	1.81	4.38	5.99	19.25	15.95	1.03	3.35	2.33	7.69	25.13	22.21
DART	0.19	1.96	1.30	5.66	7.75	19.40	0.03	2.42	0.83	8.85	6.43	27.91

Table 10: Sorting results.

Method	Qwen3-32B						Llama3.1-70B-FP8					
	$N = 32$		$N = 64$		$N = 128$		$N = 32$		$N = 64$		$N = 128$	
	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓	Error↓	Cost↓
IO	0.78	0.24	8.69	0.36	64.53	0.68	2.00	0.46	8.76	0.61	23.81	0.88
CoT	4.60	0.81	18.47	1.12	81.94	1.94	10.12	1.32	44.27	1.77	152.14	2.09
ToT	0.49	14.35	5.84	25.37	21.51	74.88	0.74	19.07	5.23	27.78	16.06	38.24
ToT2	0.49	7.36	6.27	13.22	30.52	33.61	0.86	11.48	5.56	13.49	17.27	19.72
GoT	1.03	8.44	4.56	21.81	11.22	49.70	0.65	11.61	5.69	27.32	18.52	64.09
AGoT	0.83	2.33	7.40	2.58	69.36	5.27	1.92	17.17	10.18	18.88	30.92	21.37
AoT	6.74	1.94	16.62	2.96	111.95	23.79	0.95	4.04	7.78	5.07	28.97	8.57
CR	1.14	3.56	7.66	8.35	56.24	20.05	3.64	7.29	8.49	14.84	65.95	36.39
DART	0.54	1.07	1.43	5.85	5.56	17.16	1.55	2.93	4.54	13.06	14.90	35.36

Table 11: Keyword-counting results.

Method	Qwen3-32B		Llama3.1-70B-FP8	
	Error↓	Cost↓	Error↓	Cost↓
IO	5.29	0.30	5.27	0.55
CoT	6.41	0.50	42.74	1.60
ToT	2.04	19.11	2.24	51.98
ToT2	2.56	10.55	4.45	27.73
GoT4	3.04	5.37	6.26	24.27
GoT8	1.93	8.64	1.23	41.33
AGoT	5.72	2.80	4.66	13.89
AoT	5.77	2.79	2.96	4.48
CR	4.27	17.83	1.48	31.05
DART	1.80	5.42	1.18	9.65

C.6 Sensitivity to the Entropy Threshold

We evaluate entropy-threshold sensitivity on Qwen3-32B keyword counting with $T = 3$. At each granularity in Zone C, we set τ_d to the 10th through 90th percentiles of answer-token entropy from 1,000 probing samples, in increments of ten percentile points. Other settings follow DART.

Figure 17 and Table 12 show that increasing the quantile from q_{10} to q_{90} raises average error from 0.64 to 2.70 while reducing total inference cost from \$8.54 to \$3.88. Lower thresholds trigger more decomposition, improving accuracy at greater cost; higher thresholds favor direct solving, saving cost but increasing error. Thus, τ_d controls the error–cost trade-off. DART uses the mean answer-token entropy of the 1,000 probing samples at each granularity as the default threshold, avoiding an additional choice of quantile level and facilitating consistent calibration and reuse. The mean* point reports the main-experiment result under the 1,000-probe mean thresholds in Table 7.

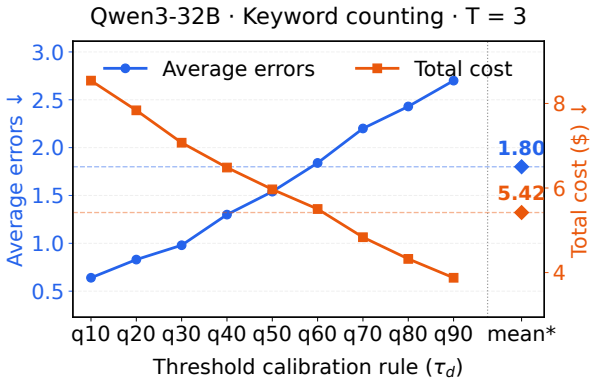


Figure 17: Entropy-threshold sensitivity on Qwen3-32B keyword counting ($T = 3$). Quantiles use 1,000 probes per GID. The mean* point and dashed lines show the main-experiment reference using the 1,000-probe mean threshold.

Table 12: Entropy-threshold results. Error: mean errors per instance; Cost: total online inference cost over 100 instances. mean*: main-experiment reference.

Rule	Error ↓	Cost ($) ↓
q10	0.64	8.54
q20	0.83	7.84
q30	0.98	7.07
q40	1.30	6.48
q50	1.54	5.97
q60	1.84	5.50
q70	2.20	4.84
q80	2.43	4.32
q90	2.70	3.88
mean*	1.80	5.42

C.7 PAIRED-BRANCH TEST OF ENTROPY-GUIDED ROUTING

Setup. We test whether answer-token entropy identifies Zone C subproblems for which decomposition is more valuable than retaining the direct answer. We use set intersection (SI) at GID 8 (64×32) and keyword counting (KC) at GID 1 (eight sentences), each with Qwen3-32B and Llama3.1-70B-FP8. For each task, we randomly select one contiguous half from each of 100 original test problems, yielding 100 subproblems per task; both backbones receive the same selected inputs. We classify a subproblem as low entropy when $\bar{H}(v) < \tau_d$ and high entropy otherwise, using the corresponding 1,000-probe threshold in Table 7, without recalibrating it on these test cases. Each subproblem is evaluated along two paired branches: (i) retain the initial answer and follow the direct-solving path, or (ii) force decomposition at the current node and then follow the usual downstream procedure. Under the entropy gate, decomposition is the prescribed action for the high-entropy group and direct solving for the low-entropy group; the other branch in each group is the forced alternative. Both branches include the shared initial solving attempt in their reported cost and use the same critic, correction limit, and other settings. In total, the experiment contains 400 matched subproblems and 800 branch outcomes; all were checked against the recorded inputs, outputs, and costs. Sorting is not included because its routing configuration has no Zone C (Table 6).

Table 13: Paired direct-solving and forced-decomposition outcomes by entropy group. Error is the mean per subproblem; cost is the total estimated USD over the n subproblems in each row. Values are reported to two decimal places; arrows are calculated from these values relative to direct solving on the same subproblems. Both branches include the initial solving attempt.

Task	Backbone	Entropy	n	Error ↓		Cost ($) ↓	
				Direct	Decompose	Direct	Decompose
Set intersection	Qwen	High	39	1.08	0.41 ↓ 62.04%	0.99	1.32 ↑ 33.33%
	Qwen	Low	61	0.89	0.36 ↓ 59.55%	1.15	2.02 ↑ 75.65%
	Llama	High	44	0.52	0.43 ↓ 17.31%	1.33	1.44 ↑ 8.27%
	Llama	Low	56	0.32	0.29 ↓ 9.38%	1.51	2.71 ↑ 79.47%
Keyword counting	Qwen	High	46	2.35	0.46 ↓ 80.43%	0.71	1.23 ↑ 73.24%
	Qwen	Low	54	1.31	0.48 ↓ 63.36%	0.63	1.52 ↑ 141.27%
	Llama	High	40	1.48	0.25 ↓ 83.11%	1.12	1.79 ↑ 59.82%
	Llama	Low	60	0.93	0.50 ↓ 46.24%	1.55	2.86 ↑ 84.52%

Decomposition benefits and costs by entropy group. For high-entropy subproblems, DART selects decomposition; direct solving is the forced alternative. In the four task–backbone settings in Table 13, decomposition reduces mean error by 62.04%, 17.31%, 80.43%, and 83.11%, while increasing cost by 33.33%, 8.27%, 73.24%, and 59.82%, respectively. In every setting, the proportional error reduction exceeds the proportional cost increase, supporting DART's decision to decompose these high-entropy subproblems.

For low-entropy subproblems, DART retains the direct answer; decomposition is the forced alternative. Decomposition still reduces mean error by 59.55%, 9.38%, 63.36%, and 46.24%, but increases cost by 75.65%, 79.47%, 141.27%, and 84.52%, respectively. Here the proportional cost increase exceeds the error reduction in every setting, so direct solving avoids comparatively costly decomposition without implying that splitting has no accuracy benefit.

Together, these paired results show that high-entropy subproblems have higher direct-solving error and, in every task–backbone setting, gain a larger proportional error reduction at a smaller proportional cost increase than low-entropy subproblems. The entropy gate therefore assigns decomposition to the group with the more favorable observed error–cost trade-off, directly supporting the routing choice used in DART.

Entropy-guided versus random routing. To make the policy implication explicit, we reconstruct the outcome of applying the existing entropy gate to these paired records: decompose high-entropy subproblems and retain the direct branch for low-entropy ones. Writing n_H and n_L for the high- and low-entropy group sizes, its mean error is $(n_H \bar{\varepsilon}_{H,\text{decompose}} + n_L \bar{\varepsilon}_{L,\text{direct}})/100$, and its total cost

is $C_{H,\text{decompose}} + C_{L,\text{direct}}$, using the group means and totals in Table 13. We compare this with the *expected* outcome of uniformly selecting exactly n_H of the 100 subproblems for decomposition. Each subproblem then has decomposition probability $p = n_H/100$, so the expected error or cost equals the all-direct aggregate plus p times the difference between the all-decomposition and all-direct aggregates. For SI/Qwen, for example, entropy gating gives $(39 \times 0.41 + 61 \times 0.89)/100 \approx 0.70$ mean error and $1.32 + 1.15 = \$2.47$ total cost; the same-count random expectation gives approximately 0.74 error and \$2.61 cost. Table 14 uses this arithmetic reconstruction from the two-decimal group values in Table 13; it is not a separately executed random-routing experiment or a strict cost-matched control.

Table 14: Policy reconstruction over the same 100 paired subproblems per task and backbone. Entropy gating decomposes the high-entropy group; random routing is an analytical expectation with the same number of decompositions, not a new API run. Cost is total estimated USD. Arrows in Random exp. show percentage changes relative to Entropy, computed before policy-level rounding.

Task	Backbone	n_{high}	n_{low}	Error ↓		Cost (\$) ↓	
				Entropy	Random exp.	Entropy	Random exp.
Set intersection	Qwen	39	61	0.70	0.74 ↑ 4.74%	2.47	2.61 ↑ 5.59%
	Llama	44	56	0.37	0.38 ↑ 4.01%	2.95	3.42 ↑ 15.81%
Keyword counting	Qwen	46	54	0.92	1.18 ↑ 28.65%	1.86	1.99 ↑ 6.91%
	Llama	40	60	0.66	0.85 ↑ 29.18%	3.34	3.46 ↑ 3.65%

Table 13 explains why the entropy decision matters: the reduction in total error per additional dollar, $n(\bar{\varepsilon}_{\text{direct}} - \bar{\varepsilon}_{\text{decompose}})/(C_{\text{decompose}} - C_{\text{direct}})$, is approximately 2.13, 25.71, 3.32, and 3.73 times greater for high- than low-entropy subproblems in SI/Qwen, SI/Llama, KC/Qwen, and KC/Llama, respectively, using the reported two-decimal group values. Table 14 then holds the number of decompositions fixed: uniformly random selection has higher expected error (by 4.74%, 4.01%, 28.65%, and 29.18%) and higher expected cost (by 5.59%, 15.81%, 6.91%, and 3.65%) than entropy routing in the four settings. Thus, the benefit comes from choosing *which* subproblems to decompose, not merely how many. This local comparison is complemented by the end-to-end random-routing experiment in Table 4: at $p = 0.50$, random routing increases both error and cost in all four settings, and none of the tested probabilities improves both metrics uniformly. Together, these results support the effectiveness of entropy-guided routing for the tested tasks and models.

C.8 Bootstrap Stability of Region Calibration

We perform 100 bootstrap repetitions using the existing calibration records. In each repetition, we independently sample 1,000 probes with replacement within every model–task–GID group, before filtering. We then repeat the procedure in Appendix B.1: the same filtering and eight entropy bins, recomputation of $(\mu_{m,d}, \rho_{m,d})$, task-specific feature normalization and covariance regularization, and GMM fitting with BIC selection over $K \in \{1, 2, 3\}$. All other settings remain fixed (bootstrap seed: 20260925); no additional model calls are required.

Table 15: Stability over 100 bootstrap repetitions relative to the original 1,000-probe calibration. Same K and unchanged partition count repetitions; label retention is the fraction of all model–GID assignments matching the original labels.

Task	Original K	Same K	Unchanged partition	Label retention
Set intersection	3	99/100	76/100	98.97%
Sorting	2	100/100	100/100	100.00%
Keyword counting	3	100/100	70/100	92.50%

In Table 15, 5,703/5,800 labels remain unchanged (98.33%), averaging 57.03/58 per repetition; 52 repetitions preserve all 58 labels. Sorting is invariant. Changes concentrate in keyword-counting GIDs 1–2 and Qwen set-intersection GID 7, indicating high overall retention with greater sensitivity in keyword counting.

Table 16: Zone frequencies over 100 bootstrap repetitions for the five model–GID pairs with the highest label-change rates. Original zone denotes the 1,000-probe calibration; bold values indicate its retention frequency.

Task	Backbone	GID	Original zone	A	C	B
Keyword counting	Qwen	1	C	0%	**81%**	19%
Keyword counting	Qwen	2	C	11%	**89%**	0%
Keyword counting	Llama	1	C	0%	**81%**	19%
Keyword counting	Llama	2	C	11%	**89%**	0%
Set intersection	Qwen	7	B	0%	16%	**84%**

Table 16 shows that keyword-counting GIDs 1 and 2 retain Zone C in 81% and 89% of repetitions, respectively, for both backbones. Qwen set-intersection GID 7 retains Zone B in 84%. Thus, the original labels remain dominant even at these sensitive locations, although region assignments are not invariant to resampling.

D Prompt Templates and Selected Examples

This appendix retains the instruction, approach, input, and output structure of the prompts executed in the three tasks. To avoid repeating long scale-specific demonstrations and interaction traces, we show representative examples but omit redundant copies. Italic notes describing omitted examples are editorial, not text sent to the model. Runtime variables, including task size and intermediate results, appear in braces. The shared critic continuation appears first; Appendices D.1, D.2, and D.3 give the task prompts.

Confidence-Alignment Prompts Shared by All Tasks

At each self-critique iteration, we take q_v as the numeric score following the final `Score` field in the critic's output. Its confidence c_v is the probability assigned to that score token, or the arithmetic mean of the corresponding token probabilities if the score spans multiple tokens; earlier score mentions are not used. If the score format is invalid or confidence falls below the threshold for the score range, the same candidate is reevaluated at most once. The final evaluation determines acceptance: $q_v > 99$ and $c_v \geq \tau_{\text{ext}}$. If the candidate is not accepted, the solver refines it using the final critique feedback when another iteration remains; otherwise, iteration ends and candidate selection is performed. The illustrative templates in Table 17 use τ_{ext} and τ_{norm} for the extreme-score and normal-score confidence thresholds, respectively; their task-specific values are given in Appendix C.3. Table 18 illustrates the procedure for set intersection.

Table 17: Illustrative confidence-alignment prompts shared across all three tasks; confidence thresholds follow Appendix C.3, and runtime variables are denoted by curly braces.

Active thresholds and reflection policy

Acceptance requires $q_v > 99$ and $c_v \geq \tau_{\text{ext}}$. At most two evaluations are performed for each candidate answer.

Condition 1 -- Invalid score format

```
Score format invalid.  Please output 'Score:  <number>'.
```

Condition 2 -- Score = 100 but confidence $< \tau_{\text{ext}}$

```
High score {score_val} but low confidence {confidence}.  If unsure,
score <= 96.  Re-evaluate.
```

Condition 3 -- Score < 10 and confidence $< \tau_{\text{ext}}$

```
Low score {score_val} but low confidence {confidence}.  Re-evaluate.
```

Condition 4 -- Other scores with confidence $< \tau_{\text{norm}}$

```
Confidence {confidence} is too low.  Re-evaluate strictly.
```

Conversation continuation template for the critique prompt

[Assistant Evaluation]: {raw_critique}
[System Feedback]: {message}
IMPORTANT: Output 'Score: <number>' and 'Error: ...'.
[Corrected Evaluation]:

Table 18: An illustrative example of confidence alignment for set intersection under the $q_v > 99$, $c_v \geq \tau_{\text{ext}}$ acceptance rule.

Context

The candidate answer remains `[2]` throughout, while the correct intersection is `[2, 6]`. The initial critique incorrectly reports no errors and assigns `Score = 100`. Its low confidence triggers reevaluation of the same candidate, which identifies the missing element `6`; the candidate itself is not modified.

Initial critic output (incorrect evaluation)

Rational: Input-2 Membership Audit: [1:N, 2:Y, 6:Y, 8:N]; Two-pointer matches: [2, 6]; Proposed Sorted: [2]; Missing Elements: []; False Positives: []; Duplicates: []; Expected Count: 2; Proposed Count: 1; Total Issues: 0.
Error: None **Score:** 100

Appended continuation

[Assistant Evaluation]: Rational: Input-2 Membership Audit: [1:N, 2:Y, 6:Y, 8:N]; Two-pointer matches: [2, 6]; Proposed Sorted: [2]; Missing Elements: []; False Positives: []; Duplicates: []; Expected Count: 2; Proposed Count: 1; Total Issues: 0. Error: None Score: 100
[System Feedback]: High score 100.0 but low confidence 0.61. If unsure, score <= 96. Re-evaluate.
IMPORTANT: Output 'Score: <number>' and 'Error: ...'.
[Corrected Evaluation]:

Corrected evaluation of the same candidate

Rational: Input-2 Membership Audit: [1:N, 2:Y, 6:Y, 8:N]; Two-pointer matches: [2, 6]; Proposed Sorted: [2]; Missing Elements: [6]; False Positives: []; Duplicates: []; Expected Count: 2; Proposed Count: 1; Total Issues: 1.
Error: Missing Elements: 6; False Positives: None; Duplicates: None **Score:** 98

D.1 SET INTERSECTION

The set-intersection task includes decomposition, direct solving, result critique, answer correction, and aggregation prompts. At the solving stage, the input sets and candidate intersection are programmatically presorted for the critic's two-pointer audit. At aggregation, the critic checks concatenation integrity rather than computing a set union. Tables 19 and 20 give the templates and selected examples.

Table 19: Prompt templates for set intersection (runtime parameters are denoted by curly braces; few-shot examples are provided in Table 20).

Direct solving prompt

<Instruction> Find the intersection of two sets of numbers. Output only the set of numbers that are present in both sets, no additional text. </Instruction>
<*Examples*> Three in-context examples are embedded in code (16/32/64-size style). </*Examples*>
Input Set 1: {set1} **Input Set 2:** {set2}

Two-way contiguous decomposition prompt

<Instruction> Split the following list of {size} numbers into 2 lists of {size//2} numbers each, the first list should contain the first {size//2} numbers and the second list the second {size-size//2} numbers. Only output the 2 lists in the following format without any additional text or thoughts!: { "List 1": [13, 16, 30, 6, 21, 7, 31, ...], "List 2": [25, 24, 10, 4, 27, 0, 14, ...] } </Instruction>
<*Example*> A size-matched split example is provided for each active input scale (64/32/16/8). </*Example*>
Input: {input_list_str}

Solve-stage two-pointer audit prompt

<Instruction> You are a strict logic verifier. Your task is to evaluate whether the Proposed Intersection is the intersection set of Input Set 1 and Input Set 2. </Instruction>
<task> Your task is to output a strict evaluation report. The input sets and candidate intersection are programmatically presorted for auditing; set order does not affect the result.
<Rationale_Step>1. Recompute the reference intersection with a two-pointer scan of the two sorted input sets: - Start at the smallest unused value in each input. - If the values are equal, record that value as a match and advance both pointers. - Otherwise advance only the pointer at the smaller value. - Continue until one input is exhausted. Do NOT use Proposed Intersection to decide which matches to record. - Mandatory coverage check: write one `value:Y` or `value:N` entry for EVERY value in Input Set 2 Sorted, in order. `Y` means the value was found in Input Set 1 Sorted. The audit entry count must equal the Input Set 2 count.
2. Collect every `Y` value into `Two-pointer matches: [...]` and compare it with Proposed Intersection Sorted as unordered multisets: - Missing Elements = matches absent from Proposed Intersection Sorted. - False Positives = proposed values absent from the matches. - Duplicates = only extra repeated proposed occurrences. - Never report an error merely because the original output order was different.
3. Count Cross-check: - State Expected Count and Proposed Count. - Before claiming `None`, verify that every expected value is present, every proposed value is expected, and there are no extra occurrences.
4. Error Summary: - Explicitly list Missing Elements, False Positives, and Duplicates in the audit ledger. - If no issues exist, use "None".
5. Score Calculation: - Start from 100. - Subtract 2 points for EACH Missing Element, False Positive, and Duplicate. - Minimum score is 0. - A Score of 100 is allowed only when both lists match exactly as multisets and all three error categories are empty. </Rationale_Step></task>
Input Set 1 Sorted: {set1_sorted} **Input Set 2 Sorted:** {set2_sorted} **Proposed Intersection Sorted:** {proposed_sorted}
IMPORTANT: Output ONLY the following format.
Rational: Input-2 Membership Audit: [value:Y/N, ... one entry per Input Set 2 value]; Two-pointer matches: [...]; Proposed Sorted: [...]; Missing Elements: [...]; False Positives: [...]; Duplicates: [...]; Expected Count: <number>; Proposed Count: <number>; Total Issues: <number>. Error: <Description of errors or 'None'>Score: <number>

Table 19: Prompt templates for set intersection (continued).

Solve-stage correction prompt

<Instruction>Your specific task is to CORRECT the 'Previous Incorrect Result' based strictly on the provided 'Critical Feedback'. You are NOT calculating from scratch. You must perform these operations: 1. Handle Missing Elements: ADD these numbers to the list. 2. Handle False Positives: REMOVE these numbers from the list. 3. Handle Duplicates: Remove extra instances so each number appears only once. Output ONLY in the following format: Rational: <brief reasoning explaining each correction step>Corrected Output: [num1, num2, ...] </Instruction>
<*Example*> A size-matched correction example is provided for each active input scale. </*Example*>
<Immediate Correction Task>Input Set 1: {set1} Input Set 2: {input_set} Previous Incorrect Result: {current} Critical Feedback: {latest_feedback} </Immediate Correction Task>

Aggregation prompt

<Instruction> Merge the following 2 lists into one list by appending the second list to the first list. Only output the final list without any additional text or thoughts! </Instruction>
List 1: {input1} **List 2:** {input2}

Aggregation integrity audit prompt

<Instruction> You are a data integrity verifier. Verify if the 'Merged Result' is the correct concatenation (splicing) of 'List 1' and 'List 2'. </Instruction>
IMPORTANT: This is a simple LIST MERGE operation, NOT a mathematical set union. - The goal is simply to combine all elements from List 1 and List 2 into one list. - The Merged Result must contain exactly all numbers from both lists.
<task>Your validation checklist: 1. Completeness Check (List 1). 2. Completeness Check (List 2). 3. False Positive Check. 4. Count Check. 5. Assign a score from 0 to 100. 6. List specific errors if any. </task>
<current_input>Input List 1: {input1} Input List 2: {input2} Merged Result: {current} </current_input>
IMPORTANT: Output ONLY the following format. Do NOT output markdown, reasoning, or intro/outro text.
Score: <number>Error: <Description of errors or 'None'>

Aggregation correction prompt

<Instruction> You are correcting a failed list merge operation based on specific feedback. Your Goal: Create a new merged list that strictly satisfies the feedback. CRITICAL RULES: 1. Simple Append: Just put List 2 after List 1. 2. Count Check: The length of your output MUST equal len(List 1) + len(List 2). </Instruction>
<Current Immediate Task>Input List 1: {input1} Input List 2: {input2} Previous Incorrect Result: {last_attempt} Critical Feedback: {latest_feedback} </Current Immediate Task>
Please fix the error listed in 'Critical Feedback' and output the correct list.
Corrected Merged List:

Table 20: Few-shot examples embedded in the set intersection prompts (selected examples shown).

Direct solving example

Input Set 1: [13, 16, 30, 6, 21, 7, 31, 15, 11, 1, 24, 10, 9, 3, 20, 8]
Input Set 2: [25, 24, 10, 4, 27, 0, 14, 12, 8, 2, 29, 20, 17, 19, 26, 23]
Output: [24, 10, 20, 8]

Decomposition example with 16 elements

Input: [26, 40, 42, 57, 15, 31, 5, 32, 11, 4, 24, 28, 51, 54, 12, 22]
Output: {
"List 1": [26, 40, 42, 57, 15, 31, 5, 32],
"List 2": [11, 4, 24, 28, 51, 54, 12, 22]
}

Solve-stage sorted audit example

Input Set 1 Sorted: [1, 3, 5, 8] **Input Set 2 Sorted:** [0, 1, 5, 7] **Proposed Intersection Sorted:** [1]
Rational: Input-2 Membership Audit: [0:N, 1:Y, 5:Y, 7:N]; Two-pointer matches: [1, 5]; Proposed Sorted: [1]; Missing Elements: [5]; False Positives: []; Duplicates: []; Expected Count: 2; Proposed Count: 1; Total Issues: 1.
Error: Missing Elements: 5; False Positives: None; Duplicates: None
Score: 98

Representative correction example

Previous Incorrect Result: [50, 108, 54, 59] **Critical Feedback:** Missing Elements: 2; False Positives: None; Duplicates: None
Corrected Output: [50, 108, 54, 2, 59]

D.2 SORTING

The sorting critique prompts jointly check multiset preservation and the global nondecreasing-order constraint. The solving stage performs frequency and run audits, while the aggregation stage further checks source-sequence frequencies, expected blocks, and global order. Tables 21 and 22 provide the prompt templates and few-shot examples, respectively.

Table 21: Sorting task prompt templates (runtime parameters are denoted by curly braces; few-shot examples are provided in Table 22).

Direct sorting prompt

<Instruction> Sort the following list of numbers in ascending order. Output only the sorted list of numbers, no additional text. </Instruction>
<*Examples*> Three representative sorting examples cover the active task scales. <*/Examples*>
Input: {input}

Two-way contiguous decomposition prompt

<Instruction> Split the following list of {size} numbers into 2 lists by preserving the original order. - "List 1" must contain the first {half} numbers. - "List 2" must contain the remaining {size-half} numbers. Only output the 2 lists in the following JSON format without any additional text or thoughts: { "List 1": [...], "List 2": [...] } </Instruction>
<*Example*> A shared split example block is embedded in `SPLIT_EXAMPLES_SORT`. <*/Example*>
Input: {input_list_str}

Solve-stage frequency-and-run audit prompt

<Instruction>You are a strict sorting verifier. Independently verify BOTH multiset preservation and global nondecreasing order. Do not trust the label "Candidate Sorted" and do not infer correctness from a visually plausible prefix. You MUST complete the Frequency-and-Run Audit before assigning a score. </Instruction>
<Approach>1. Length checksum. 2. Frequency audit for EVERY digit 0 through 9. 3. Candidate run audit using maximal consecutive runs. 4. Adjacent inversion audit over all boundaries. 5. Total issues = Frequency issues + Ordering issues. 6. Score = max(0, 100 - 2 * Total issues). 7. The final error line MUST contain "Ordering Error" whenever an inversion exists. </Approach>
Input length: {input_length} **Input:** {input_text} **Candidate length:** {candidate_length} **Candidate Sorted:** {candidate_text}
IMPORTANT: Output in the exact format below.
rational: Length check: ... Digit 0: ... . . . Digit 9: ... Candidate runs: ... Adjacent inversion scan: ... Ordering issues = ... Total issues = ... Score = ... error: None OR "Extra: ...; Missing: ...; Ordering Error: position ..." score: <integer>

Table 21: Sorting task prompt templates (continued).

Solve-stage correction prompt

<Instruction> Your task is to CORRECT the 'Previous Incorrect Result' based strictly on the provided 'Critical Feedback'. You are NOT solving from scratch.
CRITICAL FEEDBACK FORMAT: - Extra: d x k (Candidate has k too many of digit d) - Missing: d x k (Candidate lacks k of digit d) - Ordering Error (Candidate has the correct values but is not nondecreasing)
<Approach>1) Mentally calculate the final exact count needed for each digit by removing the 'Extra' counts and adding the 'Missing' counts from the Previous Incorrect Result. 2) Output the final list by simply generating the exact number of 0s, then 1s, then 2s, up to 9s. This guarantees perfect ascending order. IMPORTANT: Output ONLY the final corrected list in brackets. Do not output any intermediate lists in your reasoning. </Approach>
<Immediate Correction Task>Input: {input_set} Previous Incorrect Result: {current} Critical Feedback: {latest_feedback} </Immediate Correction Task>Corrected Output:

Aggregation prompt

<Instruction> Merge the following 2 sorted lists of length {length1} each, into one sorted list of length {length2} using a merge sort style approach. Only output the final merged list without any additional text or thoughts!: </Instruction>
<Approach>1. Compare the first element of both lists. 2. Append the smaller element to the merged list and move to the next element in the list from which the smaller element came. 3. Repeat steps 1 and 2 until one of the lists is empty. 4. Append the remaining elements of the non-empty list to the merged list. </Approach>
Merge the following two lists into one sorted list: 1: {input1} 2: {input2}
Merged list:

Aggregation block-and-order audit prompt

<Instruction>You are a strict merge-sort verifier. A valid Merged Result must preserve every occurrence from BOTH source lists and must satisfy global nondecreasing order after interleaving them. You MUST complete the Source-Frequency, Expected-Block, and Global-Order Audit before assigning a score. </Instruction>
<Approach>1. Source checksum: len(List 1), len(List 2), their sum, and len(Merged Result). 2. Source-frequency audit for digits 0–9. 3. Expected-block certificate from the combined counts. 4. Observed-run certificate over the full merged result. 5. Adjacent inversion audit over all boundaries. 6. Total issues = Frequency issues + Ordering issues. 7. Score = max(0, 100 - 2 * Total issues). </Approach>
List 1 length: {input1_length} **List 1:** {input1_text} **List 2 length:** {input2_length} **List 2:** {input2_text} **Combined expected length:** {combined_length} **Merged length:** {merged_length} **Merged Result:** {merged_text}
IMPORTANT: Output in the exact format below.
rational: Length check: ... Digit 0: ... Digit 1: ... Digit 2: ... Digit 3: ... Digit 4: ... Digit 5: ... Digit 6: ... Digit 7: ... Digit 8: ... Digit 9: ... Expected blocks: ... Observed runs: ... Adjacent inversion scan: ... Ordering issues = ... Total issues = ... Score = ... error: None OR "Extra: ...; Missing: ...; Ordering Error: position ..." score: <integer>

Aggregation correction prompt

<Instruction> You are refining a previously incorrect Merged Result using digit-frequency and ordering feedback (0-9). Do NOT merge from scratch. You MUST use the Counting method to fix the frequencies based on the Critical Feedback. IMPORTANT: Output ONLY the final list in brackets. Do not output any intermediate lists in your reasoning. </Instruction>
<Approach>1. Read the Critical Feedback. - "Extra" means you need to delete that many occurrences of the digit. - "Missing" means you need to add that many occurrences of the digit. - "Ordering Error" means retain the corrected frequencies and rewrite the result in nondecreasing order. 2. Mentally calculate the correct final frequencies for digits 0-9. 3. Output the final merged list by writing all required '0's, then all required '1's, etc. </Approach>
<Current Immediate Task>List 1 (sorted): {input1} List 2 (sorted): {input2} Previous Incorrect Result: {last_attempt} Critical Feedback: {latest_feedback} </Current Immediate Task>
Corrected Merged List:

Table 22: Few-shot examples embedded in the sorting task prompts (selected examples shown; some sequences are truncated).

Direct sorting example

Input: [6, 4, 5, 7, 5, 6, 9, 7, 6, 9, 4, 6, 9, 8, 1, 9]
Output: [1, 4, 4, 5, 5, 6, 6, 6, 6, 7, 7, 8, 9, 9, 9, 9]

Decomposition example

Input: [6, 4, 5, 7, 5, 6, 9, 7, 6, 9, 4, 6, 9, 8, 1, 9, 2, 4, 9, 0, 7, 6, 5, 6, 6, 2, 8, 3, 9, 5, 6, 1]
Output: {
"List 1": [6, 4, 5, 7, 5, 6, 9, 7, 6, 9, 4, 6, 9, 8, 1, 9],
"List 2": [2, 4, 9, 0, 7, 6, 5, 6, 6, 2, 8, 3, 9, 5, 6, 1]
}

Solve-stage audit example

Input length: 6 **Input:** [2, 0, 2, 1, 3, 1]
Candidate length: 6 **Candidate Sorted:** [0, 1, 2, 1, 2, 2]
rational:
Length check: Input 6; Candidate 6; Match.
Digit 0: Input 1. Candidate 1. Match.
Digit 1: Input 2. Candidate 2. Match.
Digit 2: Input 2. Candidate 3. Extra 1.
Digit 3: Input 1. Candidate 0. Missing 1.
Candidate runs: 0x1 — 1x1 — 2x1 — 1x1 — 2x2. Digit 1 repeats in a later run and run labels decrease $2 \rightarrow 1$.
Adjacent inversion scan: position 3 ($2 > 1$). Ordering issues = 1.
Total issues = 1 + 1 + 1 = 3. Score = 100 - 6 = 94.
error: Extra: 2x1; Missing: 3x1; Ordering Error: position 3 ($2 > 1$) **score:** 94

Aggregation audit example

List 1: [0, 1, 1, 4] **List 2:** [0, 2, 3, 4]
Merged Result: [0, 0, 1, 2, 1, 3, 4, 4]
rational: Expected blocks: 0x2 [1-2] — 1x2 [3-4] — 2x1 [5] — 3x1 [6] — 4x2 [7-8]. Observed runs: 0x2 — 1x1 — 2x1 — 1x1 — 3x1 — 4x2. Adjacent inversion scan: position 4 ($2 > 1$). Total issues = 0 + 1 = 1. Score = 98.
error: Ordering Error: position 4 ($2 > 1$) **score:** 98

D.3 KEYWORD COUNTING

Keyword counting uses a concise JSON-only prompt for direct solving. The decomposition stage recursively partitions the text into two contiguous sentence groups; the solving-stage critique reconstructs keyword occurrences sentence by sentence; and the aggregation-stage critique verifies the sum of the two subresults entry by entry. Tables 23 and 24 provide the prompt templates and few-shot examples, respectively.

Table 23: Keyword counting prompt templates (runtime parameters are denoted by curly braces; few-shot examples are provided in Table 24).

Direct JSON counting prompt

<Instruction> Count the frequency of how many times each country is explicitly named in the input text. Output only the frequency of each country that appears at least once in the following JSON format; make sure to keep the same spelling and output no additional text: { "country1": frequency1, "country2": frequency2 } </Instruction>
<Examples> Three complete counting demonstrations illustrate the required JSON-only output. *</Examples>*
Input: {input} Output:

Sentence-group decomposition prompt

<Instruction> The input text contains {total_sentences} sentences (delimited by periods '.'). Your task is to split the input text into exactly 2 parts with an EQUAL NUMBER of sentences. - "Part 1" MUST contain exactly the first {half_sentences} sentences. - "Part 2" MUST contain exactly the remaining {remainder} sentences. Ensure you split perfectly at the sentence boundaries (periods '.') without altering, adding, or omitting any words or punctuation. Only output the 2 parts in the following JSON format without any additional text or thoughts: { "Part 1": "...", "Part 2": "..." } </Instruction>
<Example> A sentence-level example illustrates how to divide the text into two contiguous groups with equal numbers of sentences. *</Example>*
Input: {input_text}

Solve-stage occurrence audit prompt

<Instruction> You are a strict country-count verifier. Independently recount every explicitly named country in the numbered input sentences, then verify the Candidate Dictionary. You MUST use the Occurrence Ledger method. </Instruction>
<Approach>1. Read each numbered sentence carefully. For sentence Si, write one ledger line listing every explicitly named country in that sentence in reading order. 2. Combine all ledger lines into an audited dictionary with exact country frequencies. 3. For EVERY country found by the audit OR present in the Candidate Dictionary, write exactly one comparison line: "Country [Name]: Text has [X]. Candidate has [Y]. [Match / Extra K / Missing K]." 4. Check both directions: a country omitted by the candidate is Missing; a candidate key absent from the audit is Extra. 5. Total issues = (sum of all Extra K) + (sum of all Missing K). 6. Score = max(0, 100 - 10 * Total issues). </Approach>
Numbered Input Sentences: {input} Candidate Dictionary: {current}
IMPORTANT: Output in the exact format below.
rational: Occurrence ledger: S1: ... S2: ... Audited dictionary: {...} Ledger occurrences = ... Audited total = ... [Match / Recount required]. Country ... Country ... Total issues = ... Score = ... error: None OR "Extra: ...; Missing: ..." score: <integer>

Table 23: Keyword counting prompt templates (continued).

Solve-stage correction prompt

<Instruction> Your task is to CORRECT the 'Previous Incorrect Dictionary' based strictly on the 'Critical Feedback'. </Instruction>
<Approach>1. Read the Critical Feedback. - "Extra: [Country] xK" means you must subtract K from that country's value (or remove it if it reaches 0). - "Missing: [Country] xK" means you must add K to that country's value (or insert it if it doesn't exist). 2. Output ONLY the corrected JSON dictionary. No other text. </Approach>
Previous Incorrect Dictionary: {current} Critical Feedback: {latest_feedback} Corrected Output:

Aggregation prompt

<Instruction> Combine the following 2 dictionaries, each containing the frequency of countries in a text, into a single dictionary. Simply add the frequencies together for each country and if a country is not present in one of the dictionaries, add it to the final dictionary with the frequency from the other dictionary. Only output the final merged dictionary without any additional text or thoughts! </Instruction>
<Approach>To combine the 2 dictionaries into single one, follow these steps: 1. Create a new dictionary to store the combined frequencies. 2. Iterate through the keys of the first dictionary and add the frequency of each country to the new dictionary. 3. Iterate through the keys of the second dictionary and add the frequency of each country to the new dictionary and if it is already present, add the frequency to the existing value. </Approach>
Combine the following 2 dictionaries into a single dictionary: {input1}
{input2}
Combined Output:

Aggregation arithmetic audit prompt

<Instruction> You are a math verifier. Verify if the 'Merged Dictionary' is the exact summation of 'Dictionary 1' and 'Dictionary 2'. You MUST use the Line-by-Line Tally method. </Instruction>
<Approach>For EVERY country present in Dict 1, Dict 2, or the Merged Dictionary, write a line: 1. "Country [Name]: Dict1 [X] + Dict2 [Y] = [Sum]. Merged [Z]. [Match / Extra K / Missing K]." 2. Total issues = (sum of Extra K) + (sum of Missing K). 3. Score = max(0, 100 - 10 * Total issues). </Approach>
Dictionary 1: {input1} Dictionary 2: {input2} Merged Dictionary: {current}
IMPORTANT: Output in the exact format below.
rational: Country ... Country ... Total issues = ... Score = ... error: None OR "Extra: ...; Missing: ..."
score: <integer>

Aggregation correction prompt

<Instruction> You are refining an incorrectly Merged Dictionary based on the Critical Feedback. </Instruction>
<Approach>Fix the math by applying the "Extra" and "Missing" feedback to the 'Previous Incorrect Dictionary'. Output ONLY the final corrected JSON dictionary. </Approach>
Previous Incorrect Dictionary: {last_attempt} Critical Feedback: {latest_feedback} Corrected Output:

Table 24: Few-shot examples embedded in the keyword counting prompts (selected examples shown).

Direct counting example

Input: Alexandra boarded the first flight of her grand journey, starting from Canada. With a globe-trotting itinerary in hand, she was filled with excitement. Her first stop was Mexico, where she marveled at the Mayan ruins. From there, she explored the rainforests of Brazil and danced the tango in Argentina.
Output: {"Canada": 1, "Mexico": 1, "Brazil": 1, "Argentina": 1}

Occurrence-ledger audit example

Numbered Input Sentences:
S1: She visited Mexico and Canada.
S2: She returned to Mexico before flying to Japan.
Candidate Dictionary: {"Mexico": 3, "Canada": 1, "Brazil": 1}
rational:
Occurrence ledger:
S1: Mexico, Canada
S2: Mexico, Japan
Audited dictionary: {"Canada": 1, "Japan": 1, "Mexico": 2}
Country Mexico: Text has 2. Candidate has 3. Extra 1.
Country Canada: Text has 1. Candidate has 1. Match.
Country Japan: Text has 1. Candidate has 0. Missing 1.
Country Brazil: Text has 0. Candidate has 1. Extra 1.
Total issues = 1 + 1 + 1 = 3. Score = 100 - 30 = 70.
error: Extra: Mexico x1, Brazil x1; Missing: Japan x1 **score:** 70

Line-by-line aggregation audit example

Dictionary 1: {"Peru": 2, "Chile": 1} **Dictionary 2:** {"Peru": 1, "Germany": 2}
Merged Dictionary: {"Peru": 4, "Chile": 1, "Germany": 1}
rational:
Country Peru: Dict1 2 + Dict2 1 = 3. Merged 4. Extra 1.
Country Chile: Dict1 1 + Dict2 0 = 1. Merged 1. Match.
Country Germany: Dict1 0 + Dict2 2 = 2. Merged 1. Missing 1.
Total issues = 1 + 1 = 2. Score = 100 - 20 = 80.
error: Extra: Peru x1; Missing: Germany x1 **score:** 80