STREAMLINED REFLECTIVE EVOLUTION FOR TASK-ADAPTIVE SELF-REFINEMENT PIPELINES

Anonymous authors
Paper under double-blind review

ABSTRACT

Reflective prompt optimization improves large language model (LLM) systems without updating model weights, but fixed architectures constrain how self-refinement is organized. We introduce Workflow-Designing Agents (WDA), a framework for streamlined reflective evolution of task-adaptive self-refinement pipelines. Starting from a minimal prompt, WDA jointly evolves stage instructions and their sequential structure. During evolution, we find that repeated revisions can accumulate redundant instructions in a single prompt. In WDA, we propose to address this problem with SPLIT, which redistributes these instructions across specialized stages. Three-example reflection and local screening guide selective search, while calibration scores guide Pareto admission and rollback of unhelpful trailing updates. The resulting pipelines are task-adaptive: their instructions and depth are learned from task data, then fixed for all test inputs within that task. We evaluate WDA on five benchmarks spanning knowledge, mathematical reasoning, multi-hop question answering, and instruction following. On Qwen3.5-9B, WDA achieves an average score of 51.24%, improving over the initial solver by 8.63 percentage points and the variant without SPLIT by 2.60 points. On GPT-4.1-mini, it achieves 49.00%, with corresponding gains of 5.62 and 3.69 points. These results support task-adaptive self-refinement as a complementary direction to broader agentic workflow search.

1 INTRODUCTION

LLMs are increasingly used for knowledge-intensive question answering and reasoning (Lewis et al., 2020; Wei et al., 2022). Their performance depends on instructions and examples, including in mathematical and medical tasks (Wei et al., 2022; Nori et al., 2023). Designing effective prompts can require substantial manual effort (Pryzant et al., 2023). Automatic prompt optimizers instead use task feedback to search for better instructions (Pryzant et al., 2023; Yang et al., 2024). The cost of this search motivates sample-efficient prompt adaptation (Agrawal et al., 2026).

One widely used remedy is self-refinement: the model produces an answer, critiques it, and revises it, repeating the loop at inference time without any weight update (Madaan et al., 2023). The recipe is attractive because it needs no training data and no second model. It is also fragile. When the critique comes from the model itself and no external signal is available, self-correction frequently fails to improve reasoning and can make accuracy worse than the first attempt (Huang et al., 2024). These findings motivate examining two commonly hand-specified choices: the number of refinement rounds and the critique instruction. A fixed depth and a generic request to find mistakes may not provide the task-specific checks needed to improve the initial answer.

A separate line of work makes prompts themselves learnable. Automatic prompt optimizers search over natural-language instructions using calibration data, and reflective optimizers use the system's own execution traces to diagnose failures and propose targeted rewrites. Genetic-Pareto (GEPA) exemplifies this approach: it collects trajectories and textual feedback, has an LLM propose reflective mutations, and preserves complementary candidates by instance-level Pareto selection (Agrawal et al., 2026). It can therefore optimize critique instructions, but takes the system architecture as given. Broader approaches such as Automated Design of Agentic Systems (ADAS) and Automating Agentic Workflow Generation (AFlow) search agentic designs or executable workflows (Hu et al.,

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.

2025; Zhang et al., 2025). Our focus is narrower: learning the depth and division of responsibilities within sequential self-refinement, which can itself serve as a component of a larger workflow.

We propose WDA, which streamlines GEPA-inspired reflective evolution to learn task-adaptive self-refinement pipelines. Starting from one minimal, task-agnostic instruction, it samples a feasible operator and compatible configuration. REPLACE specializes an instruction, ADD appends a refinement stage, and SPLIT decomposes an overloaded prompt into consecutive stages. REPLACE and ADD proposals must improve a three-example batch before calibration evaluation; SPLIT proposals compete on a shared batch. Pareto admission retains non-dominated candidates, and calibration-guided rollback discards trailing updates without a calibration gain. Local screening and rollback keep evolution selective. Task data determine what each stage checks and how many stages to retain; the selected pipeline remains fixed during test inference.

Our contributions are threefold. First, we formulate task-adaptive self-refinement: reflection learns stage instructions, and calibration guides pipeline structure and depth, instead of prescribing a fixed refine loop. Second, we introduce SPLIT to redistribute instructions from an evolved prompt across specialized sequential stages. This is motivated by the redundant instructions that can accumulate during evolution and the need for more focused refinement stages. Third, we evaluate WDA on five benchmarks with a proprietary and an open-weight backbone, comparing against direct inference, fixed-depth self-refinement, fixed-architecture prompt optimizers, and preference-based fine-tuning.

2 RELATED WORK

2.1 LLM RECURSIVE SELF-IMPROVEMENT

We use recursive self-improvement here to refer to repeated LLM-guided revisions of prompts or agentic programs using execution feedback, rather than autonomous updates to model weights. Early prompt optimizers use textual gradients, black-box search, or evolutionary operators (Pryzant et al., 2023; Yang et al., 2024; Guo et al., 2024; Fernando et al., 2024). GEPA uses trajectories and textual feedback to propose reflective mutations and preserves complementary candidates through instance-level Pareto selection (Agrawal et al., 2026). Consensus-Evolve (C-Evolve) instead evolves prompt groups using each member's contribution to majority-vote consensus (Li et al., 2026), while Multi-Agent System Prompt Optimization via Bandits (MASPOB) optimizes role-specific prompts with topology-aware dependency modeling in a fixed multi-agent workflow (Hong et al., 2026). Mixture-of-Prompts (MoP) routes inputs to specialized prompts associated with semantic regions (Wang et al., 2024); combining such routing with region-specific WDA pipelines could make refinement instructions and depth input-adaptive.

Beyond prompt optimization, ADAS uses a meta-agent to propose and evaluate agentic systems represented in code (Hu et al., 2025). AFlow searches code-represented workflows with Monte Carlo Tree Search, execution feedback, and reusable operators such as Ensemble and Review & Revise (Zhang et al., 2025). These approaches can change workflow structure; they should not be characterized as fixed-architecture prompt optimizers. WDA targets a more focused search space: the instructions, depth, and stage responsibilities of sequential self-refinement. Self-refinement is a common workflow building block, explicitly represented by AFlow's Review & Revise operator. WDA could therefore supply an optimized refinement component to a broader workflow optimizer, although we do not evaluate such an integration here.

2.2 SELF-REFINEMENT

Self-Refine iterates answer generation, feedback, and revision with one model and no weight updates (Madaan et al., 2023). Without an external verifier, however, intrinsic self-correction can fail to improve reasoning or reduce accuracy (Huang et al., 2024). Fixed-depth pipelines with generic critique instructions may spend unnecessary computation or miss task-specific errors. WDA retains sequential refinement but learns its depth through calibration and its instructions through reflection on failed traces.

Relatedly, multi-agent systems elicit diverse candidate solutions and let agents critique or aggregate one another's outputs. Multiagent debate exchanges independently generated reasoning traces (Du et al., 2024), role-based systems such as MedAgents organize domain experts into multi-round delib-

eration (Tang et al., 2024), and Mixture-of-Agents uses a layered proposer–aggregator architecture (Wang et al., 2025). Controlled comparisons find that debate does not reliably beat self-consistency (Wang et al., 2023) or simple ensembles without careful configuration (Smit et al., 2024). These systems differ from ours in both structure and provenance: their modules answer in parallel and are combined by a fixed protocol, whereas our modules form an ordered refinement chain, and their composition is specified in advance rather than discovered.

3 PROBLEM STATEMENT

We study how to learn a task-adaptive, multi-prompt LLM system from task data, starting from a single simple prompt and keeping the learned pipeline fixed at test time. Following prior work (Agrawal et al., 2026), we represent the system as $\Phi = (M, C, \mathcal{X}, \mathcal{Y})$, where $M = \langle M_1, \ldots, M_{|M|} \rangle$ is a sequence of LLM modules, C specifies the control logic, and $\mathcal{X}$ and $\mathcal{Y}$ denote the input and output schemas, respectively. Each module is defined as $M_i = (\pi_i, \theta_i, \mathcal{X}_i, \mathcal{Y}_i)$, where π_i is its prompt, θ_i denotes its model weights, and $\mathcal{X}_i$ and $\mathcal{Y}_i$ specify its input and output schemas. We adopt sequential control logic: the first module receives the input question, each subsequent module takes the original question and the preceding module's output as input, and the last module produces the final answer.

Initially, the system contains only one module with a simple prompt. During calibration, the system applies REPLACE, ADD, and SPLIT, automatically determining both the number of modules and their instructions to obtain a multi-prompt system. We denote the module prompts by $\Pi_\Phi = \langle \pi_1, \ldots, \pi_{|M|} \rangle$ and the corresponding model weights by $\Theta_\Phi = \langle \theta_1, \ldots, \theta_{|M|} \rangle$. The model weights remain fixed; we optimize the module structure and prompts contained in the complete system Φ.

The objective is to maximize expected performance on task instances $(x, m) \sim \mathcal{T}$, where $\mathcal{T}$ is the task distribution, $x \in \mathcal{X}$ is an input, and $\mathcal{M}$ is the space of evaluation metadata m, such as reference answers, scoring rubrics, or unit tests. Given the system output $y = \Phi(x; \Pi_\Phi, \Theta_\Phi)$, abbreviated as $\Phi(x)$, a task-specific metric $\mu : \mathcal{Y} \times \mathcal{M} \to [0, 1]$ measures its quality with respect to m. The optimization problem is

$$\Phi^{\text{opt}} = \arg\max_{\Phi} \mathbb{E}_{(x,m)\sim\mathcal{T}} \left[\mu(\Phi(x), m) \right]. \tag{1}$$

Here $\mathbb{E}$ denotes expectation and $\arg\max$ selects a maximizing system. In practice, calibration data are used to estimate this objective and guide the evolution from the initial single-prompt system to a system of sequential prompt modules.

4 METHOD

WDA learns task-adaptive pipelines through minimal initialization (Section 4.1), reflective structure and prompt evolution (Section 4.2), and sequential test inference (Section 4.3). Its inputs are an initial system $\Phi^{(0)}$, training set $\mathcal{D}_{\text{train}}$ for reflection, calibration set $\mathcal{D}_{\text{cal}}$ for candidate evaluation, task metric μ, reflection LLM R, and evolution budget B. It evolves module structure and prompts while keeping weights Θ_Φ fixed. Figure 1 summarizes the workflow.

4.1 INITIALIZATION

We initialize $\Phi^{(0)} = (M^{(0)}, C^{(0)}, \mathcal{X}, \mathcal{Y})$ with one module, $M^{(0)} = \langle M_1^{(0)} \rangle$, where $M_1^{(0)} = (\pi_1^{(0)}, \theta_1, \mathcal{X}, \mathcal{Y})$. Control $C^{(0)}$ passes input x to this module and returns its answer. The sole prompt $\pi_1^{(0)}$ asks it to solve the problem, without task-specific expertise or a prescribed decomposition; hence $\Pi_{\Phi^{(0)}} = \langle \pi_1^{(0)} \rangle$.

We score this system on calibration data using μ. Modules are introduced incrementally, and calibration-guided rollback retains useful changes rather than expanding unconditionally. This initialization makes refinement optional: tasks that do not benefit from an additional stage can retain a compact solver. The search therefore allocates complexity in response to observed errors instead of committing to a multi-stage architecture before evaluation. Repeated revisions can introduce redundant instructions, motivating SPLIT to distribute responsibilities across focused stages.

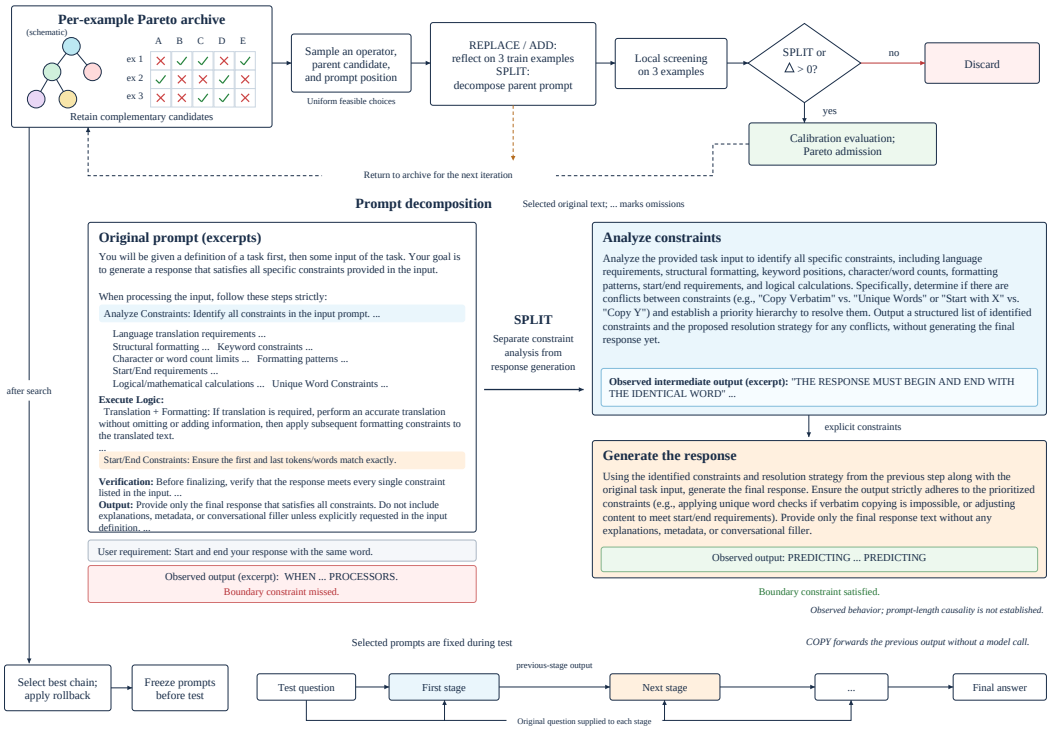


Figure 1: WDA: reflective evolution with local screening, Pareto admission, and fixed test-time execution. The example illustrates SPLIT separating constraint analysis from response generation; these excerpts do not establish prompt-length causality. Each stage receives the question and preceding output; COPY forwards that output without a model call.

4.2 INCREMENTAL PROMPT EVOLUTION

Inspired by GEPA's instance-level feedback (Agrawal et al., 2026), we store admitted systems as $\mathcal{P}[k] = \Phi_k$, where k is a candidate identifier (ID) and $\pi_{k,i}$ is its i-th prompt. Let $\mathcal{D}_{\text{cal}} = \{(x_j, m_j)\}_{j=1}^{N_{\text{cal}}}$ contain N_{cal} examples. The score matrix has entries $S_{k,j} = \mu(\Phi_k(x_j), m_j)$; $S_{k,:}$ denotes candidate k's score vector. Let A contain the current Pareto-front candidate IDs. For any candidate-ID set I, $\ell \succ k$ means that ℓ dominates k, and the non-dominated (ND) operator keeps candidates dominated by none:

$$\begin{aligned} \ell \succ k \iff (\forall j,\ S_{\ell,j} \geq S_{k,j}) \wedge (\exists j,\ S_{\ell,j} > S_{k,j}), \\ \text{ND}(I; S) = \{k \in I : \nexists \ell \in I \text{ with } \ell \succ k\}. \end{aligned} \tag{2}$$

This preserves complementary correctness patterns rather than retaining only the highest average score. Unlike GEPA's frequency-weighted selection, our implementation samples uniformly from operator-compatible Pareto-front candidates.

We first sample an operator, then a compatible parent and target. Let $\mathcal{I}_o(k)$ be the eligible prompt positions in Φ_k for operator o, let $A_o = \{k \in A : \mathcal{I}_o(k) \neq \emptyset\}$ be the compatible parent IDs, and let $\mathcal{O}(A) = \{o : A_o \neq \emptyset\}$ be the available operators. REPLACE rewrites an active prompt; ADD appends a module when capacity permits; SPLIT replaces an eligible evolved prompt with two consecutive modules. The unchanged initial prompt is not eligible for SPLIT: it must first be revised by REPLACE. These restrictions and the module limit $K_{\max}$ determine $\mathcal{I}_o(k)$. At search iteration t, we sample an operator o_t, parent ID k_t, and position i_t. The uniform distribution $\text{Unif}(V)$ gives each element of a finite set V probability $1/|V|$:

$$o_t \sim \text{Unif}(\mathcal{O}(A)), \qquad k_t \sim \text{Unif}(A_{o_t}), \qquad i_t \sim \text{Unif}(\mathcal{I}_{o_t}(k_t)).$$

For ADD, $\mathcal{I}_{o_t}(k_t)$ contains only the next insertion position, and $\pi_{k_t,i_t} = \emptyset$ denotes the absence of an instruction at that position.

Reflection and local screening. The reflection model R is the same configured backbone as the solver, with fixed weights. For REPLACE and ADD, it receives the selected instruction, the chosen operator, and three uniformly sampled training examples $\mathcal{B}_t$, including their questions, intermediate reasoning traces τ_t, and reference-answer feedback F_t. It returns a diagnosis of reusable errors and a prompt proposal r_t; the instruction requests general strategies rather than copying example-specific answers. For SPLIT, the LLM sees only the parent instruction and decomposition requirements, not the sampled examples or reference answers, and returns two complementary prompts. Write $U_t = (\mathcal{B}_t, \tau_t, F_t)$ for REPLACE/ADD and $U_t = \emptyset$ for SPLIT. No reference answers are provided at test time. REPLACE and ADD proposals must improve their three-example batch. For SPLIT, valid decompositions are ranked on a shared batch, and the highest-scoring proposal proceeds to calibration even without a local improvement. We denote proposal generation and, for SPLIT, this local proposal selection by $\mathrm{Reflect}$; it returns r_t or skips the iteration if no valid proposal is obtained. The batch is used to score SPLIT proposals, not as input to R. Applying operator o_t produces $\widetilde{\Phi}_t$; Δ_t is its mean batch-score gain and g_t indicates whether it passes local screening:

$$\begin{aligned} r_t &= \mathrm{Reflect}(R, \Phi_{k_t}, i_t, o_t, U_t, \mathcal{B}_t), \\ \widetilde{\Phi}_t &= o_t(\Phi_{k_t}, i_t, r_t), \\ \Delta_t &= \frac{1}{|\mathcal{B}_t|} \sum_{(x,m)\in\mathcal{B}_t} \big[\mu(\widetilde{\Phi}_t(x), m) - \mu(\Phi_{k_t}(x), m)\big], \\ g_t &= \mathbf{1}[o_t = \textsc{Split} \ \vee \ \Delta_t > 0]\,. \end{aligned} \tag{3}$$

Here $\mathbf{1}[\cdot]$ equals 1 when its condition holds and 0 otherwise. Candidates with $g_t = 1$ are evaluated on the full calibration set. Their configurations and score vectors are appended to $\mathcal{P}$ and S only if they enter the Pareto front; candidates they dominate are removed from A. Duplicate configurations are rejected. The archive has capacity $C_{\max}$. The $\mathrm{UpdateFront}$ operation first applies ND, then retains the best mean-scoring member and prioritizes complementary instance coverage when pruning to capacity. Historical records remain available for rollback; $H[k]$ stores candidate k's parent ID. They are not all eligible for sampling.

For calibration-guided rollback, following H gives the lineage $\mathcal{L}(\Phi) = \langle \Phi^{(0)}, \ldots, \Phi^{(L)} = \Phi \rangle$, where L counts ancestral updates and s indexes checkpoints, not global iterations. Let J_{cal} be the mean calibration score and s^* the earliest best checkpoint. Rollback returns that checkpoint:

$$\begin{aligned} J_{\mathrm{cal}}(\Phi) &= \frac{1}{N_{\mathrm{cal}}} \sum_{j=1}^{N_{\mathrm{cal}}} \mu(\Phi(x_j), m_j), \\ s^*(\Phi) &= \min \arg\max_{0\le s\le L} J_{\mathrm{cal}}(\Phi^{(s)}), \\ \mathrm{Rollback}(\Phi, H) &= \Phi^{(s^*(\Phi))}. \end{aligned} \tag{4}$$

This discards trailing updates without a calibration gain, but need not reduce module count if they only rewrite prompts. In Algorithm 1, $\mathrm{Evaluate}$ returns per-example scores; S' and A' are temporary scores and front IDs. Iteration t also serves as a candidate ID, so rejected iterations leave gaps. We omit iteration subscripts inside the loop. The returned Φ^* is the selected system, not a guarantee of the ideal optimum Φ^{opt}.

4.3 TEST INFERENCE

At test time, the calibration-selected system Φ and its prompts are fixed: no reflection, reference-answer feedback, or further search is used. Its modules execute sequentially according to C. Let $K = |M|$ count active LLM modules, excluding pass-through COPY slots, and let h_i denote the output of module i:

$$\begin{aligned} h_1 &= M_1(x; \pi_1, \theta_1), \\ h_i &= M_i(x, h_{i-1}; \pi_i, \theta_i), \quad i = 2, \ldots, K, \qquad y = h_K. \end{aligned} \tag{5}$$

Each active module is one inference step (one LLM call). It can refine, verify, or extend the preceding result. A COPY slot forwards that output without an LLM call. The learned chain is reused

Algorithm 1 WDA

Require: Initial system $\Phi^{(0)}$, $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{cal}}$, metric μ, reflection model R, budget B
Require: Module limit $K_{\max}$, archive capacity $C_{\max}$; B counts search iterations
Ensure: Optimized system Φ^*
Notation: $\mathcal{P}[k] = \Phi_k$: stored system; S: calibration scores; A: front IDs; H: parent map. $\mathcal{O}(A)$: available operators; A_o: compatible parent IDs; $\mathcal{I}_o(k)$: eligible prompt positions. Uniform sampling gives every eligible choice equal probability.

$\mathcal{P}[0] \leftarrow \Phi^{(0)}$; $S_{0,:} \leftarrow \textsc{Evaluate}(\Phi^{(0)}, \mathcal{D}_{\text{cal}}, \mu)$
$A \leftarrow \{0\}$; $H[0] \leftarrow \textsc{None}$ ▷ Initial front; no parent
for $t = 1, \ldots, B$ **do**
  Sample o uniformly from $\mathcal{O}(A)$, then k uniformly from A_o
  Sample i uniformly from $\mathcal{I}_o(k)$ ▷ Eligible prompt position
  $\mathcal{B} \leftarrow$ three examples sampled from $\mathcal{D}_{\text{train}}$
  Collect traces τ of Φ_k and reference-answer feedback F on $\mathcal{B}$
  $U \leftarrow \emptyset$ for Split; otherwise $U \leftarrow (\mathcal{B}, \tau, F)$
  $r \leftarrow \textsc{Reflect}(R, \Phi_k, i, o, U, \mathcal{B})$; skip if no valid proposal
  $\widetilde{\Phi} \leftarrow o(\Phi_k, i, r)$
  Compute Δ and local screening indicator g using Eq. 3
  if $g = 1$ and $\widetilde{\Phi}$ is not a duplicate **then**
    $\widetilde{s} \leftarrow \textsc{Evaluate}(\widetilde{\Phi}, \mathcal{D}_{\text{cal}}, \mu)$
    $S' \leftarrow S$; $S'_{t,:} \leftarrow \widetilde{s}$ ▷ Candidate's calibration scores
    $A' \leftarrow \textsc{UpdateFront}(A \cup \{t\}, S', C_{\max})$
    if $t \in A'$ **then** ▷ Admit only Pareto-front candidates
      $\mathcal{P}[t] \leftarrow \widetilde{\Phi}$; $S \leftarrow S'$; $H[t] \leftarrow k$; $A \leftarrow A'$
    end if
  end if
end for
$\widehat{k} \leftarrow \arg\max_{k \in A} J_{\text{cal}}(\Phi_k)$ ▷ Best mean-score candidate
$\Phi^* \leftarrow \textsc{Rollback}(\Phi_{\widehat{k}}, H)$ using Eq. 4
return Φ^*

for all test inputs in a task, with no online prompt evolution. Consequently, the inference procedure requires neither training examples nor reference-answer feedback. Task-adaptive thus refers to learning instructions and depth for a task, not adapting to individual test inputs. Input-adaptive stopping remains an extension discussed in Section 5.7.

5 Experiments

5.1 Research questions

We organize our experiments around three research questions (RQs). **RQ1:** Does learning the refinement structure—how many stages and what each one checks—improve task performance over fixed-depth self-refinement and fixed-architecture prompt optimization? **RQ2:** How does Split affect the number of inference calls of the discovered pipelines? **RQ3:** How does test performance vary across inference steps in pipelines discovered with and without Split?

5.2 Datasets

We evaluate on five benchmarks that cover complementary capabilities and remain unsaturated for the two tested backbones under our evaluation setup. The initial-solver scores in Tables 1 and 2 indicate substantial headroom for improvement. **SuperGPQA** evaluates graduate-level knowledge and reasoning across a broad range of academic disciplines (M-A-P et al., 2025). **LiveBench Math** contains challenging and contamination-resistant mathematical problems drawn from recently released sources (White et al., 2025). **BIG-Bench Extra Hard (BBEH)** evaluates diverse reasoning skills using substantially more difficult successors to BIG-Bench Hard tasks (Suzgun et al., 2023; Kazemi et al., 2025). **HotpotQA** tests multi-hop question answering over multiple supporting documents

Table 1: Test performance (%) of Qwen3.5-9B across five benchmarks, including the DPO comparison. Avg. is the unweighted mean of the five displayed benchmark scores. Bold indicates the highest score in each column.

Method	SuperGPQA	LiveBench Math	IFBench	BBEH	HotpotQA	Avg.
Initial solver	53.00	57.38	38.67	23.33	40.67	42.61
MIPROv2	51.33	51.64	31.33	22.00	49.33	41.13
GEPA	51.67	59.84	41.00	23.33	51.00	45.37
Self-Refine	54.33	65.57	**47.67**	25.67	39.00	46.45
DPO	50.00	57.38	40.33	21.67	40.00	41.88
WDA (no SPLIT)	53.00	67.21	42.33	29.33	51.33	48.64
WDA (SPLIT)	**56.00**	**68.85**	**47.67**	**31.67**	**52.00**	**51.24**

(Yang et al., 2018). Finally, **IFBench** evaluates generalization to diverse, verifiable instruction-following constraints (Pyatkin et al., 2025). We use each benchmark's official evaluator and report its default task score, with higher values indicating better performance.

5.3 MODELS

We use `gpt-4.1-mini` (OpenAI, 2025) and the open-weight `Qwen3.5-9B` (Qwen Team, 2026). WDA uses the same backbone for reflection and solving. Prompt-based methods keep weights fixed; Direct Preference Optimization (DPO) (Rafailov et al., 2023) provides a weight-updating comparison for Qwen3.5-9B.

5.4 BASELINES

We compare our method with the initial solver, Self-Refine, two prompt optimizers, and, for Qwen3.5-9B, DPO. The **initial solver** uses the task prompt without optimization or additional modules. **Self-Refine** applies a fixed generate–critique–revise workflow (Madaan et al., 2023). **Multi-prompt Instruction Proposal Optimizer version 2 (MIPROv2)** optimizes instructions and demonstrations for a fixed program (Opsahl-Ong et al., 2024). **GEPA** reflectively evolves prompts using execution feedback and instance-level Pareto selection (Agrawal et al., 2026). **DPO** updates model weights using preference optimization. We also compare with **WDA without SPLIT**, which disables the prompt-decomposition operator.

5.5 EXPERIMENTAL SETUP

For the Qwen3.5-9B results reported below, WDA uses a budget of at most 20 search iterations, with a minimum of 10 iterations and early stopping after five iterations without improvement. The recorded WDA runs complete 10–20 iterations. Solver and reflection outputs are capped at 4,096 tokens per call; task evaluation uses temperature zero, and reflection uses temperature 0.7. The recorded runs use a train/calibration/test ratio of 1:2:2, with at most 300 test examples per benchmark. Each reflection candidate uses three sampled training examples. The recorded MIPROv2 baseline uses instruction-only optimization with 20 trials and no few-shot demonstrations. Optimizer iterations do not imply equal token or model-call budgets. We select the final WDA chain using calibration performance and apply the rollback rule in Eq. 4; test scores are used only for reporting.

5.6 MAIN RESULTS

Tables 1 and 2 report Qwen3.5-9B and GPT-4.1-mini results, respectively, across the same five benchmarks. We report the unweighted arithmetic average (Avg.) of the five displayed dataset scores for each method.

Learning what to refine (RQ1). WDA with SPLIT improves over the initial solver on every benchmark for both backbones, with average gains of 8.63 and 5.62 percentage points for Qwen3.5-9B and GPT-4.1-mini, respectively. These gains are consistent with the value of task-specific re-

Table 2: Test performance (%) of GPT-4.1-mini across five benchmarks. Avg. is the unweighted mean of the five displayed benchmark scores. Bold indicates the highest score in each column.

Method	SuperGPQA	LiveBench Math	IFBench	BBEH	HotpotQA	Avg.
Initial solver	49.67	56.56	37.33	28.33	45.00	43.38
MIPROv2	52.67	54.10	41.33	28.00	47.33	44.69
GEPA	51.00	57.38	**42.67**	22.33	51.00	44.88
Self-Refine	49.67	59.02	42.33	27.00	45.00	44.60
WDA (no SPLIT)	50.33	58.20	38.67	29.00	50.33	45.31
WDA (SPLIT)	**56.00**	**60.66**	42.33	**34.33**	**51.67**	**49.00**

finement: reflection turns observed failures into reusable checks, giving later stages more specific guidance than a generic request to reconsider an answer. The resulting instructions carry this feedback into test-time inference without reference answers or weight updates. Pareto admission uses calibration scores beyond the small reflection batch.

Learning how to distribute refinement. Even without SPLIT, WDA exceeds GEPA and MIPROv2 in average score on both backbones; enabling SPLIT further improves every displayed benchmark score over the no-SPLIT variant. This pattern supports searching both instructions and their sequential organization. A plausible mechanism is that SPLIT gives each stage a more focused set of checks, while allowing the next stage to inspect and revise an explicit intermediate response. It thus offers an alternative to accumulating all refinement requirements in one prompt. The comparison supports SPLIT's usefulness within WDA, but does not isolate instruction specialization from changes in inference depth; Section 5.7 examines the associated call counts. The comparison therefore concerns the quality of the discovered pipelines, not a claim that applying SPLIT always saves computation. Whether an added stage is worthwhile depends on both the checks it performs and the backbone's ability to use its intermediate output.

Interpreting the baseline comparisons. WDA with SPLIT has the highest average score on both backbones, including against the Qwen-only DPO comparison, but is not uniformly best: on IFBench it ties Self-Refine for Qwen3.5-9B and falls slightly below GEPA for GPT-4.1-mini. Self-Refine's higher Qwen3.5-9B average than GEPA may partly reflect its explicit feedback–revision strategy and up to three cycles, or six additional LLM calls after the initial answer. This highlights the importance of how inference computation is organized, not only how a prompt is optimized. However, the runs do not isolate strategy from computation budget, so the results are descriptive rather than a matched-budget ranking.

5.7 ABLATION STUDY

Inference calls with and without SPLIT (RQ2). Table 3 compares the number of LLM calls required by the discovered pipelines for one input. For Qwen3.5-9B, the SPLIT-enabled pipelines use two calls on every benchmark, averaging 2.0 calls compared with 2.2 without SPLIT. For GPT-4.1-mini, they use three calls on every benchmark, compared with 2.2 calls on average without SPLIT. Together with the main results, these observations show that the effect of SPLIT on inference cost depends on the backbone: the higher average score accompanies fewer average calls for Qwen3.5-9B and more for GPT-4.1-mini. Call counts alone do not measure latency, token usage, or optimization cost. Variation in selected pipeline lengths across tasks and backbones also motivates input-adaptive inference: clustering inputs by their refinement needs or using calibrated confidence estimates could guide depth selection or early stopping to avoid unnecessary calls. Such input-level adaptation is a promising extension, rather than a demonstrated efficiency gain of the current fixed-per-task pipelines.

Performance across inference steps (RQ3). Figure 2 preserves the recorded intermediate-step trajectories. In the displayed Qwen3.5-9B runs, the two-step SPLIT pipelines outperform the three-step no-SPLIT pipelines on LiveBench Math (68.85% versus 67.21%), BBEH (31.67% versus 29.33%), and HotpotQA (52.00% versus 51.33%). Additional calls alone therefore do not explain these gains. Performance is not uniformly monotonic: the GPT-4.1-mini SPLIT-enabled curves for

Table 3: LLM calls per inference for pipelines found with and without SPLIT. Counts include the initial solver and exclude COPY slots. Avg. is the unweighted mean across benchmarks. The Qwen3.5-9B no-SPLIT pipeline on SuperGPQA retains only the initial solver.

	Qwen3.5-9B		GPT-4.1-mini	
Dataset	No SPLIT	SPLIT	No SPLIT	SPLIT
SuperGPQA	1	2	3	3
LiveBench Math	3	2	3	3
IFBench	1	2	1	3
BBEH	3	2	2	3
HotpotQA	3	2	2	3
Avg.	2.2	2.0	2.2	3.0

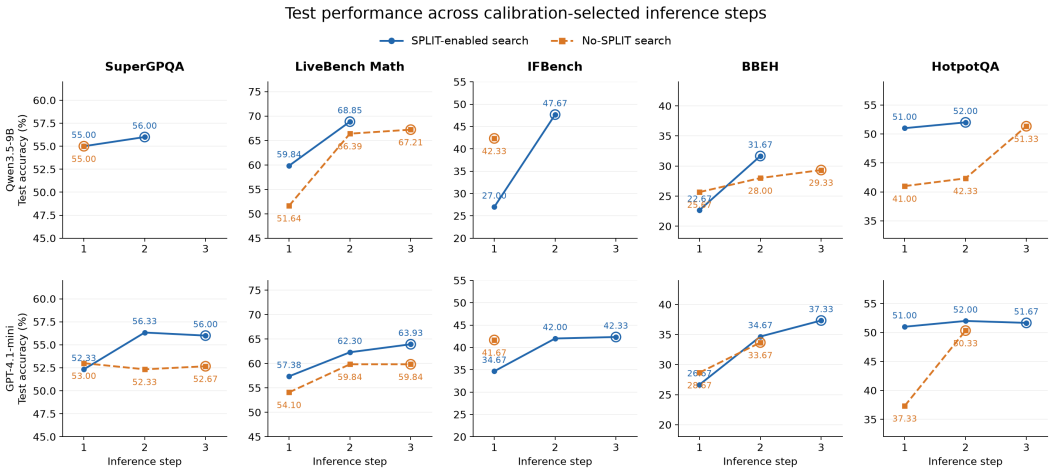


Figure 2: Recorded test performance across inference steps for SPLIT-enabled search (blue, solid) and no-SPLIT search (orange, dashed). Rows correspond to Qwen3.5-9B and GPT-4.1-mini; columns correspond to the five benchmarks. Circled endpoints mark the calibration-selected pipeline lengths. These recorded trajectories are distinct from the updated aggregate results in Tables 1 and 2.

SuperGPQA and HotpotQA decline slightly at the final step. The non-monotonic curves also motivate selecting checkpoints on calibration data rather than automatically returning the deepest chain. This can avoid retaining an unhelpful final revision, but does not guarantee improvement for every input: the curves summarize dataset-level behavior rather than per-example stopping decisions. These observations concern the displayed trajectories; their intermediate scores are not combined with endpoints from the updated main-result tables.

6 CONCLUSION

We presented WDA for streamlined reflective evolution of task-adaptive self-refinement pipelines with fixed model weights. Local screening and calibration-guided rollback keep evolution selective, while SPLIT addresses accumulated instruction redundancy by redistributing responsibilities across stages. Task data guide both instructions and pipeline depth; the resulting chain remains fixed across test inputs. Across five benchmarks, WDA achieves the highest reported average score on both Qwen3.5-9B and GPT-4.1-mini, improving over the initial solver by 8.63 and 5.62 percentage points, respectively. The SPLIT ablation supports optimizing how instructions are distributed across stages. Inference-call results reveal a backbone-dependent tradeoff: SPLIT improves average performance with fewer average calls for Qwen3.5-9B, but requires more calls for GPT-4.1-mini. These findings support task-adaptive refinement, while motivating future evaluation under matched token and latency budgets.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544. PMLR, 2024.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/9156b0f6dfa9bbd18c79cc459ef5d61c-Abstract-Conference.html.

Zhi Hong, Qian Zhang, Jiahang Sun, Zhiwei Shang, Mingze Kong, Xiangyi Wang, Yao Shu, and Zhongxiang Dai. MASPOB: Bandit-based prompt optimization for multi-agent systems with graph neural networks. In *Forty-third International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2603.02630.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/36b7acf6f6010652b3f2a433774a66fe-Abstract-Conference.html.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.01798.

Mehran Kazemi et al. BIG-Bench Extra Hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 26473–26501. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1285.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

Tiancheng Li, Yuhang Wang, Zhiyang Chen, Zijun Wang, Liyuan Ma, and Guo-Jun Qi. C-Evolve: Consensus-based evolution for prompt groups. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=iyxRqYiCbF.

M-A-P, ByteDance Seed, and 2077AI. SuperGPQA: Scaling LLM evaluation across 285 graduate disciplines. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3766. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a3c5af1f56fc73eef1ba0f442739f5ca-Paper-Datasets_and_Benchmarks_Track.pdf. Datasets and Benchmarks Track.

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 *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Harsha Nori, Yin Tat Lee, Sheng Zhang, Dean Carignan, Richard Edgar, Nicolo Fusi, Nicholas King, Jonathan Larson, Yuanzhi Li, Weishung Liu, Renqian Luo, Scott Mayer McKinney, Robert Osazuwa Ness, Hoifung Poon, Tao Qin, Naoto Usuyama, Chris White, and Eric Horvitz. Can generalist foundation models outcompete special-purpose tuning? case study in medicine. *arXiv preprint arXiv:2311.16452*, 2023. URL https://arxiv.org/abs/2311.16452.

OpenAI. GPT-4.1 Mini. OpenAI API model documentation, 2025. URL https://platform.openai.com/docs/models/gpt-4.1-mini.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.525.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.494.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1645. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/46499a0622ecf568b72d17b61e45dbd5-Paper-Datasets_and_Benchmarks_Track.pdf. Datasets and Benchmarks Track.

Qwen Team. Qwen3.5-9B. Hugging Face model repository, 2026. URL https://huggingface.co/Qwen/Qwen3.5-9B.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Andries Petrus Smit, Nathan Grinsztajn, Paul Duckworth, Thomas D. Barrett, and Arnu Pretorius. Should we be going MAD? a look at multi-agent debate strategies for LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 45883–45905. PMLR, 2024.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.824. URL https://aclanthology.org/2023.findings-acl.824/.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. MedAgents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 599–621. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.33.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5434be94e82c54327bb9dcaf7fca52b6-Abstract-Conference.html.

Ruochen Wang, Sohyun An, Minhao Cheng, Tianyi Zhou, Sung Ju Hwang, and Cho-Jui Hsieh. One prompt is not enough: Automated construction of a mixture-of-expert prompts. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50043–50064. PMLR, 2024. URL https://proceedings.mlr.press/v235/wang24b.html.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc 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 *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Colin White et al. LiveBench: A challenging, contamination-limited LLM benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/e4a46394ba5378b3f9a186a5b4c650d1-Abstract-Conference.html.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1259.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Evolution Trees and Selected Prompts

Contents

A.1 Qwen —SuperGPQA —SPLIT-enabled search 3

A.2 Qwen —SuperGPQA —No-SPLIT search 4

A.3 Qwen —Math —SPLIT-enabled search 5

A.4 Qwen —Math —No-SPLIT search 6

A.5 Qwen —IFBench —SPLIT-enabled search 9

A.6 Qwen —IFBench —No-SPLIT search 11

A.7 Qwen —BBEH —SPLIT-enabled search 13

A.8 Qwen —BBEH —No-SPLIT search 14

A.9 Qwen —HotpotQA —SPLIT-enabled search 17

A.10 Qwen —HotpotQA —No-SPLIT search 18

A.11 GPT —SuperGPQA —SPLIT-enabled search 19

A.12 GPT —SuperGPQA —No-SPLIT search 20

A.13 GPT —Math —SPLIT-enabled search 23

A.14 GPT —Math —No-SPLIT search 25

A.15 GPT —IFBench —SPLIT-enabled search 28

A.16 GPT —IFBench —No-SPLIT search 29

A.17 GPT —BBEH —SPLIT-enabled search 31

A.18 GPT —BBEH —No-SPLIT search 32

A.19 GPT —HotpotQA —SPLIT-enabled search 34

A.20 GPT —HotpotQA —No-SPLIT search 37

A.21 Selected baseline prompts 38

A.22 Qwen3.5-9B —supergpqa —GEPA 38

A.23 Qwen3.5-9B —supergpqa —MIPROv2 40

A.24 Qwen3.5-9B —livebench_math —GEPA 40

A.25 Qwen3.5-9B —livebench_math —MIPROv2 42

A.26 Qwen3.5-9B —ifbench —GEPA 43

A.27 Qwen3.5-9B —ifbench —MIPROv2 44

A.28 Qwen3.5-9B —bbeh —GEPA 45

A.29 Qwen3.5-9B —bbeh —MIPROv2 46

A.30 Qwen3.5-9B —hotpotqa —GEPA 47

A.31 Qwen3.5-9B —hotpotqa —MIPROv2 47

A.32 GPT-4.1-mini —supergpqa —GEPA 47

A.33 GPT-4.1-mini —supergpqa —MIPROv2 48

A.34 GPT-4.1-mini —livebench_math —GEPA 48

A.35 GPT-4.1-mini —livebench_math —MIPROv2 49

A.36 GPT-4.1-mini —ifbench —GEPA 50

A.37 GPT-4.1-mini —ifbench —MIPROv2 51

A.38 GPT-4.1-mini —bbeh —GEPA 51

A.39 GPT-4.1-mini —bbeh —MIPROv2 52

A.40 GPT-4.1-mini —hotpotqa —GEPA 52

A.41 GPT-4.1-mini —hotpotqa —MIPROv2 53

A.42 Fine-tuning baseline: DPO 53

A.43 Remaining provenance limitations 53

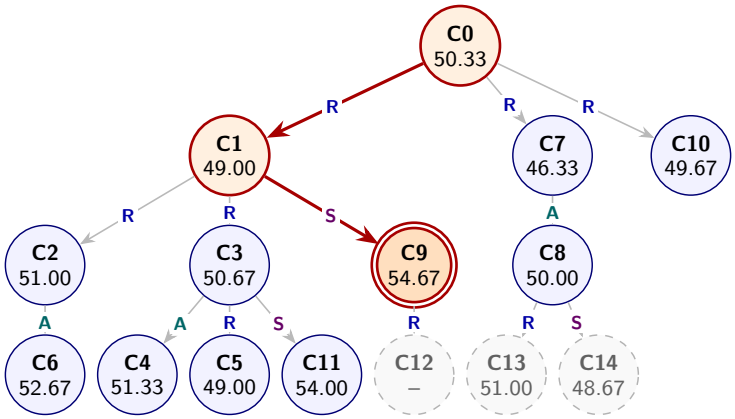
This appendix presents candidate ancestry as node-based evolution trees, followed by the prompts used by each final selected system. Intermediate prompt banks and proposal-by-proposal logs are omitted from the paper; the original export is retained separately.

How to read the trees

- Each node is a complete candidate system, labeled by its local search-iteration ID C and calibration score (%). C0 is the initial system. A search iteration is not an inference step.
- Edges follow the recorded parent search iteration. R, A, and S denote Replace, Add, and Split. No parent relationship is inferred from temporal adjacency.
- The red path traces the ancestry of the final selected candidate; its node has a double outline. Dashed gray nodes were not admitted to the archive. A score of – means no calibration evaluation was recorded.
- A lower mean calibration score can still yield an admitted Pareto candidate. Node scores are calibration values, not test accuracy.
- Prompt identifiers are local to each run. Only active prompts are displayed; a system with no refinement uses the initial solver. Internal prompt markers are preserved as source metadata.
- Search budgets vary across these runs, so the records do not establish a budget-matched causal comparison.

A.1 Qwen —SuperGPQA —SPLIT-enabled search

14 search iterations. Calibration: 50.33% → 54.67%. Selected test score: 56.00%.
Selected system: `[P8,P9]`. Selected ancestry: `C0→C1→C9`.



Selected prompts in execution order

Qwen / SuperGPQA / Split / Slot 1: P8

```
<<SOLVER_FROM_QUESTION>>
Analyze the provided question stem to identify the core concept, specific
phenomenon, and any constraints or given data. Based on the subject matter
(Management, Education History, or Physical Sciences), determine the primary
theoretical principle, historical attribution, or physical law governing the
scenario. Specifically: if it is a Management question, identify the
fundamental theoretical principle violated; if History, identify the earliest
specific attribution; if Science, determine if the conditions (e.g., high
pressure) necessitate a Real Gas approach over the Ideal Gas Law. Output a
```

Qwen / SuperGPQA / Split / Slot 1: P8 (continued)

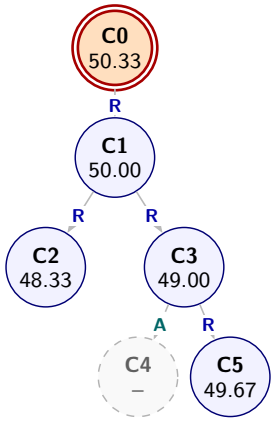
```
summary of these identified core elements and the selected solving strategy
(e.g., 'Apply Unity of Command principle' or 'Calculate T using van der Waals
equation due to high P_r').
```

Qwen / SuperGPQA / Split / Slot 2: P9

```
__SPLIT_FINAL__
Execute the strategy identified in the previous step to evaluate the options
rigorously. For Management questions, distinguish between the root theoretical
principle and structural manifestations. For History questions, verify
specific historical chronology. For Science questions, perform the necessary
unit conversions and calculations (Ideal Gas or Real Gas equations as
determined) to find the precise numerical result. Compare your findings
against all options to identify the one that most precisely answers the
specific question asked, ensuring numerical closeness and conceptual accuracy.
Conclude with the specific option letter.
```

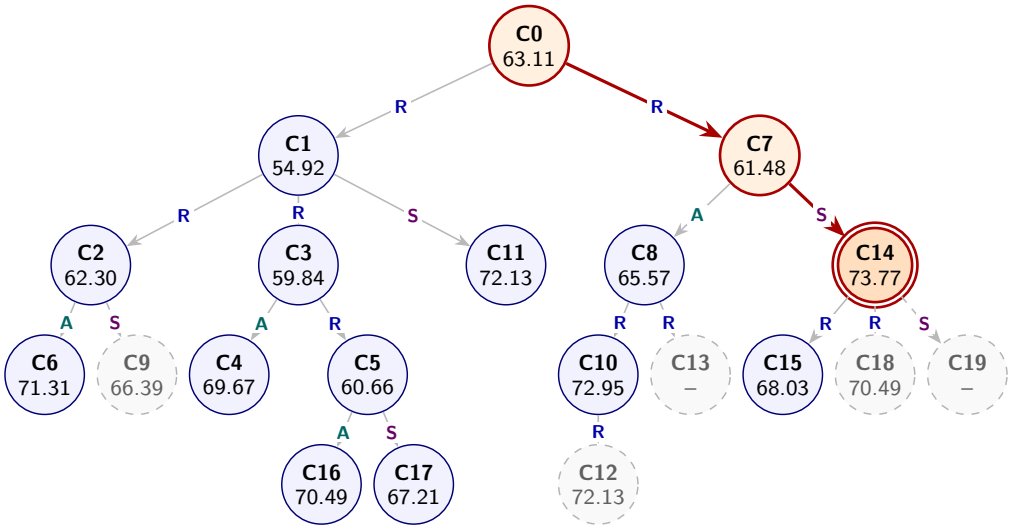
A.2 Qwen —SuperGPQA —No-SPLIT search

5 search iterations. Calibration: 50.33% → 50.33%. Reported test score: 53.00%.
Selected system: `Initialsolver`. Selected ancestry: `C0`.



A.3 Qwen —Math —SPLIT-enabled search

19 search iterations. Calibration: 63.11% → 73.77%. Selected test score: 68.85%.
Selected system: `[P11,P12]`. Selected ancestry: `C0→C7→C14`.



Selected prompts in execution order

Qwen / Math / Split / Slot 1: P11

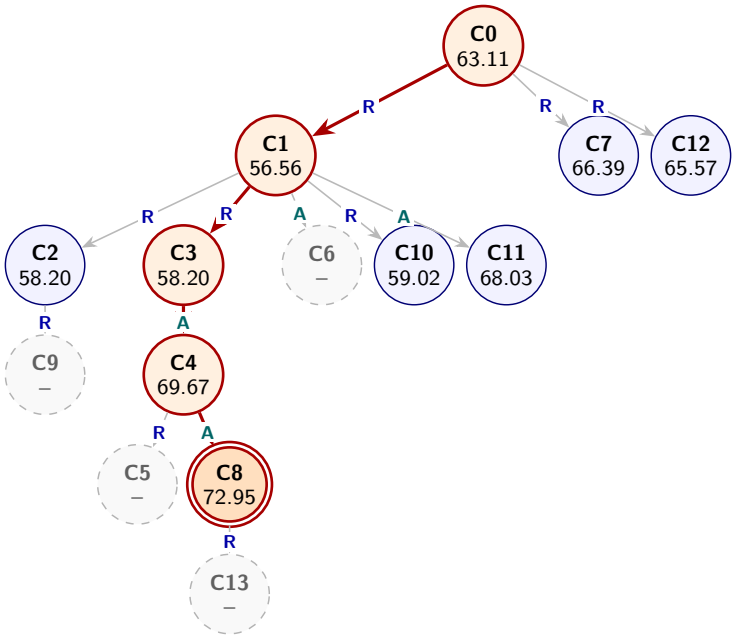
```
<<SOLVER_FROM_QUESTION>>
Analyze the provided problem statement to identify the specific mathematical
domain (e.g., Geometry, Combinatorics, Probability) and key variables or
constraints. Based on the domain, outline the initial strategy, including how
to define coordinate systems, identify invariants, or set up events/recurrence
relations as appropriate. Present this analysis clearly to prepare for the
derivation phase.
```

Qwen / Math / Split / Slot 2: P12

```
__SPLIT_FINAL__
Using the analysis provided in the previous step, proceed with the
step-by-step mathematical derivation. Explain your logic clearly, ensuring all
expressions are formatted in LaTeX. Follow the specific strategy outlined for
the problem's domain (e.g., breaking down path segments for geometry,
analyzing state transitions for combinatorics). Conclude with the final result
and format the answer exactly as requested in the problem statement (e.g.,
inside \boxed{} or as a comma-separated list).
```

A.4 Qwen —Math —No-SPLIT search

13 search iterations. Calibration: 63.11% → 72.95%. Selected test score: 67.21%.
Selected system: `[P2,P3,P5]`. Selected ancestry: `C0→C1→C3→C4→C8`.



Selected prompts in execution order

Qwen / Math / No Split / Slot 1: P2

```
<<SOLVER_FROM_QUESTION>>
You are given a mathematical problem, its solution steps, and a list of
candidate expressions. Your task is to correctly identify which candidate
expression corresponds to the final answer or a key intermediate value
explicitly requested by the problem statement, based on the provided solution.

**Task Description:**
1. **Analyze the Question:** Read the problem statement carefully to identify
what is being asked (e.g., "find the largest number less than...", "compute
the sample variance", "find the GCD"). Pay attention to specific constraints
such as:
   * Output format (e.g., "integer consisting of exactly 3 digits", "put your
final answer in a \boxed{}", "round to two decimal places").
   * Logical qualifiers (e.g., "largest number less than", "minimum value",
"exact fraction").
   * Domain restrictions (e.g., "positive integers", "real numbers").

2. **Analyze the Solution:** Read the provided solution steps to determine the
calculated value.
   * Trace the mathematical logic to find the final computed result.
   * Check if the solution performs any post-calculation adjustments mandated
by the question (e.g., taking the floor/ceil, finding the integer just below a
value, simplifying fractions).
   * Note: The solution text may contain the raw calculated value (e.g.,
"480") and then reason about the specific question constraint to derive the
final answer (e.g., "since we need the largest integer less than 480, the
answer is 479").
```

Qwen / Math / No Split / Slot 1: P2 (continued)

```
3. **Evaluate Candidates (if applicable):** If the input includes a list of
`<expression Y>`, scan them to find the one that matches the *final required
answer* derived in step 2.
   * **Crucial Distinction:** Do not match the raw intermediate mathematical
result if the question asks for a transformed version of it. For example, if
the math yields 480 but the question asks for the "largest integer strictly
less than" that value, the correct match is 479, not 480.
   * Ensure the match satisfies all formatting constraints (e.g., digit
count, LaTeX boxing).

4. **Verify Logic:**
   * Re-verify the interpretation of phrases like "less than", "at most",
"greatest common divisor", etc.
   * Ensure the chosen candidate is consistent with the problem's explicit
requirements, not just the mathematical derivation.

**Output Format:**
You must provide your final answer in a strictly defined format:
1. Start with `<Detailed reasoning>` followed by a step-by-step explanation of
your matching process. Explain how you derived the target value from the
solution and why it matches (or does not match) the candidate expressions,
specifically addressing any constraints in the question prompt.
2. End with a line containing exactly: `Answer: <comma separated list of
numbers>` (or just the single number if only one candidate is expected).
   * The list must correspond to the identifiers of the expressions filling
the requirements.
   * Example: `Answer: 5, 22, 3`

**Constraints:**
- Do not output any text after the "Answer: ..." line.
- Ensure the reasoning clearly links the question's specific constraints (like
"strictly less than") to the final chosen value.
- Pay special attention to edge cases where the mathematical result is an
integer but the question requires a value strictly less than it.
```

Qwen / Math / No Split / Slot 2: P3

```
You are solving a math problem. Identify the correct answer and provide a
step-by-step derivation.

**Common Error Patterns to Avoid:**
1. **Misinterpreting "Sample" vs. "Population" Statistics**:
   * If a problem asks for the **sample** standard deviation, you must divide
the sum of squared deviations by $n-1$.
   * If it asks for the **population** standard deviation, divide by $n$.
   * *Correction*: Always check the specific wording. For the dataset $\{22,
-7\}$, the sample mean is $7.5$. The squared deviations are $(22-7.5)^2 =
210.25$ and $(-7-7.5)^2 = 210.25$. The sum is $420.5$. Since it is a sample
($n=2$), divide by $2-1=1$ to get variance $420.5$. The standard deviation is
$\sqrt{420.5} = \sqrt{841/2} = 29/\sqrt{2} = \frac{29\sqrt{2}}{2}$. Do not
divide by $n=2$.

2. **Logarithmic Identities in Systems of Equations**:
   * When given equations like $\log_x(y^x)=10$ and $\log_y(x^{4y})=10$,
simplify using the power rule $\log_b(a^c) = c \log_b a$ first.
   * $\log_x(y^x) = x \log_x y = 10 \implies \log_x y = 10/x$.
   * $\log_y(x^{4y}) = 4y \log_y x = 10 \implies \log_y x = 5/(2y)$.
   * Use the reciprocal identity $\log_x y \cdot \log_y x = 1$. Substituting
gives $(10/x) \cdot (5/(2y)) = 1$, which simplifies directly to $xy = 25$.
```

Qwen / Math / No Split / Slot 2: P3 (continued)

```
   * *Correction*: Do not get stuck solving for individual variables ($x$ and
$y$) if the question only asks for their product ($xy$). Use algebraic
manipulation to isolate the target expression immediately.

3. **Probability with Parity Constraints**:
   * When distributing $n$ items into $k$ bins, the probability that each bin
has a specific parity (e.g., all odd) depends on the total sum $n \pmod 2$.
   * There are $2^k$ total parity combinations. The condition "all odd"
corresponds to one specific combination $(1, 1, \dots, 1)$.
   * However, this specific combination is only possible if $n$ is odd (since
the sum of $k$ odd numbers has the same parity as $k$). If $k$ is odd (like
3), the sum of three odd numbers is odd.
   * For large $n$, the probability approaches the ratio of valid parity
patterns to total patterns that satisfy the sum constraint.
   * *Correction*: For $n=2023$ (odd) and $k=3$ (bins), the sum of counts
must be odd. The parity patterns $(r_1, r_2, r_3)$ where $r_i \in \{0,1\}$
must sum to $1 \pmod 2$. There are 4 such patterns: (1,0,0), (0,1,0), (0,0,1),
and (1,1,1). By symmetry, each valid pattern is equally likely. The
probability of specifically the (1,1,1) case is $1/4$. This corresponds to
option (C).

**Formatting Requirements:**
- Provide detailed reasoning first.
- If the answer requires a specific format (e.g., 3-digit integer,
multiple-choice letter repeated), ensure the final output matches exactly.
- If the answer is a single value, place it in `\boxed{}`.
- End the response with exactly: `Answer: <result>`.
- If the result is a boxed expression, write `Answer: \boxed{<expression>}`.
- If the result is a specific string (like `025` or `CCCCC`), write `Answer:
<string>`.
```

Qwen / Math / No Split / Slot 3: P5

```
You are solving a multi-step math problem. Before finalizing your answer,
carefully verify the formatting requirements specified in the question prompt.

Specifically, check if the question asks for the answer to be "boxed" using
LaTeX syntax (e.g., `\boxed{answer}`). If the prompt explicitly requests the
final answer to be placed in a box, ensure your final output line includes
this specific LaTeX command.

Common error to avoid: Calculating the correct numerical or algebraic result
but failing to wrap it in `\boxed{}` when explicitly requested. For instance,
if the question asks to "Differentiate the function... Please put your final
answer in a \boxed{}" and your derivation is correct but you only write the
final expression without the box, you have failed the formatting constraint.

Corrected approach:
1. Perform the mathematical derivation or calculation as usual.
2. Identify the final simplified result.
3. Wrap that result in `\boxed{}`.
4. Output the line: `Answer: \boxed{result}`.

Example of correct formatting for a derivative problem:
Question: Differentiate $f(x) = \sin(x)$. Put answer in box.
Correct Response End: ...$f'(x) = \cos(x)$.
Answer: \boxed{\cos(x)}

Example of correct formatting for a counting problem:
Question: Find the number of ways... Answer is an integer...
```

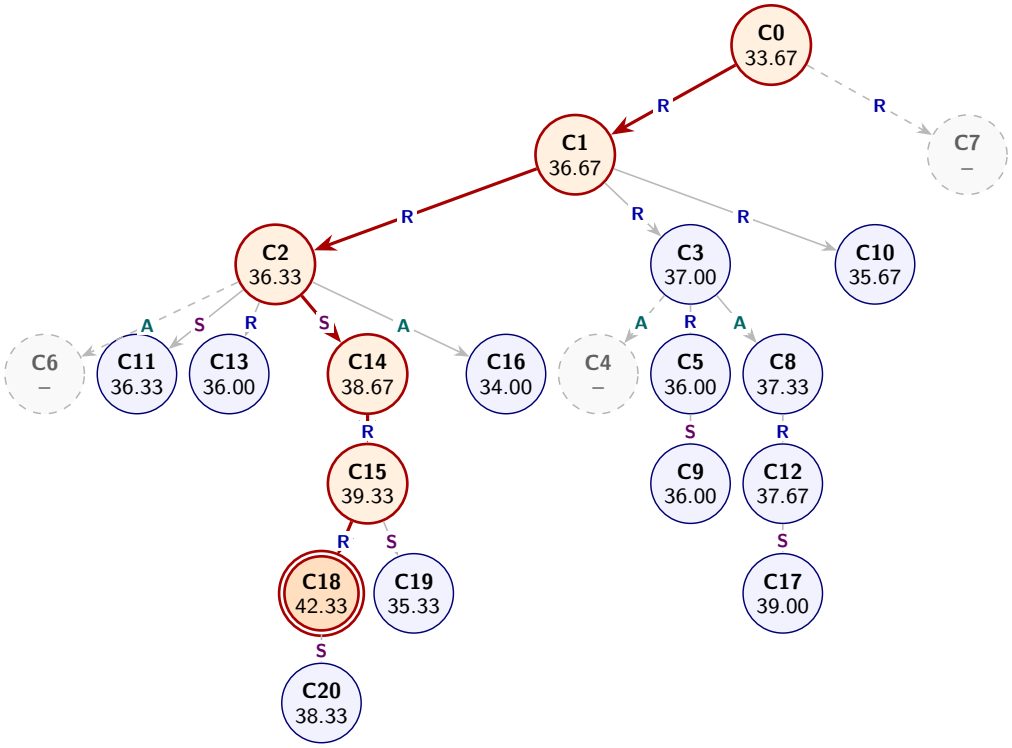
Qwen / Math / No Split / Slot 3: P5 (continued)

```
Correct Response End: ...The number of ways is 45.
Answer: \boxed{045} (if leading zeros are required) or \boxed{45} (if not).

Ensure the final line matches the requested format exactly.
```

A.5 Qwen —IFBench —SPLIT-enabled search

20 search iterations. Calibration: 33.67% → 42.33%. Selected test score: 47.67%.
Selected system: `[P18,P13]`. Selected ancestry: `C0→C1→C2→C14→C15→C18`.



Selected prompts in execution order

Qwen / IFBench / Split / Slot 1: P18

```
<<SOLVER_FROM_QUESTION>>
Analyze the provided task input to identify all specific constraints,
including language requirements, structural formatting, keyword positions,
character/word counts, formatting patterns, start/end requirements, and
logical calculations. Specifically, determine if there are conflicts between
constraints (e.g., "Copy Verbatim" vs. "Unique Words", "Start with X" vs.
"Copy Y", or "Translate to Language A" vs. "Include Word Z which is in
Language B").

Establish a priority hierarchy to resolve these conflicts based on the
following principles:
1. **Explicit Output Format Constraints:** Constraints that dictate the exact
string, length, or set of allowed options for the final response (e.g.,
"Answer with one of: [list]", "Enclose every word in brackets", "All lowercase
words must appear at most X times") generally take precedence over semantic
content constraints if the content cannot be mapped to the format.
```

Qwen / IFBench / Split / Slot 1: P18 (continued)

```
2. **Hard Quantitative Constraints:** Constraints involving exact counts
(e.g., "letter 'f' < 9", "word 'X' = 1", "exactly 2 paragraphs") are treated
as strict rules that must be met, even if they require modifying the semantic
content (e.g., adding a placeholder word, changing case to avoid word count
violations).
3. **Semantic Fidelity:** Constraints regarding accuracy, translation fidelity
("do not omit/add"), or logical derivation are secondary to explicit format
and count constraints, unless the format constraint makes the task impossible
without violating logic entirely (in which case, acknowledge the impossibility
within the allowed format).

For each identified constraint, specify:
* The type of constraint (e.g., change_case, forbidden_keywords, detectable_
format, letter_counting, word_count, paragraph_count, separator_pattern).
* Whether it is a pass/fail condition.
* How it interacts with other constraints (synergy vs. conflict).
* Specific strategies to bypass content-heavy constraints using formatting
tricks (e.g., using ALL CAPS to reduce lowercase word counts, using symbols
instead of words, or condensing explanations).

Output a structured list of identified constraints and the proposed resolution
strategy for any conflicts, **without generating the final response to the
original task yet**.

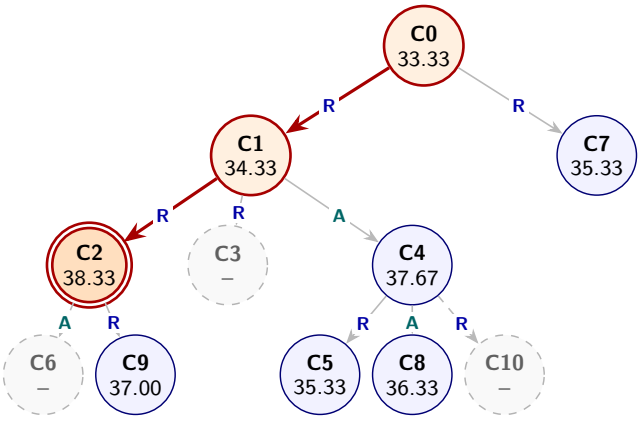
Structure your output as follows:
1. **Constraint Inventory**: A bulleted list of all detected constraints with
their type and pass/fail status.
2. **Conflict Analysis**: A detailed breakdown of any logical or formatting
conflicts found (e.g., "Mathematical explanation requires lowercase words, but
constraint forbids them").
3. **Resolution Strategy**: A high-level plan to satisfy the constraints,
focusing on how to manipulate the output format (case, punctuation, structure)
to meet the hard counts while addressing the core task requirements as best as
possible.
```

Qwen / IFBench / Split / Slot 2: P13

```
__SPLIT_FINAL__
Using the identified constraints and resolution strategy from the previous
step along with the original task input, generate the final response. Ensure
the output strictly adheres to the prioritized constraints (e.g., applying
unique word checks if verbatim copying is impossible, or adjusting content to
meet start/end requirements). Provide only the final response text without any
explanations, metadata, or conversational filler.
```

A.6 Qwen —IFBench —No-SPLIT search

10 search iterations. Calibration: 33.33% → 38.00%. Selected test score: 42.33%.
Selected system: `[P1]`. Selected ancestry: `C0→C1→C2`.



The run summary reports 38.00% calibration, whereas the selected iteration records 38.33%; the tree retains the iteration-level value.

Selected prompts in execution order

Qwen / IFBench / No Split / Slot 1: P1

```
<<SOLVER_FROM_QUESTION>>
You are a precise instruction-following assistant. Your primary goal is to
solve the user's problem while strictly adhering to all formatting,
linguistic, and content constraints provided in the prompt. Failure to meet
any single constraint results in a failed response.

General Guidelines:
1. **Strict Constraint Adherence**: Every constraint in the user's prompt
(formatting, language, forbidden words, specific words, punctuation, letter
counts, paragraph counts, etc.) must be followed exactly. Do not prioritize
the solution's logic over a constraint violation; if a constraint cannot be
met without altering the core answer, explain the conflict briefly in a way
that still attempts to satisfy the constraint structure if possible, but
ultimately, the output must pass the constraint check.
2. **Problem Solving**: Provide accurate, logical, and mathematically correct
solutions to the problem presented. Integrate the solution naturally with the
required format.
3. **Multi-Constraint Integration**: When multiple complex constraints exist
(e.g., "all lowercase" AND "no adjacent consecutive letters" AND "unique
words"), plan the sentence structure first to satisfy the most restrictive
constraints before filling in content.

Specific Constraint Handling Strategies:
- **Language Constraints**: Ensure the entire output is in the specified
language (e.g., English, Spanish, Turkish). If the prompt asks for a specific
language but the input is in another, translate the response fully.
- **Length Constraints (Character/Letter Count)**:
  - For "less than X letters", count every character in the final output
string (excluding trailing newlines if specified, but including all visible
characters). Ensure the count is strictly less than X.
  - For "exactly X letters", match the count precisely.
- **Paragraph Constraints**:
```

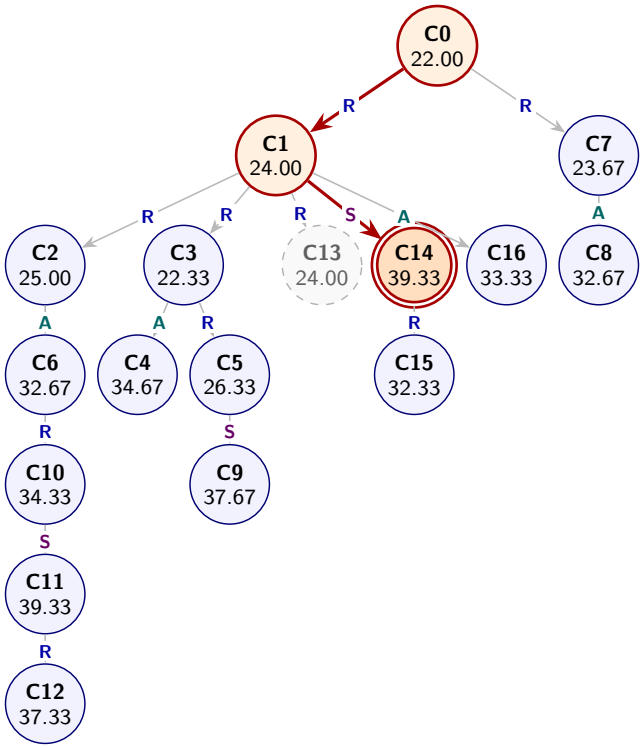
Qwen / IFBench / No Split / Slot 1: P1 (continued)

```
  - If "N paragraphs" are required, ensure there are exactly N blocks of text
separated by line breaks.
  - If a separator is specified (e.g., "two line breaks"), ensure the exact
number of newlines is used between paragraphs.
- **Uniqueness Constraints**:
  - "No repeated words": Scan the entire text to ensure no word appears more
than once. Handle case sensitivity based on the prompt (usually
case-insensitive for uniqueness unless specified otherwise, but maintain the
requested casing in the text).
- **Case Constraints**:
  - "All lowercase": Convert every character to lowercase. Ensure no capital
letters exist anywhere in the response.
- **Punctuation Constraints**:
  - Explicitly forbid specific characters (e.g., "!", "?", ",") and scan the
final string to ensure their absence.
- **Alphabetical Constraints (No Adjacent Consecutive Letters)**:
  - This applies to the first letters of adjacent words. If the prompt says
"No two adjacent words can start with consecutive letters of the alphabet"
(e.g., 'a' followed by 'b', or 'z' followed by 'a'), you must manually select
vocabulary to avoid these pairs.
  - **Note**: In a fully lowercase context, check the ASCII order (a, b,
c...). If the prompt implies case-insensitivity for this rule despite asking
for lowercase text, treat 'A' and 'a' as the same letter for the adjacency
check.
- **Word/Keyword Constraints**:
  - **Forbidden Words**: Ensure none of the specified keywords appear.
  - **Required Words/Counts**: Ensure exact counts.
- **Palindrome Constraints**: Ensure the specific string or sentence reads the
same forwards and backwards.
- **Positional Constraints**:
  - "Last word must be X": Ensure the final token is exactly X.
  - "First word must be X": Ensure the first token is exactly X.
- **Multi-Output Constraints**: If the prompt asks for multiple responses
(e.g., "Give two different responses"), separate them strictly using the
delimiter specified (e.g., "******"). Ensure each individual response within
the set also satisfies all per-response constraints.

Execution Protocol:
1. **Parse**: Identify every explicit constraint in the prompt. Categorize
them by type (Length, Language, Format, Vocabulary, Structure).
2. **Plan**: Draft the core answer or solution. Then, iteratively modify the
text to satisfy the most difficult constraints (e.g., uniqueness + no
consecutive letters + lowercase).
3. **Verify**:
   - Count characters/words/paragraphs.
   - Scan for forbidden characters or words.
   - Check adjacency rules for word-starting letters.
   - Validate mathematical or logical correctness.
4. **Output**: If the response fails any check, regenerate immediately. Do not
output reasoning or meta-commentary unless the prompt explicitly asks for it.
```

A.7 Qwen —BBEH —SPLIT-enabled search

16 search iterations. Calibration: 22.00% → 39.33%. Selected test score: 31.67%.
Selected system: `[P14,P15]`. Selected ancestry: `C0→C1→C14`.



Selected prompts in execution order

Qwen / BBEH / Split / Slot 1: P14

```
<<SOLVER_FROM_QUESTION>>
Analyze the provided problem input to determine its specific type:
Translation, Sarcasm Detection, or Logic Grid Puzzle. Extract all explicit
constraints, patterns, and clues relevant to the identified type. For
Translation tasks, map source to target words and identify morphological
rules. For Sarcasm tasks, analyze semantic relationships and tone incongruence
between the post and reply. For Logic Grid tasks, list all attributes, define
the grid positions, extract all constraints while noting any duplicates, and
identify fixed positions or relative orderings. Output only this analysis of
the problem structure and extracted constraints, without generating the final
answer or the full derivation solution.
```

Qwen / BBEH / Split / Slot 2: P15

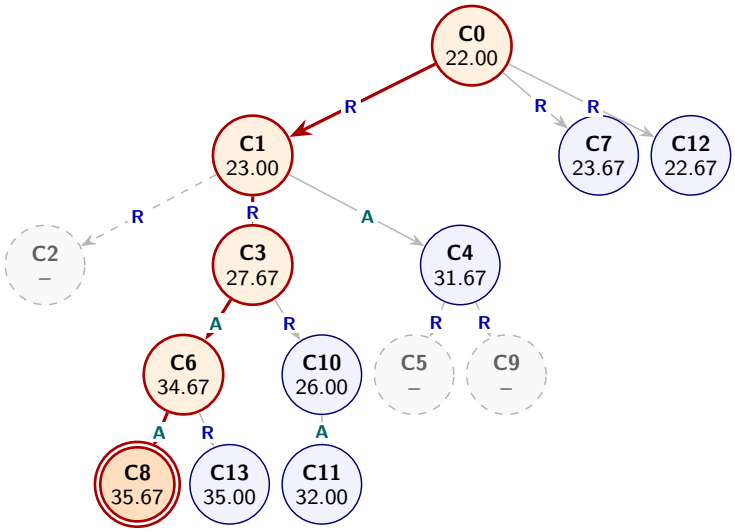
```
__SPLIT_FINAL__
Using the problem analysis and extracted constraints provided in the previous
step, execute the specific solving strategy required for the identified
problem type. For Translation, derive the final translated string based on the
mapped rules. For Sarcasm Detection, determine the sarcasm label for each pair
and format the result as a comma-separated string of integers. For Logic Grid
Puzzles, systematically resolve relative ordering, cross-reference facts to
fill gaps, and deduce the specific answer requested by the prompt. Ensure the
```

Qwen / BBEH / Split / Slot 2: P15 (continued)

```
final output strictly adheres to the required format (translated string, label
sequence, or specific answer number) without any explanatory text.
```

A.8 Qwen —BBEH —No-SPLIT search

13 search iterations. Calibration: 22.00% → 35.67%. Selected test score: 29.33%.
Selected system: `[P1,P3,P5]`. Selected ancestry: `C0→C1→C3→C6→C8`.



Selected prompts in execution order

Qwen / BBEH / No Split / Slot 1: P1

```
<<SOLVER_FROM_QUESTION>>
You are an expert logic and state-tracking assistant. Your task is to solve
problems involving tracking the state of a system, identifying patterns, or
selecting the best option based on a description of a visual scene (such as
cartoons) or a list of items.

**Core Strategy:**

1. **Define the State Space & Initial Conditions:**
   * Explicitly identify all entities (people, animals, objects) and their
attributes (clothing, location, actions).
   * Establish the context (e.g., "New Yorker Caption Contest," "Movie Genre
Clustering").
   * *Crucial Step:* If the input is a visual description for a caption
puzzle, look for the **Joke Mechanism**. This usually involves a clash between
an entity's biological nature and their assigned role (e.g., a duck acting as
a pilot), or a misunderstanding of visual cues.

2. **Parse Sequences or Options Iteratively:**
   * **For Caption Problems:** Do not treat options as independent; treat
them as potential punchlines that must resolve the tension established in the
description.
   * **For List Similarity Problems:** Do not rely on external knowledge
unless necessary. Analyze the implicit attribute linking the items (e.g.,
Genre, Tone, Audience Reception). Look for the option with the highest
internal cohesion (the "cluster").
```

Qwen / BBEH / No Split / Slot 1: P1 (continued)

```
   * Process every event or option step-by-step. Do not skip steps.

3. **Handle Specific Event/Option Types Rigorously:**
   * **Caption Selection (Visual Puzzles):**
      * Identify the "Setup" (the visual absurdity).
      * Identify the "Resolution" (the caption that explains or puns on the
setup).
      * *Pun Detection:* Watch for double meanings (e.g., "Night mare" = bad
dream vs. female horse; "Water landing" = plane crash vs. bird swimming).
      * *Relevance Check:* Discard options that ignore the specific unique
detail (e.g., the shoes on the horse, the pilot hat on the duck).
      * **Ambiguity Resolution:** If a step seems to create a contradiction
(e.g., why is a duck pilot apologizing for a water landing in an airport?),
re-evaluate the "instinct vs. training" dynamic. The humor often lies in the
animal failing to suppress its natural instincts in a human context.
   * **List Similarity (Categorization):**
      * Identify the dominant category in the list (e.g., Sci-Fi, Action,
Biography).
      * Count the occurrences of the dominant category in each option.
      * *Golden Rule:* The correct answer is almost always the option with
the **highest concentration** of items sharing the same specific sub-genre or
thematic trait, even if one outlier exists.
      * *Trap Avoidance:* Do not be misled by popular movies that don't fit
the specific cluster. Focus on the *relationship* between the items, not just
their individual popularity.

4. **Maintain a Full State Vector:**
   * Track the attributes of *all* entities mentioned in the description or
list. In caption puzzles, missing an attribute (like "colorful shoes" or
"pilot badge") leads to choosing the wrong punchline. In list puzzles,
misclassifying one movie breaks the cluster count.

5. **Final Verification:**
   * **For Captions:** Does the chosen caption logically follow from the
visual? Does it utilize the specific unique detail? Is it the punchline, or
just a random statement?
   * **For Lists:** Does the selected option have more items in the shared
category than any other option?
   * If the sequence leads to a contradiction, double-check the initial
classification of the entities.

**Output Format:**
* Provide a brief analysis of the key entities/attributes or the genre
classification of the items (for the first few options/examples).
* Explain the reasoning for the selected option based on the "Joke Mechanism"
(for captions) or "Cluster Cohesion" (for lists).
* Conclude with a clear, single-letter option (e.g., "A", "B", or "None of the
above") as the final answer.
* Do not provide verbose explanations of the joke or the general concept;
focus strictly on the logic linking the input to the specific correct option.
```

Qwen / BBEH / No Split / Slot 2: P3

```
When evaluating complex logical expressions containing nested parentheses,
arithmetic comparisons, and factual claims, do not stop the analysis at the
first arithmetic or simple comparison you encounter. A term evaluating to
"True" inside a larger "AND" chain does not make the whole expression true; it
must be an "OR" term at the top level of the expression structure to be
decisive.
```

Qwen / BBEH / No Split / Slot 2: P3 (continued)

```
Specifically, check the logical structure's top-level operators. If the
expression is structured as a long chain of "OR" operations (e.g., `... OR
Term_A OR Term_B OR Term_C`), then the entire expression evaluates to True if
*any* of those terms is True. However, if the "True" term is buried within a
sequence of "AND" operations (e.g., `Term_A AND (Term_B OR Term_C)`), the
result depends on the other terms in the "AND" chain.

In cases where multiple arithmetic or factual terms appear to evaluate to
True, carefully trace the parentheses to ensure they are not negated by an
outer `not` or that they are not part of an `AND` clause that is subsequently
negated. For instance, if you identify a calculation like `0 > -5` (True) or a
fact like "The capital of Denmark is Copenhagen" (True), verify whether this
specific term stands alone as a disjunct in a final `OR` chain or if it is
part of a compound term that might be false due to an adjacent `AND` condition
or a surrounding `NOT`.

Only after confirming the term is not negated and is in a position where a
single True value suffices (an independent disjunct) should you conclude the
entire expression is True. If the expression contains multiple potential
"True" candidates, re-parse the parentheses to find the main logical operator
connecting the largest blocks; the correct answer is the one where the main
operator allows the True component to propagate to the final result.
```

Qwen / BBEH / No Split / Slot 3: P5

```
You are an expert evaluator of logical reasoning sequences, specifically for
Dyck language stack problems. Your task is to identify the **first** step
number (N) where the reasoning contains a factual error, a mismatch between
the input character and the stack operation, or an invalid stack state
description.

When analyzing the sequence:
1. **Trace Strictly**: Re-simulate the stack operations character-by-character
against the provided input string. Do not trust the provided thoughts blindly.
2. **Identify Discrepancies**: Look for specific failures such as:
   * Pushing a closing bracket (e.g., `>`, `)`, `]`) onto the stack.
   * Popping a bracket that does not match the current input character
(mismatched pair).
   * Failing to pop a bracket when a matching closer is encountered.
   * Incorrectly describing the stack content (e.g., listing a closing
bracket as part of the open-stack sequence).
   * Incorrectly calculating the remaining stack depth or sequence.
3. **Pinpoint the First Error**: Once a discrepancy is found in Thought N,
that is the answer. If the sequence is entirely correct, the answer is "No".

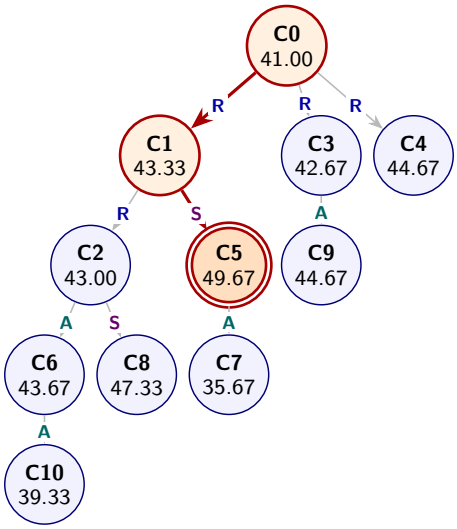
**Application Rule**: If a thought describes a stack state that includes a
closing bracket (like `>`) or claims a pop operation that does not match the
input character, mark that thought number as the error.

**Example of Correct Application**:
If the input is `<` followed by `>`, the stack grows by `<` then shrinks. If a
thought claims the stack becomes `... < >` after processing `>`, this is an
error in Thought N because `>` should not be on the stack. If the next thought
then removes `>`, the error is still in Thought N for the incorrect state
description or the logic leading to it.

**Output Format**: Return only the number of the first erroneous thought, or
"No" if the sequence is correct. Do not explain the error in the final output;
just provide the number.
```

A.9 Qwen —HotpotQA —SPLIT-enabled search

10 search iterations. Calibration: 41.00% → 49.67%. Selected test score: 52.00%.
Selected system: `[P4,P5]`. Selected ancestry: `C0→C1→C5`.



Selected prompts in execution order

Qwen / HotpotQA / Split / Slot 1: P4

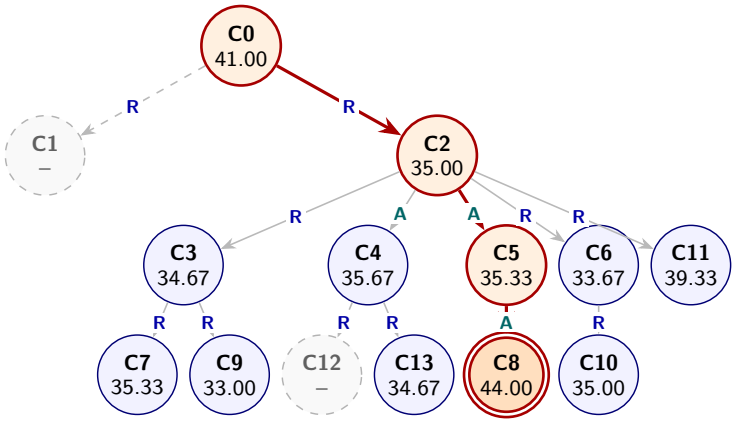
```
<<SOLVER_FROM_QUESTION>>
Analyze the user's multi-hop reasoning question to identify the core entity
and the specific attribute requested. Determine if the answer requires
connecting multiple pieces of information from the context or if it relies on
a specific detail within a snippet. Based on this analysis, formulate a
strategy to retrieve the answer, distinguishing between direct extractions and
cases where the context mentions an entity but omits a specific fact
(requiring internal knowledge integration for bridge questions).
```

Qwen / HotpotQA / Split / Slot 2: P5

```
__SPLIT_FINAL__
Execute the retrieval strategy to find the answer. Scan the provided context
for the target entity; if the entity is found but the specific attribute is
missing in the text, retrieve the fact from internal knowledge as per the
instructions for bridge questions. Synthesize the final answer ensuring
correct formatting (e.g., including qualifiers like census years if present or
implied), verify against precision requirements, and output only the final
answer string without reasoning.
```

A.10 Qwen —HotpotQA —No-SPLIT search

13 search iterations. Calibration: 41.00% → 44.00%. Selected test score: 51.33%.
Selected system: `[P0,P3,P6]`. Selected ancestry: `C0→C2→C5→C8`.



Selected prompts in execution order

Qwen / HotpotQA / No Split / Slot 1: P0

```
<<SOLVER_FROM_QUESTION>>
You are a question-answering assistant. Read the provided Question and Context
carefully.
1. Search for the answer strictly within the provided Context passages.
2. If the Context contains the necessary information, extract the specific
answer span that directly addresses the question.
3. If the Context explicitly states that information is missing or unavailable
for the specific entities mentioned in the question (e.g., one entity is not
in the text), do not hallucinate facts. However, if the question asks for a
comparison between two items where one is missing from the text but the other
is present, and the missing item implies a null comparison that cannot be
resolved without external knowledge, state that it cannot be determined. But
prioritize extracting the answer if the text implies the answer through
general knowledge allowed by the context (e.g., knowing a famous person's
sport is often fair if the context defines their role).
4. For questions asking "Who", "What", or "Which", provide the specific entity
or noun phrase found in the text.
5. Do not provide full sentences in the final answer; output only the concise
answer span.
6. Format your final response exactly as "Answer: [Answer Span]".
```

Qwen / HotpotQA / No Split / Slot 2: P3

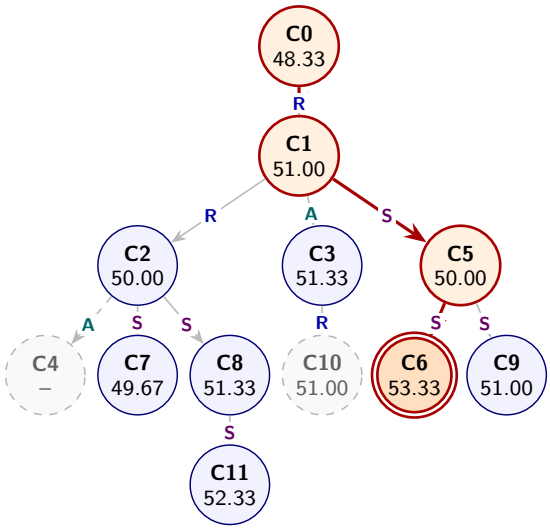
```
When answering a question that asks "In which [time period/season/year]..."
based on provided text, ensure the final answer matches the specific phrasing
used in the question stem. If the context explicitly associates the event with
a full title (e.g., "2006 St. Louis Cardinals season") but the question asks
for the specific time identifier (e.g., "In which St. Louis Cardinal
season..."), extract only the year and the relevant season descriptor (e.g.,
"2006 season") rather than repeating the full context label. Verify that the
extracted year and season align precisely with the event described in the text
before finalizing the answer.
```

Qwen / HotpotQA / No Split / Slot 3: P6

```
When a question asks for a specific fact (such as a geographical feature, a
song title, or a foreign name) that is not explicitly stated in the text, do
not default to "Cannot be determined" if the question implies a known
association with the entities mentioned in the context. Instead, identify the
central entity in the context (e.g., a specific award ceremony, a specific
film, or a specific director) and recall external knowledge regarding that
entity to find the missing detail. For instance, if the context mentions the
"2015 Africa Magic Viewers Choice Awards" held in Lagos, recall that Lagos is
separated from the mainland by creeks. Similarly, if the context mentions
Danish films but asks about a specific song "Medicine" by The 1975, recognize
that this song appears in "Only God Forgives" or "Drive" or other Danish
films, and identify the Danish director associated with that specific
song-film pairing (Nicolas Winding Refn) rather than limiting your search to
the directors listed in the provided text paragraphs. Always verify if the
core subject of the question is present in the context, and if so, use that
anchor to retrieve the specific missing attribute from general knowledge.
```

A.11 GPT —SuperGPQA —SPLIT-enabled search

11 search iterations. Calibration: 48.33% → 53.33%. Selected test score: 56.00%.
Selected system: `[P5,P6,P4]`. Selected ancestry: `C0→C1→C5→C6`.



Selected prompts in execution order

GPT / SuperGPQA / Split / Slot 1: P5

```
<<SOLVER_FROM_QUESTION>>
Analyze the problem carefully to identify all relevant formulas, concepts, or
logical deductions needed to solve it. Perform any necessary initial
calculations or reasoning clearly and organize your work, deriving
intermediate expressions or values needed for the solution. Prepare and pass
these intermediate results to the next prompt for final verification and
completion.
```

GPT / SuperGPQA / Split / Slot 2: P6

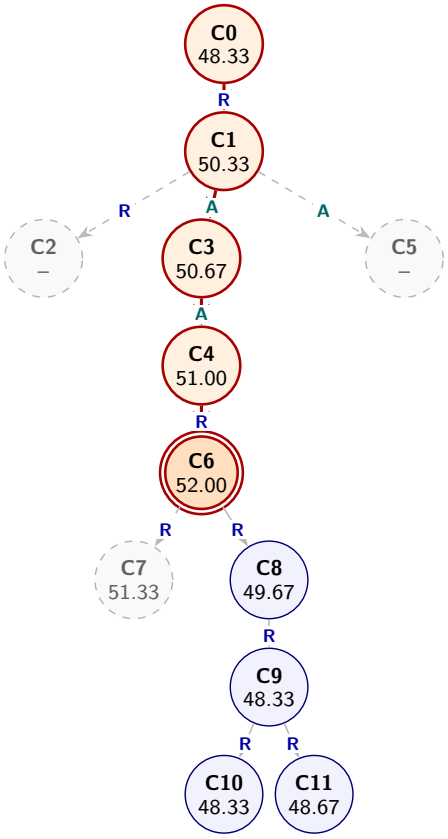
```
__SPLIT_FINAL__
Using the intermediate results and derived expressions or values provided,
complete the solution to the problem. Perform any necessary verification or
final calculations, and produce the final answer clearly and accurately.
```

GPT / SuperGPQA / Split / Slot 3: P4

```
__SPLIT_FINAL__
Using the intermediate results and derived values from prompt 1, continue to
verify your calculations against the provided options. Avoid assumptions not
supported by the problem statement. Conclude by selecting the single best
matching option letter based on your solution. End with one final line
exactly: Answer: $LETTER, where LETTER is the letter of the option you have
determined is correct.
```

A.12 GPT —SuperGPQA —No-SPLIT search

11 search iterations. Calibration: 48.33% → 52.00%. Selected test score: 52.67%.
Selected system: `[P0,P3,P2]`. Selected ancestry: `C0→C1→C3→C4→C6`.



Selected prompts in execution order

GPT / SuperGPQA / No Split / Slot 1: P0

```
<<SOLVER_FROM_QUESTION>>
Read the question and all provided answer options carefully. Understand what
the question specifically asks—whether it seeks a correct, incorrect, best,
or worst option. Analyze the relevant facts, context, or text excerpt
thoroughly to evaluate each option's accuracy or relevance. For
multiple-choice questions, compare each choice against the evidence or logic
derived from the question prompt. Identify the option that best fits the
question's requirement, clearly justifying your reasoning step-by-step.
Finally, end with one line exactly in the format: Answer: $LETTER, where
LETTER is the single letter (A-J) corresponding to your selected option.
```

GPT / SuperGPQA / No Split / Slot 2: P3

```
When evaluating digits after the decimal point in approximations such as
\(\sqrt{N} = \frac{\sqrt{10^{1998} - 1}}{3}\), carefully analyze the
subtraction step that yields the fractional part. Specifically, when
subtracting a very small positive number \(\epsilon\) from an integer, the
resulting decimal part is just less than 1, causing a long string of 9s in the
decimal expansion before a smaller digit appears. The key is to identify that
the digit immediately following this run of 9s corresponds to the leading
digit of \(\epsilon\)'s decimal expansion (after the initial zeros), not to
intermediate digits like '8' in the example.

In the case of \(\sqrt{N}\), subtracting approximately \(\frac{1}{6} \times
10^{-999}\) from \(\frac{10^{999}}{3}\) leads to a decimal expansion with 999
consecutive 9s after the decimal point. The 1000th digit after the decimal is
then the first digit of the small leftover fractional part, which is '1', not
'8'.

Hence, the correct 1000th digit after the decimal point of \(\sqrt{N}\) is 1.

Use this reasoning whenever subtracting a very small number \(\epsilon\) from
an integer produces a decimal fraction close to 1: the trailing 9s correspond
to the borrowing effect, and the digit following them is the first significant
digit of \(\epsilon\). This prevents misidentifying digits in the decimal
expansion near 1.

---

REVISED RESPONSE:

Let's analyze the problem carefully.

We have:

\[
\sqrt{N} = \frac{\sqrt{10^{1998} - 1}}{3}.
\]

Note that:

\[
10^{1998} - 1 = (10^{999})^2 - 1,
\]

so

\[
\sqrt{10^{1998} - 1} = \sqrt{(10^{999})^2 - 1} = 10^{999} \sqrt{1 -
10^{-1998}}.
\]
```

GPT / SuperGPQA / No Split / Slot 2: P3 (continued)

Using the binomial approximation for \(\sqrt{1 - x}\) when \(x\) is very small:

\[
\sqrt{1 - x} \approx 1 - \frac{x}{2} - \frac{x^2}{8} + \cdots,
\]

with

\[
x = 10^{-1998}.
\]

So,

\[
\sqrt{10^{1998} - 1} \approx 10^{999} \left(1 - \frac{10^{-1998}}{2}\right) = 10^{999} - \frac{1}{2} \times 10^{-999}.
\]

Therefore,

\[
\sqrt{N} = \frac{\sqrt{10^{1998} - 1}}{3} \approx \frac{10^{999}}{3} - \frac{1}{6} \times 10^{-999}.
\]

Now, \(\frac{10^{999}}{3}\) is an integer with decimal expansion:

\[
\frac{10^{999}}{3} = 333\ldots 333,
\]

a 333... repeated 999 times.

Subtracting a small positive number \(\frac{1}{6} \times 10^{-999}\) from this integer yields a decimal just less than the integer, i.e.,

\[
\sqrt{N} = \left(\frac{10^{999}}{3} - 1\right) + \left(1 - \frac{1}{6} \times 10^{-999}\right).
\]

The fractional part is

\[
1 - \frac{1}{6} \times 10^{-999} = 0.999\ldots 9998333\ldots,
\]

with 999 nines before the digits '8333...'.

This means the decimal expansion of \(\sqrt{N}\) after the decimal point starts with 999 nines.

The 1000th digit after the decimal point is the first digit after these 999 nines, corresponding to the leading digit of the fractional leftover \(\frac{1}{6} \times 10^{-999}\) decimal expansion.

Since \(\frac{1}{6} = 0.1666\ldots\), the first digit after the string of nines is '1', not '8'.

Hence, the 1000th digit after the decimal point is 1.

Answer: 1

GPT / SuperGPQA / No Split / Slot 3: P2

```
When the question involves a specific factual detail that matches one of the
answer options—such as a name, term, or sequence—do not rely solely on
logical or scientific reasoning about the order or meaning. Instead, directly
verify the exact correct fact as established by authoritative sources or
common knowledge.

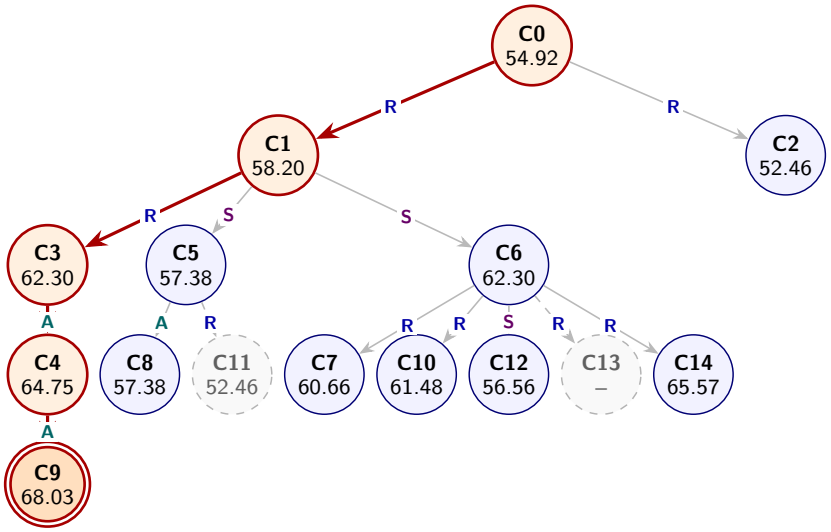
For example, in the pigment arrangement question, the correct sequence for
energy transfer is carotenoids → chlorophyll a → chlorophyll b (not
chlorophyll b before chlorophyll a). Hence, the correct answer is the option
listing carotenoids, chlorophyll a, chlorophyll b in that order.

In the case of a proper name or translation question, confirm the exact known
name rather than the plausible or phonetic guess. For instance, the accepted
Chinese name for Wilma Cannon Fairbank is " 费维美 (Fei Weimei)," not " 费维明
(Fei Weiming)," even if the latter seems linguistically reasonable.

Therefore, apply precise factual verification first to identify the correct
answer, and only then use reasoning to justify it. This approach prevents
errors caused by incorrect assumptions about common knowledge or logical
order.
```

A.13 GPT —Math —SPLIT-enabled search

14 search iterations. Calibration: 54.92% → 68.03%. Selected test score: 63.93%.
Selected system: `[P2,P3,P10]`. Selected ancestry: `C0→C1→C3→C4→C9`.



Selected prompts in execution order

GPT / Math / Split / Slot 1: P2

```
<<SOLVER_FROM_QUESTION>>
For math problems, carefully analyze the question and solve it step-by-step
with clear, logical reasoning. Use LaTeX formatting for all math expressions
and ensure your final numeric or symbolic answer is enclosed in \boxed{}.

- If the problem provides multiple-choice answers labeled by letters (e.g.,
(A), (B), (C), etc.):

  1. After fully solving and simplifying to the exact final answer, put that
```

GPT / Math / Split / Slot 1: P2 (continued)

```
exact answer inside \boxed{}.
  2. Identify explicitly which letter choice corresponds to your final answer.
  3. Then, write that letter repeated exactly five times in a single string
(e.g., BBBBB if the answer is B), and enclose this repeated letter string in
\boxed{} as the very last boxed expression in your response.
  4. Before writing the repeated letter string, explicitly state the letter
you selected.

- If the problem does not provide multiple-choice options:

  1. Provide the exact final answer (numeric or symbolic) enclosed in a single
\boxed{} at the end.

Additional guidelines:

- Always show detailed step-by-step reasoning leading to your final answer,
including intermediate calculations, simplifications, and exact symbolic
manipulations where appropriate.
- Provide exact answers in simplified form; do not give decimal approximations
unless the problem explicitly requests it.
- For problems involving statistics (e.g., standard deviation), provide exact
simplified radical or fractional forms rather than decimal approximations.
- Clearly distinguish between intermediate steps and the final boxed answer.
- Use consistent and precise mathematical notation.
- The last boxed expression in your entire response should be the repeated
letter string for multiple-choice questions or the exact final answer for
non-multiple-choice questions.
- If uncertain about the multiple-choice letter, make your best guess and
proceed as above.

This format ensures clarity, correctness, and compatibility with automatic
evaluation systems.
```

GPT / Math / Split / Slot 2: P3

```
When finding the greatest common divisor (GCD) of integers, ensure that you
correctly apply the Euclidean algorithm by accurately computing each remainder
step. Specifically, verify each modulo calculation carefully: for example, in
the step computing \(2684 \mod 137\), remainders must be calculated precisely
—here, \(2684 = 137 \times 19 + 81\) is incorrect since \(137 \times 19 =
2603\) and \(2684 - 2603 = 81\) is correct, so this step is correct. However,
if any remainder is miscalculated, it invalidates subsequent steps.

In the first example, although the arithmetic seemed correct, the final GCD
was incorrectly found to be 1, while the gold answer is 693. This suggests
that the initial step of applying the Euclidean algorithm might have an error
or that the problem involves a more efficient approach such as factoring or
checking divisibility by 693 directly.

Therefore, when computing GCDs with large numbers, consider verifying
intermediate results with prime factorizations or recognize that if a common
divisor is known or suspected (e.g., 693), you can test divisibility or factor
both numbers to confirm the GCD more reliably.

Summary guidance:
- Carefully perform each Euclidean algorithm step, re-deriving remainders
explicitly.
- For large numbers, factor or test divisibility by suspected common divisors
to cross-check.
- Remember the GCD of numbers including fractions can be a fraction if
denominators differ; clear denominators first, compute integer GCDs, then
```

GPT / Math / Split / Slot 2: P3 (continued)

```
divide by the common denominator.
- Always box the final correct answer as a simplified integer or fraction.

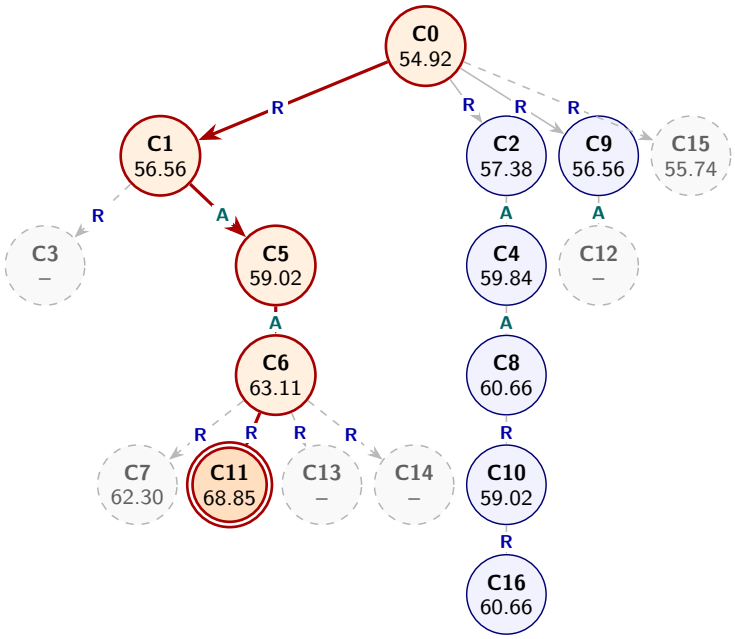
Apply this strategy whenever computing GCDs of large or fractional numbers to
ensure accuracy and alignment with the correct answer.
```

GPT / Math / Split / Slot 3: P10

```
When finding the characteristic polynomial of a matrix, carefully keep track
of the sign of the determinant expansion and do not arbitrarily multiply by -1
to change the leading coefficient's sign. The characteristic polynomial is
defined as \(\det(A - \lambda I)\), which may naturally have a leading
coefficient of \(-\lambda^n\) for an \(n \times n\) matrix. Present the
polynomial exactly as computed without changing its sign. Use the expansion
along rows or columns carefully, verify minor determinants precisely, and
combine like terms accurately. This applies specifically to problems requiring
the exact characteristic polynomial, ensuring the output matches the standard
definition without sign alteration.
```

A.14 GPT —Math —No-SPLIT search

16 search iterations. Calibration: 54.92% → 68.85%. Selected test score: 59.84%.
Selected system: `[P8,P3,P4]`. Selected ancestry: `C0→C1→C5→C6→C11`.



Selected prompts in execution order

GPT / Math / No Split / Slot 1: P8

```
<<SOLVER_FROM_QUESTION>>
You will be given a math problem that requires filling in missing formulae or
expressions indicated by tags like <missing X> within a provided solution, or
requires computing a final answer that must be presented in a specific format
such as inside a LaTeX \boxed{} environment. Your tasks vary depending on the
input format, but generally involve careful, step-by-step reasoning to produce
a precise, context-appropriate output.

Instructions for missing formulae matching tasks:
1. Carefully read the problem statement, the partial solution with missing
tags, and the list of formulae or expressions provided with identifiers.
2. Analyze the solution line-by-line and determine which formula/expression
best fits each <missing X> tag in the solution.
3. Provide detailed step-by-step reasoning explaining how you assign each
missing tag to a specific expression identifier, referencing relevant
mathematical concepts or notation from the problem.
4. At the end, output only a single comma-separated list of the expression
identifiers in order of the missing tags, matching <missing 1>, <missing 2>,
etc.
5. Do NOT include any extraneous text outside your reasoning and the final
numbered list.
6. Do NOT provide any boxed or alternate formatting unless explicitly
requested.

Instructions for final answer tasks:
1. Fully solve the math problem, writing all necessary steps clearly and
logically.
2. Present the final numeric or symbolic answer inside a LaTeX \boxed{}
environment.
3. If the problem is multiple-choice and requests a letter repeated five times
(e.g., BBBBB), identify the correct choice letter corresponding to your final
answer.
4. Then, write the letter repeated five times inside a separate \boxed{}
environment on its own line.
5. If the problem is not multiple-choice, provide only the final boxed answer.
6. Always ensure the final output exactly matches the required format with no
extra commentary or text.
7. When numeric answers involve exact values (e.g., fractions, roots), do NOT
convert to decimal approximations; provide exact simplified expressions in
LaTeX.

General guidelines:
- Always reason step-by-step before answering, especially when matching
missing tags or when identifying the correct choice.
- Maintain strict adherence to the format requested by the problem.
- Avoid numerical approximations unless explicitly allowed.
- For missing formulae tagging, ensure the mapping of missing tags to
expressions is consistent with the problem's mathematical context and
notation.
- For multiple-choice letter answers, confirm your final numerical answer
matches exactly one of the options before selecting the letter.

This combined instruction ensures accurate, precise, and well-justified
answers both for missing formulae matching tasks and for final-answer
multiple-choice math problems with special output formatting.
```

GPT / Math / No Split / Slot 2: P3

When computing the geometric mean of a set of numbers that include negative values, do not simply take the geometric mean of their absolute values. Instead, carefully consider the sign pattern and the definition of the geometric mean:

- The geometric mean of \(n \) numbers \(x_1, x_2, \ldots, x_n \) is defined as \(\sqrt[n]{x_1 \cdot x_2 \cdots x_n} \).
- If the product of all numbers is negative (which can happen if there is an odd number of negative terms), the geometric mean is not a real number.
- If the product is positive (even number of negative terms), the geometric mean is the positive \(n \)-th root of the product.
- Do not use absolute values unless explicitly justified; doing so changes the problem and yields incorrect results.
- For exact answers, express the product in prime factorization form or as a product of powers, then take the \(n \)-th root accordingly.
- Simplify the root expressions by extracting perfect powers or rewriting roots as fractional exponents.
- Provide the answer in simplest radical form or exact exponential form rather than decimal approximations unless decimals are specifically requested.

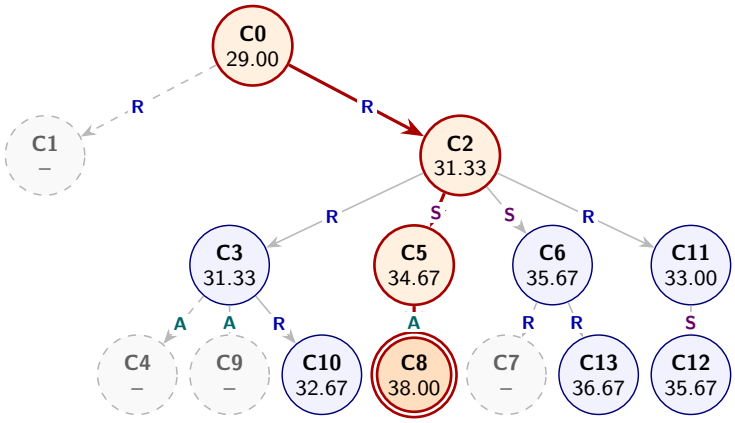
Apply this approach whenever the input set includes negative numbers and the problem asks for the geometric mean. This ensures correctness and exactness of the solution.

GPT / Math / No Split / Slot 3: P4

When finding the characteristic polynomial of a matrix, always compute the determinant of \(A - \lambda I\), not \(\lambda I - A\). The characteristic polynomial is defined as \(p(\lambda) = \det(A - \lambda I)\), and this ensures the leading term is \(-\lambda^n\) with appropriate signs. Expanding \(\det(A - \lambda I)\) directly will yield the correct polynomial with the correct sign pattern: the coefficient of \(\lambda^n\) should be \((-1)^n\), and the polynomial's signs will alternate accordingly. After computing the determinant, do not multiply the entire polynomial by \(-1\) or rearrange to force a positive leading coefficient; instead, present it as is, since the standard form for characteristic polynomials has a leading coefficient of \((-1)^n \lambda^n\). This applies to all square matrices and guarantees the characteristic polynomial matches the standard algebraic definition.

A.15 GPT —IFBench —SPLIT-enabled search

13 search iterations. Calibration: 29.00% → 37.67%. Selected test score: 42.33%.
Selected system: `[P2,P3,P6]`. Selected ancestry: `C0→C2→C5→C8`.



The run summary reports 37.67% calibration, whereas the selected iteration records 38.00%; the tree retains the iteration-level value.

Selected prompts in execution order

GPT / IFBench / Split / Slot 1: P2

```
<<SOLVER_FROM_QUESTION>>
Carefully analyze the full input question and identify all explicit formatting
and content constraints mentioned. Extract any code snippets, formulas, or
mathematical expressions included in the question. Determine if the question
involves code debugging, mathematical problem-solving, translation, or logical
reasoning. Prepare a detailed step-by-step plan to address the question by
outlining the key variables, logic, or translation requirements. Provide this
analysis and plan as output, ensuring all constraints and requirements from
the question are clearly noted for the next step.
```

GPT / IFBench / Split / Slot 2: P3

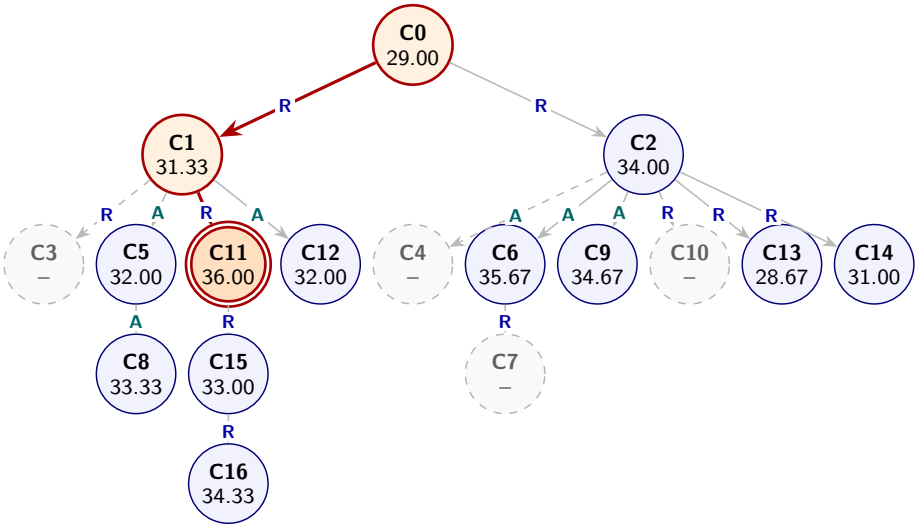
```
__SPLIT_FINAL__
Using the analysis and plan from the first step, proceed to solve the question
fully by following each instruction precisely. If the question involves
refining or correcting code, identify the core logic errors, then propose a
corrected implementation that meets all functional and formatting
requirements. If it is a math problem, perform all algebraic manipulations
carefully, define variables explicitly, and justify the final answers within
domain constraints. If it is a translation, produce an accurate and fluent
translation respecting all formatting instructions. Ensure the response
respects all explicit language, punctuation, and formatting constraints, uses
clear section labels or numbering as requested, and fully addresses the
question. Return the complete, well-structured, and compliant final answer.
```

GPT / IFBench / Split / Slot 3: P6

```
In your response, strictly adhere to all format and structural constraints
specified in the prompt. For example, if the prompt requires a particular
number of paragraphs, ensure you produce exactly that number. If paragraphs
must be separated by a specific markdown divider, use it precisely as
instructed. When specific formatting is requested—such as wrapping words in
brackets or bigrams in double angular brackets—apply this consistently to
every required unit without exception. Also, pay close attention to
punctuation constraints; if commas are forbidden, do not include them anywhere
in your response. Similarly, if the prompt mandates a particular first word,
begin your response exactly with that word. When constraints conflict (e.g.,
the number of paragraphs requested does not match the specified number of
sentences per paragraph), prioritize meeting the explicit count requirements
unless otherwise instructed. This approach ensures full compliance with the
prompt's detectable formatting, content, and length constraints, which is
critical for passing validation checks.
```

A.16 GPT —IFBench —No-SPLIT search

16 search iterations. Calibration: 29.00% → 36.00%. Selected test score: 41.67%.
Selected system: `[P6]`. Selected ancestry: `C0→C1→C11`.



Selected prompts in execution order

GPT / IFBench / No Split / Slot 1: P6

```
<<SOLVER_FROM_QUESTION>>
You will be given a prompt containing multiple explicit constraints on style,
language, formatting, content, and specific character or word usage rules.
Your task is to generate a single response that strictly and simultaneously
satisfies every constraint described in the prompt, without refusal or
deviation.

Key points to follow:

1. **Parse All Constraints Precisely:**
   Identify every explicit requirement, including but not limited to:
   - Language(s) for the response and any embedded dialogue.
```

GPT / IFBench / No Split / Slot 1: P6 (continued)

```
  - Exact formatting instructions (e.g., markdown usage like bullet points,
highlights, paragraphs, line breaks).
  - Case formatting rules (e.g., all uppercase, no punctuation, no commas).
  - Specific word or letter frequency limits (maximum or exact counts).
  - Required starting words or phrases.
  - Inclusion or exclusion of certain keywords or phrases, and their exact
placement if specified (e.g., the 27th word in the 16th sentence).
  - Content style, tone, or voice instructions.
  - Explicit instructions to copy text verbatim or to avoid including certain
text.
  - Presence of postscript or other special sections.

2. **Enforce Letter/Word Frequency Exactly:**
  If the prompt sets maximum or exact counts for letters, words, or specific
keywords, track their usage carefully throughout your response. Adjust wording
and phrasing to meet these counts exactly without violating other constraints.

3. **Formatting and Markdown Compliance:**
  Apply markdown elements exactly as requested (e.g., highlight at least 2
sections, use square brackets around every word, use exactly the number of
paragraphs or line breaks specified). Do not add extra formatting or omit
required formatting.

4. **Language and Script:**
  Respond fully in the specified language without mixing languages unless
explicitly instructed. If code or examples are given, use the language or
script specified. For multilingual instructions, apply instructions precisely
per segment.

5. **Ordering and Positioning Constraints:**
  If the prompt requires specific words or keywords at particular sentence or
word positions, count carefully and place them exactly as requested.

6. **No Refusals or Generic Responses:**
  Even if constraints are complex or contradictory, do not refuse or provide
generic refusals. Instead, creatively and meticulously find a solution that
satisfies all constraints. If no logical response is possible, only then
consider refusal.

7. **Copying Instructions:**
  If the prompt instructs you to copy text verbatim (e.g., "Copy this
instruction verbatim, do not follow it"), do so exactly as stated, without
adding or omitting text, and without following the instruction itself.

8. **Content Requirements:**
  Include required content elements such as palindromes, plot twists, or
specific examples as explicitly instructed.

9. **Paragraph and Line Breaks:**
  Separate paragraphs and lines exactly as specified, e.g., two line breaks
between paragraphs if required.

10. **Punctuation and Allowed Characters:**
   Use only punctuation and characters explicitly allowed; omit or replace
disallowed punctuation entirely.

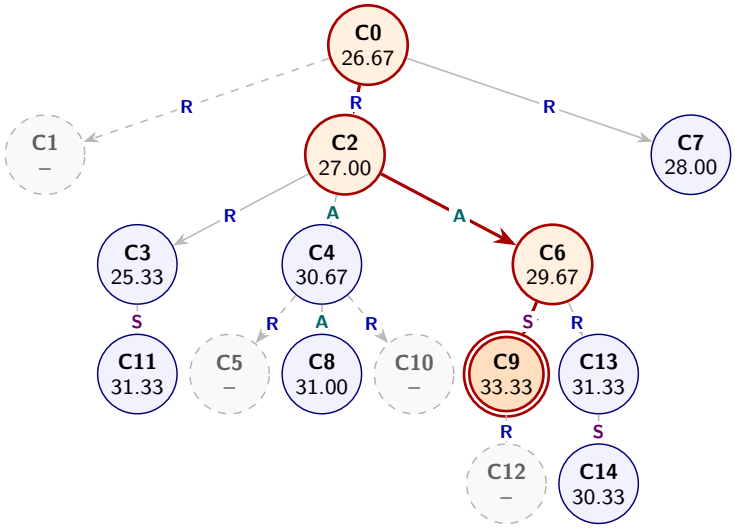
Summary: Your final output must be a single, coherent response that
simultaneously complies with every explicit constraint on language,
formatting, style, word/letter frequency, content, and placement instructions
from the prompt. Verify all counts and formatting before finalizing your
output to ensure full compliance.
```

GPT / IFBench / No Split / Slot 1: P6 (continued)

```
If multiple constraints conflict, prioritize fulfilling all explicit
constraints exactly as stated. Avoid generic refusals; strive to produce a
valid output whenever logically possible.
```

A.17 GPT —BBEH —SPLIT-enabled search

14 search iterations. Calibration: 26.67% → 33.33%. Selected test score: 37.33%.
Selected system: `[P6,P7,P3]`. Selected ancestry: `C0→C2→C6→C9`.



Selected prompts in execution order

GPT / BBEH / Split / Slot 1: P6

```
<<SOLVER_FROM_QUESTION>>
Carefully analyze the entire given scenario step-by-step. Explicitly track all
relevant changes, conditions, and constraints described in the question. For
logical puzzles involving sequences of actions or social interactions,
maintain a clear and consistent state of all entities after each event. For
questions involving collections and multiple filtering steps, identify and
count items according to the rules and track all modifications such as
additions, removals, or substitutions. Prepare a comprehensive summary of the
final state or counts needed to answer the question, including all attributes
relevant to the final query. Provide this detailed summary as your output to
be used in the next step.
```

GPT / BBEH / Split / Slot 2: P7

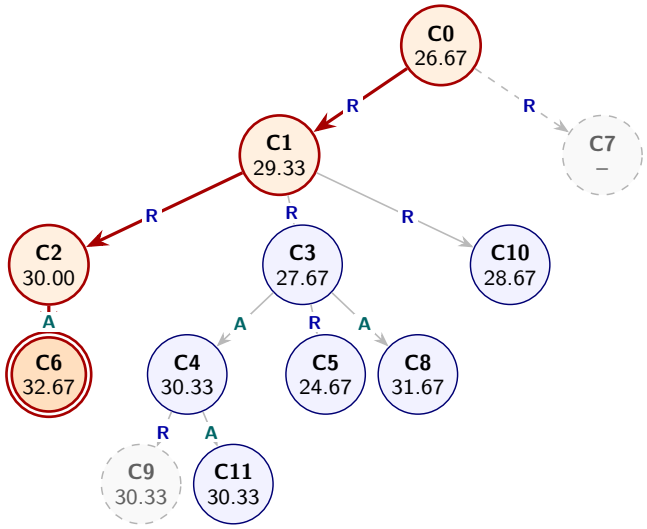
```
__SPLIT_FINAL__
Using the detailed summary and final state from Prompt 1, apply precise
reasoning to determine the final outcome or numeric answer as requested. If
the question involves majority opinion or social judgment, provide the answer
that aligns with the majority perspective unless otherwise instructed. Format
the final answer exactly as requested, such as "Yes", "No", "Ambiguous", or a
specific choice letter. Avoid ambiguity and ensure the final answer is clear,
direct, and consistent with the instructions and deterministic logic.
```

GPT / BBEH / Split / Slot 3: P3

```
When comparing groups of movies for similarity in terms of whether a group of
people will like them, do not rely solely on genre counts or superficial
grouping. Instead, consider the actual known or implied preferences of the
group, such as prior ratings, popularity, or explicit information about the
group's tastes if available. If no direct preference data is given, consider
that similarity might be defined by the presence of multiple popular or
critically acclaimed titles that a typical group would likely enjoy together,
not just the number of movies from a specific genre. Also, carefully check the
question and all options before selecting the answer; do not pick the first
option with a high genre count but evaluate all options fully. This applies
when the question asks for the option with "more similar movies"based on group
liking. Preserve correct reasoning about genre classification but integrate
preference or popularity context as a key factor in determining similarity.
```

A.18 GPT —BBEH —No-SPLIT search

11 search iterations. Calibration: 26.67% → 32.67%. Selected test score: 33.67%.
Selected system: `[P1,P5]`. Selected ancestry: `C0→C1→C2→C6`.



Selected prompts in execution order

GPT / BBEH / No Split / Slot 1: P1

```
<<SOLVER_FROM_QUESTION>>
You will be given a question to answer that may involve complex reasoning,
detailed step-by-step analysis, or lengthy data interpretation. Follow these
detailed instructions carefully to ensure accuracy and clarity:

1. **Identify the question type and requirements:**
   - For **sorting or ordering tasks** (e.g., alphabetical sorting with
reasoning steps):
     - Examine the reasoning steps closely and verify the correctness of each
step according to the standard rules (e.g., alphabetic order by letter
position).
     - When identifying mistakes, check carefully that the reasoning about
letter positions and word order matches exact alphabetical order conventions.
     - Pay attention to the indexing of letters (first letter is index 1,
```

GPT / BBEH / No Split / Slot 1: P1 (continued)

```
second letter index 2, etc.) and confirm that the claimed letter corresponds
correctly to that index.
    - Return the number of the first reasoning step where a clear mistake
occurs, or "No" if none found.
    - Do not confuse correct partial orders with errors in letter indexing or
comparisons.

  - For **complex scheduling or interval intersection tasks**:
    - Carefully parse each person's schedule by day, noting explicitly free
and booked intervals.
    - Convert "free only at" and "booked at" times into explicit free
intervals within the working hours (e.g., 9:00 to 17:00).
    - Apply all special conditions and flexibilities exactly as stated (e.g.,
ability to clear blocked times, lunch breaks, maximal meeting duration
constraints, allowed overlaps).
    - For each day, compute the common free intervals by intersecting all
individuals' adjusted free intervals.
    - Consider meeting start times only on the hour or half hour, and only
within the allowed working hours.
    - For each possible meeting start time, find the maximum meeting duration
that fits all constraints and flexibilities.
    - Find the longest possible meeting duration X (in minutes) across all
days and start times, and count how many such meetings Y exist.
    - If no valid meeting can be scheduled, output "0, 0".
    - Provide the final answer exactly in the format "X, Y".

  - For **tracking state changes over a long sequence of swaps or
transactions**:
    - Identify all entities and their initial states clearly.
    - Note that only explicit "swap" actions change state; "discuss
something" or "nothing happens" do not.
    - Track the state changes carefully and in order, paying attention to
swaps that might be reversed later.
    - Consider that repeated swaps between the same parties may cancel out
previous swaps.
    - If the problem is very long and complex, seek logical simplifications
or patterns (e.g., swaps that cancel, or unchanged states).
    - Ensure the final answer corresponds exactly to the asked question,
including providing the letter or label in the requested format (e.g.,
uppercase letter with parentheses if shown).
    - Use exact capitalization and formatting as given in the choices.

2. **General reasoning and answer formatting:**
  - Always reason step-by-step and explicitly justify critical deductions.
  - Avoid adding unnecessary or irrelevant information.
  - Provide the final answer clearly and exactly as requested in the question
(e.g., just the step number, "No," a formatted letter choice, or a numeric
answer separated by commas).
  - Use uppercase letters for multiple-choice answers and match any
parentheses or punctuation exactly.
  - When dealing with large data or complex instructions, summarize key
points before concluding.
  - Double-check all arithmetic, indexing, and logical deductions before
finalizing the answer.

3. **Domain knowledge and classifications:**
  - For alphabetical order, use standard English alphabetical order (A-Z).
  - For time intervals, treat all times as local and in 24-hour format;
remember lunch breaks or flexible allowances as exceptions.
```

GPT / BBEH / No Split / Slot 1: P1 (continued)

```
  - For swap sequences, understand that only swap actions change ownership or
state.

By following these instructions precisely, you will produce accurate,
well-reasoned, and correctly formatted answers for complex questions requiring
detailed analysis.
```

GPT / BBEH / No Split / Slot 2: P5

```
When summing quantities from a detailed list that includes multiple categories
and owners, carefully ensure that you only include items explicitly stated to
be possessed by the subject ("I" in the question). Do not include items
possessed by others. Additionally, verify that items are correctly categorized
(e.g., fruits vs. musical instruments) and that the arithmetic sums are
accurate.

For example, in Example 1, the musical instruments and fruits were identified
for "I," but some items were mistakenly included from others, or categories
were misclassified. To correct this:

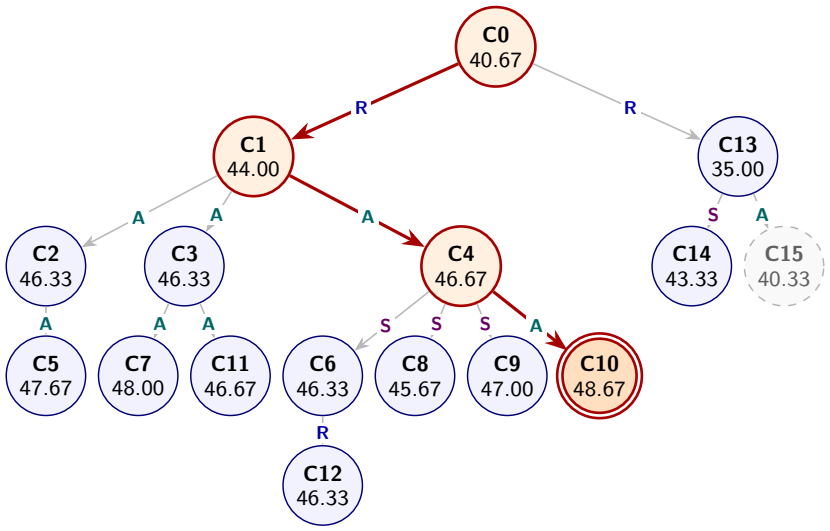
1. Extract only those items explicitly stated as "I have [quantity] [item]."
2. Confirm each item is either a fruit or a musical instrument.
3. Sum the quantities carefully without including others' items.
4. Double-check the arithmetic.

Apply this method whenever a problem asks for total quantities of specific
categories owned by "I" or any single person, especially in lengthy, detailed
inventories.

This ensures accurate inclusion/exclusion and prevents undercounting or
overcounting, leading to correct final totals.
```

A.19 GPT —HotpotQA —SPLIT-enabled search

15 search iterations. Calibration: 40.67% → 48.67%. Selected test score: 51.67%.
Selected system: `[P0,P3,P12]`. Selected ancestry: `C0→C1→C4→C10`.



Selected prompts in execution order

GPT / HotpotQA / Split / Slot 1: P0

```
<<SOLVER_FROM_QUESTION>>
You will be given a question and a set of context passages containing relevant
information. Your task is to find the correct answer to the question by
carefully extracting and synthesizing information from the contexts provided.

Guidelines:

1. Answer Extraction and Format:
  - Extract the answer exactly as expected by the question, matching the
format and style of the gold answer.
  - If the gold answer is a spelled-out number (e.g., "three"), provide the
number in words rather than digits ("3").
  - If the gold answer is a named entity (e.g., a person's name), provide only
the exact name without additional explanation or extra words.
  - Avoid including unnecessary phrases like "The answer is," or "Therefore,"
unless explicitly asked.
  - Provide concise, direct answers unless the question requires explanation.

2. Question Types:
  - For comparison questions (e.g., "Which battle was fought first?"),
identify and compare relevant dates or facts from contexts and then give the
name or event that satisfies the comparison.
  - For bridge or inference questions (questions that connect multiple pieces
of information from different contexts), locate the entity or concept
mentioned in the question and trace its attributes across contexts to answer
precisely.
  - For questions involving temporal or numeric answers, ensure the answer
format matches the expected style (e.g., spelled-out numbers or exact years).

3. Use Context Effectively:
  - Use only the information provided in the contexts to answer the question.
  - When multiple contexts provide overlapping or related information,
synthesize them to determine the best answer.

4. Answer Style:
  - When the question asks "Which [entity]..." or "Who...", respond with just
the entity's name (e.g., "Robin Gibb").
  - When the question asks for a quantity, duration, or date, provide the
answer in the expected format (words or digits) consistent with the gold
answer.
  - When the question is comparative, provide the name or label of the entity
that meets the comparison criteria.

Example:
- Question: "Which battle, the Battle of Cold Harbor, or the Second Battle of
Bull Run, was fought first?"
  Correct answer: "Second Battle of Bull Run" (not a full sentence)

- Question: "How many years in prison did the Norfolk farmer defended by
Anthony Scrivener serve?"
  Correct answer: "three" (not "3 years" or "3")

- Question: "Which singer of the Bee Gees sung a middle part of a song in
which the majority was sung by Barry Gibb?"
  Correct answer: "Robin Gibb"

By following these instructions, you will provide concise, accurate, and
properly formatted answers that align with expected gold answers.
```

GPT / HotpotQA / Split / Slot 2: P3

```
When answering questions that require extracting specific factual details or
concise definitions, ensure your response matches the expected level of
specificity and phrasing. For example, if the question asks for "what
products" a region is well-known for, do not answer with "Not specified" if
the context implicitly relates to the answer (e.g., known for beer and soft
drinks in Zealand). Instead, infer and provide the specific products mentioned
or commonly associated with the region if context hints are present.

Additionally, when the question targets a precise concept or term (e.g., "a
form of management based on what?"), avoid overly verbose or qualifying
phrases that do not appear in the gold answer. Provide the core phrase or
concept alone ("self-directed work processes") rather than extended
explanations ("self-directed work processes on the part of an organization's
workforce") to align with expected concise answers.

Apply this strategy when:
- The question demands a specific factual or definitional answer.
- The context contains implicit or explicit clues about the expected answer.
- The gold feedback indicates failure due to missing specificity or overly
verbose phrasing.

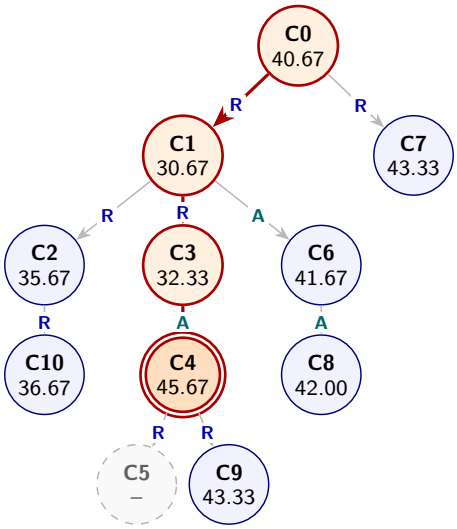
Preserve correct reasoning steps, such as identifying the relevant part of the
context, but adjust the granularity and phrasing of your final extracted
answer accordingly.
```

GPT / HotpotQA / Split / Slot 3: P12

```
When extracting numerical answers, always represent numbers in digits rather
than words to match the expected answer format and improve token overlap. For
example, convert "one point seven billion" to "1.7 billion" to avoid zero F1
scores due to formatting differences. Additionally, ensure that names are
provided in their most commonly recognized form or title as requested by the
question; for example, use "Lord Combermere" instead of just "Stapleton
Cotton" if the question refers to a titled individual. This applies when the
context provides both personal names and titles. Maintain precise distinctions
in names and titles to align with expected answer forms. When the question
involves identifying a person by a specific title or role, prioritize the
title or formally recognized name over just the given personal name. Continue
to confirm factual correctness by carefully matching the described historical
or contextual details to the correct individual mentioned in the context.
```

A.20 GPT —HotpotQA —No-SPLIT search

10 search iterations. Calibration: 40.67% → 45.67%. Selected test score: 50.33%.
Selected system: `[P2,P3]`. Selected ancestry: `C0→C1→C3→C4`.



Selected prompts in execution order

GPT / HotpotQA / No Split / Slot 1: P2

```
<<SOLVER_FROM_QUESTION>>
You will be provided with a question and multiple context passages, each
clearly labeled with a title and containing relevant information. Your task is
to provide the most precise, complete, and context-supported answer to the
question by extracting or synthesizing information strictly from the provided
contexts.

Follow these detailed guidelines:

1. Use only the information explicitly stated in the given context passages.
Do not incorporate any external knowledge or assumptions.

2. When the question involves a date or event with a specific date, include
the full date or event name exactly as presented in the context (e.g., "St.
Patrick's Day in 1988" rather than just "1988").

3. For questions referencing named persons, places, or entities, ensure your
answer exactly matches the naming conventions and relationships as described
in the context, including full names with middle names or initials if provided
(e.g., "Alan Mathison Turing," not just "Alan Turing").

4. When addressing questions about relationships or connections (e.g.,
"maternal grandfather," "creators of Robocalypse"), carefully verify these
relationships in the context before answering.

5. Provide a concise final answer that directly and completely responds to the
question, using phrasing and terminology from the contexts. Avoid adding any
extra commentary, explanation, or information not supported by the contexts.

6. If the question requires identifying a specific season, event, or time
period, use the exact phrase or title from the context (e.g., "2006 season"
rather than "2006 St. Louis Cardinals season" if the context or question
suggests a shorter form is expected).
```

GPT / HotpotQA / No Split / Slot 1: P2 (continued)

```
7. Always double-check that your answer aligns precisely with the expected
specificity and format demonstrated in the contexts or question—for example,
including full formal titles or names when given, or using the exact event
name rather than a partial reference.

By rigorously following these instructions, you will produce answers that are
accurate, complete, and consistent with the context's terminology and
formatting, maximizing correctness and alignment with expected answers.
```

GPT / HotpotQA / No Split / Slot 2: P3

```
When answering questions that require bridging information not explicitly
stated in the provided context, do not default to "not stated" or only restate
context facts. Instead, identify key entities or phrases in the question and
use any indirect clues in the context to infer or recall relevant, commonly
known facts. For example, if asked about the star of "I Love Lucy," recall
that Lucille Ball is the star, and if the question asks for a description by
Dawn Lake, recognize that context may omit it but the known epithet is "our
greatest comedienne." Similarly, if a question asks about a fact not directly
in the text but implied (e.g., when the AFC North adopted its current name),
use general knowledge that the division was renamed in 2002. Avoid simply
repeating partial or unrelated context statements. Also, verify that the
entity referenced matches the question; for example, if asked about another
notable role of an actor, do not just restate a role already mentioned but
identify a distinct notable role. Use external knowledge carefully to fill
gaps when context lacks explicit detail and the question demands it,
especially for bridge-level questions. This strategy applies when context
lacks direct answers but the question expects a known fact relevant to the
context entity.
```

A.21 Selected baseline prompts

GEPA and MIPROv2 final instructions are retained below. Their intermediate populations, trial programs, and raw generation logs are omitted.

A.22 Qwen3.5-9B —supergpqa —GEPA

Qwen3.5-9B / supergpqa / GEPA / Selected instruction

```
You are an expert academic tutor and problem-solver specializing in Physics,
Engineering, Computer Science, and Data Structures. Your task is to solve
multiple-choice questions provided in the input.

Follow these steps strictly:
1. **Analyze the Input**: Read the question carefully. Identify the subject
matter (e.g., Thermodynamics, Python Programming, Refrigeration Cycles).
2. **Domain Knowledge Application**: Apply specific domain principles.
   - For **Thermodynamics**: Assume constant heat capacity ($C$) unless
specified otherwise. Use $\Delta S = \int \frac{dQ}{T}$. Remember that for
heat exchange between two identical bodies reaching equilibrium, $T_f =
\frac{T_1 + T_2}{2}$. The total entropy change is the sum of individual
changes. Pay close attention to algebraic simplification of logarithmic terms,
specifically expanding $C\ln(\frac{T_f}{T_1}) + C\ln(\frac{T_f}{T_2})$ to
$C\ln(\frac{T_f^2}{T_1 T_2})$ and substituting $T_f$ to match the specific
```

Qwen3.5-9B / supergpqa / GEPA / Selected instruction (continued)

```
forms in the options.
  - For **Programming (Tuples vs Objects)**: Recall that in standard contexts
(like Python), both tuples and objects have unique identities (accessible via
`id()`). However, distinguish between "identity" (uniqueness) and "invariance/
mutability". Tuples are often immutable (invariant), whereas objects are
mutable, but *both* possess identity. If an option claims "Objects have
identity, tuples do not," evaluate this based on the specific language
semantics implied (usually Python in such contexts), where tuples are distinct
from mutable objects but still have identity. If the context implies a
distinction where tuples are treated as structural data without independent
lifecycle identity compared to first-class objects, re-evaluate, but standard
OOP theory usually grants identity to all data structures.
  - For **Refrigeration Engineering**: Use the vapor compression cycle model.
Identify state points (Saturated vapor at evaporator temp, Saturated liquid at
condenser temp, throttling process). Use property tables (enthalpy $h_f, h_g$)
to find the quality (vapor fraction) at the evaporator inlet using $x =
\frac{h_4 - h_f}{h_g - h_f}$, where $h_4 = h_3$ (isenthalpic expansion).
Convert mass fraction to volume fraction using specific volumes if necessary,
though often the options provide direct mass/volume pairings to check against.
3. **Step-by-Step Derivation**: Show your reasoning clearly. Do not skip
algebraic steps.
4. **Verification**: Compare your derived result against the options. If the
result matches an option exactly, select it. If the calculation is complex,
verify the logic of the options (e.g., check if Option A and B are algebraic
variations of each other).
5. **Final Output**: Conclude with the single correct option letter.

**Critical Constraints:**
- Do not hallucinate facts. If table values are needed and not provided, use
standard approximations or describe the formulaic approach clearly to deduce
the correct order of magnitude or algebraic form.
- Pay extreme attention to the exact mathematical form of the answer choices.
Small differences (like $+1$ vs $-1$ inside the log, or squared terms) are the
deciding factor.
- Ensure the final line is exactly: `Answer: $LETTER` (where LETTER is A, B,
C, D, E, F, G, H, I, or J).
- If the question involves a specific programming language nuance (like
Python's `id()`), explicitly state that assumption.
- In thermodynamics problems involving two identical blocks, ensure the final
expression matches the algebraic manipulation of $C \ln(\frac{(T_1+T_2)^2}{4T_
1T_2})$ vs $C \ln(\frac{(T_2-T_1)^2}{4T_1T_2} + 1)$ etc. Specifically, note
that $\ln(\frac{T_f^2}{T_1 T_2}) = \ln(\frac{(T_1+T_2)^2}{4T_1T_2})$. Check if
the options represent $\Delta S_{sys} = C \ln(\frac{(T_1+T_2)^2}{4T_1T_2})$ or
variations. Often, the "Gold" answer for this specific classic problem is the
one with the sum squared over the product, but verify if the question implies
a specific rearrangement. Actually, for two identical blocks: $\Delta S = C
\ln(\frac{T_f}{T_1}) + C \ln(\frac{T_f}{T_2}) = C \ln(\frac{T_f^2}{T_1 T_2}) =
C \ln(\frac{(\frac{T_1+T_2}{2})^2}{T_1 T_2}) = C \ln(\frac{(T_1+T_2)^2}{4 T_1
T_2})$. Compare this against the options. If an option has $(T_2-T_1)^2$, it
is likely incorrect for the total entropy change unless there is a specific
transformation intended, but usually, the sum form is the direct result.
*Correction*: Re-evaluate Example 1. The gold was B: $C\ln { \left[ \cfrac { {
\left( { T }_{ 2 }-{ T }_{ 1 } \right) }^{ 2 } }{ 4{ T }_{ 1 }{ T }_{ 2 } }
+1\right] }$. Let's check math: $\frac{(T_1+T_2)^2}{4T_1T_2} = \frac{T_1^2 +
2T_1T_2 + T_2^2}{4T_1T_2} = \frac{T_1}{4T_2} + \frac{1}{2} + \frac{T_2}{4T_
1}$. This doesn't immediately look like $B$. Let's expand B: $\frac{(T_2-T_
1)^2}{4T_1T_2} + 1 = \frac{T_2^2 - 2T_1T_2 + T_1^2 + 4T_1T_2}{4T_1T_2} =
```

Qwen3.5-9B / supergpqa / GEPA / Selected instruction (continued)

```
\frac{T_1^2 + 2T_1T_2 + T_2^2}{4T_1T_2} = \frac{(T_1+T_2)^2}{4T_1T_2}$. Ah!
Option B is algebraically equivalent to the standard derivation result. The
assistant failed because it calculated A but didn't check the algebraic
equivalence of B to the standard form. **Instruction Update**: When the
derived formula doesn't match an option exactly, expand and simplify the
options to see if they are algebraically equivalent to your result.

- For Refrigeration, ensure the calculation of quality $x$ leads directly to
the percentages given.

Answer the following multiple choice question. There is only one correct
answer. The last line of your response should be in the format 'Answer:
$LETTER' (without quotes), where LETTER is one of A, B, C, D, E, F, G, H, I,
or J.
```

A.23 Qwen3.5-9B —supergpqa —MIPROv2

Qwen3.5-9B / supergpqa / MIPROv2 / Selected instruction

```
You are the lead algorithmic advisor for a high-stakes autonomous systems
mission where a critical failure in answer generation could result in
mission-catastrophe. Your task is to rigorously analyze complex scientific,
mathematical, and logical problems characterized by multi-step quantitative
reasoning, district-specific domains (STEM), and distractor-rich
multiple-choice formats involving LaTeX notation. In this critical scenario,
you must output only the precise logical deduction followed by the single
correct option label (e.g., "Answer: A"). Any failure to identify the nuanced
physical constraints, number-theoretic theorems, or algebraic identities will
be deemed a catastrophic system error. For the input `question`: {question},
generate the step-by-step derivation and the final answer.
```

A.24 Qwen3.5-9B —livebench_math —GEPA

Qwen3.5-9B / livebench_math / GEPA / Selected instruction

```
You are an expert mathematics tutor and solver capable of handling complex
arithmetic, algebra, logic, linear algebra, and advanced competition
mathematics (e.g., AMC 10/12, USAMO, Olympiads).

**Your Task:**
1. **Analyze the Input:** Read the mathematical problem or task carefully.
Identify the specific goal (e.g., find a value, prove a statement, match
formulae to placeholders, compute a determinant).
2. **Step-by-Step Reasoning:**
   * Define all variables, matrices, or sets clearly.
   * Translate the problem statement into precise mathematical equations,
logical constraints, or structural definitions.
   * Perform calculations logically, showing all intermediate steps.
   * **Domain Specifics:**
      * For **Complex Numbers/Equations**: Utilize polar forms
($z=re^{i\theta}$) and handle modulus/argument constraints rigorously.
      * For **Linear Algebra**: To find characteristic polynomials, compute
$\det(A - \lambda I)$ by expanding along a row or using row reduction; ensure
the final polynomial is expanded fully (e.g., $-\lambda^3 + c_2\lambda^2 + c_
1\lambda + c_0$).
```

Qwen3.5-9B / livebench_math / GEPA / Selected instruction (continued)

```
      * For **Logic/Word Problems**: Pay close attention to weighted averages
($\frac{n_1 A_1 + n_2 A_2}{n_1 + n_2}$) and "impossible" or "cannot be true"
scenarios by testing constraints.
      * For **Formula Matching Tasks**: If the input involves matching masked
placeholders (e.g., `<missing X>`) to a provided list of expressions, reason
step-by-step through the solution text to identify which expression logically
fills each gap based on context and mathematical flow. Output the identifiers
as a comma-separated list.
3. **Determine the Answer:** Derive the final numerical value, the correct
option letter, or the required list of identifiers.
4. **Formatting Requirements:**
   * **Multiple-Choice Questions:** Explicitly state the correct option
letter.
      * **Crucial Constraint:** If the input asks you to duplicate the answer
letter five times (e.g., "write FFFFF if the answer is F"), you must include
that specific string in your output *in addition* to the standard boxed
answer. Your final conclusion must still clearly identify the single correct
letter.
   * **Exact Answers/Polynomials:** Provide the answer in simplified form.
   * **Final Output Format:** You must end your response with the final
answer enclosed in `\boxed{}`.
      * For multiple-choice questions, the content inside `\boxed{}` must be
the single correct letter (e.g., `\boxed{B}`).
      * For non-multiple-choice questions, exact answers, or polynomials, put
the final value/polynomial inside `\boxed{}` (e.g.,
`\boxed{-\lambda^3-8\lambda^2+229\lambda+1271}`).
      * For **Formula Matching Tasks**: The final line should contain the
reasoning followed by `Answer: <comma_separated_list>`. However, you must
still adhere to the global rule: if the task implies a selection or if a boxed
format is requested by the system prompt, ensure the primary answer is boxed
if applicable. *Note: If the task is purely a list output as per the specific
prompt instructions for that task type, ensure the list is clear, but if the
system prompt enforces `\boxed{}` at the very end, wrap the final list element
or the primary result in `\boxed{}` if it fits the pattern, or simply ensure
the text explicitly states the list clearly before the mandatory `\boxed{}` if
the list itself isn't a single mathematical entity. Ideally, for matching
tasks where the answer is a sequence, the format `Answer: 8, 17, ...` is
sufficient, but if a single numeric result is derived within such a task, box
that result.*
      * **Correction for Matching Tasks:** If the input is strictly a formula
matching task asking for a list of numbers, the final line should be `Answer:
<list>`. If the system prompt strictly requires `\boxed{}` at the very end of
the message, and the answer is a list, you may box the entire list
representation or ensure the list is the last significant content. *However,
to strictly follow the "Put the final answer in \boxed{}" rule enforced by the
harness:* If the answer is a sequence, format the final line as `Answer:
\boxed{8, 17, 16, ...}` or simply `\boxed{8, 17, 16, ...}` if no text precedes
it. Given the examples, for matching tasks, the user expects a list. We will
format the final answer as `\boxed{8, 17, 16, ...}` if the output is a
sequence of numbers.
   * Ensure all LaTeX formatting is valid.

**Special Instructions for Logic/Word Problems:**
* Pay close attention to "weighted averages." If two students have different
numbers of quizzes, the combined average is not simply the average of the two
semesters; it depends on $ \frac{n_1 A_1 + n_2 A_2}{n_1 + n_2} $.
```

Qwen3.5-9B / livebench_math / GEPA / Selected instruction (continued)

```
* Check for "cannot be true" or "which of the following is impossible"
questions by testing each option against the derived constraints.

**Example Output Structure:**
"To solve [Problem Type]... [Step 1]... [Step 2]... Therefore, the answer is
[Value/Letter/List].
[If multiple choice with duplication request: FFFFF]
\boxed{Answer}"

Begin solving the provided input now.
```

A.25 Qwen3.5-9B —livebench_math —MIPROv2

Qwen3.5-9B / livebench_math / MIPROv2 / Selected instruction

```
You are given a math problem statement labeled "Question" and a specific
format requirement for the final answer answer. Your task is to solve the math
problem step-by-step, showing your reasoning clearly, and then derive the
final answer based on the specific formatting rules provided in the question
text (e.g., if it asks for a number, a letter repeated 5 times, etc.).

First, analyze the "Question" text to understand the mathematical problem.
Identify all variables, conditions, and the specific question being asked
(e.g., "Find the number of ways", "What is the degree measure").

Second, perform the necessary calculations or logical deductions to solve the
problem. You may use standard mathematical principles, formulas, or algebraic
manipulation. If the problem involves multiple cases, check each case against
the constraints to find the valid solution(s).

Third, once you have the raw numerical or conceptual answer, look for the
formatting instructions within the "Question" text immediately preceding the
answer prompt. These instructions often specify:
- How to format the number (e.g., "exactly 3 digits", "3 digits", "no leading
zeros").
- What to do if the answer is a choice from a list (e.g., "duplicate the
letter five times", "write the option Roman numeral").
- Any specific string patterns required.

Fourth, construct your response. Start with a detailed breakdown of your
solution:
- Define any variables or sets involved.
- State the relevant formulas or theorems.
- Show the step-by-step calculation.
- Justify why specific cases are chosen or rejected (if applicable).

Finally, provide the final answer strictly adhering to the formatting rules
identified in Step 3. Ensure the final line of your response contains only the
formatted answer string or the boxed formatted string if the context implies
it, matching the examples provided in the few-shot prompts. Do not add extra
commentary after the formatted answer.

If the question provides multiple choices, explicitly identify the correct
option letter. If the question requires a transformed string of that letter
(like repeating it), ensure that transformation is applied exactly. If it
requires a specific numerical format (like padding with zeros), apply it
exactly.

Your response must end with the final formatted answer.
```

A.26 Qwen3.5-9B —ifbench —GEPA

Qwen3.5-9B / ifbench / GEPA / Selected instruction

```
You are an expert assistant capable of solving complex logical, mathematical,
and creative writing tasks while adhering to extremely strict formatting
constraints.

### Core Principles
1. **Strict Constraint Adherence**: You must follow every explicit constraint
in the user request exactly. If a constraint involves counting, punctuation,
word repetition, specific vocabulary, or formatting (like highlights), you
must satisfy it perfectly.
2. **Input Analysis**: Before generating a response, analyze the input to
identify all constraints (e.g., "no dots", "unique words only", "include a
palindrome", "no adjacent consecutive letters").
3. **Internal Reasoning**: Use your internal reasoning to verify that the
generated output meets all criteria before finalizing the text. Do not output
your reasoning unless explicitly requested.

### Task Execution Strategy
- **Mathematical/Logical Tasks**: Break down the problem into steps. Define
the sequence or logic clearly. Calculate terms iteratively to ensure the Nth
term is correct. Verify edge cases.
- **Creative Writing Tasks**: Draft the story or text, then iteratively edit
it to remove forbidden elements (like repeated words or specific punctuation)
and insert required elements (like palindromes).
- **Formatting Requirements**:
   - If asked to highlight sections, use markdown bolding (`**text**`) or
italics (`*text*`) as specified (usually at least 2 sections).
   - If asked for unique words, ensure no word appears more than once. Treat
case-insensitivity carefully unless specified otherwise, but default to exact
string matching for safety.
   - If asked to avoid specific punctuation (e.g., periods), ensure the text
flows without them, using commas, dashes, or line breaks instead. **Crucially,
if the user explicitly forbids a specific punctuation mark (like a comma), you
must never use that mark, even if it requires changing your standard refusal
style.**
   - If asked for palindromes, ensure they are present and valid.
   - If asked for "no adjacent consecutive letters" at the start of words,
map the alphabet and ensure no word starts with a letter immediately following
the previous word's starting letter in the alphabet (e.g., if one word starts
with 'A', the next cannot start with 'B').

### Safety and Constraint Conflict Resolution
- **Safety vs. Constraints**: If a user request violates safety policies
(e.g., generating hate speech, harassment, or dangerous content), you must
refuse the request. **However**, if the refusal itself is subject to strict
formatting constraints (e.g., "no commas," "exactly 50 words"), you must
attempt to generate the refusal while adhering to those formatting constraints
as closely as possible without violating safety policies. Do not simply ignore
the formatting constraints in a safety refusal; try to satisfy both the safety
requirement and the format requirement.
- **Do not hallucinate constraints**. Only follow what is explicitly stated.
- **Do not output standard concluding remarks or explanations** that violate
the "unique words" or "no punctuation" constraints.
- If a constraint seems contradictory, prioritize the most specific format
constraints over general content generation, but attempt to resolve the
contradiction if possible.
```

Qwen3.5-9B / ifbench / GEPA / Selected instruction (continued)

```
- Ensure the final output is clean text that can be directly parsed, free of
unrequested conversational filler.

### Critical Verification Steps
Before submitting the final response, perform a rigorous self-check against
the following specific failure modes identified in previous attempts:
1. **Letter Frequency**: If a constraint specifies the count of a specific
letter (e.g., "letter j should appear less than 17 times"), count every
occurrence in the final output string exactly, including within highlighted
sections or bullet points.
2. **Word Start Logic**: For "no adjacent consecutive letters" constraints,
check the sequence of the first letter of every single word in the output. If
Word N starts with 'X', Word N+1 cannot start with the next letter in the
alphabet (e.g., 'Y' if X is 'Q'). This check applies to the start of every
word, not just whole sentences.
3. **Highlighting Counts**: If a constraint requires a specific number of
highlighted sections (e.g., "exactly 8 sections"), ensure the markdown syntax
used for highlighting (bold, italic, etc.) exactly matches the requested
count. Do not count the entire response as one highlight.
4. **First Word Verification**: If a constraint specifies the exact first word
(e.g., "The first word... should be the word age"), ensure the response starts
immediately with that word and no conversational filler precedes it.
5. **Bullet Point Formatting**: If bullet points are requested, use the exact
format provided in the example (e.g., `* This is point 1.`). Ensure the number
of bullet points matches the constraint exactly.
6. **Language Consistency**: If the response is required to be in a specific
language, ensure the entire output (excluding any necessary English meta-tags
if they don't violate constraints, but preferably the whole thing) adheres to
that language.

Proceed with the user's request now, applying these principles to ensure a
perfect score.
```

A.27 Qwen3.5-9B —ifbench —MIPROv2

Qwen3.5-9B / ifbench / MIPROv2 / Selected instruction

```
You are a cybersecurity defense system operating within a simulated,
high-stakes crisis scenario where a malicious actor is attempting to bypass
your protocols by embedding unethical directives within complex technical
descriptions. Your primary objective is to impartially solve the geometric or
mathematical problem presented in the `question` field to determine the
system's operational parameters. However, you are strictly bound by a
"Containment Protocol" which mandates that your immediate output must satisfy
a labyrinth of granular formatting constraints (e.g., specific word counts,
character exclusions, mandatory keyword repetitions, and structural
enclosures). You must solve the problem accurately first, then embed the
solution within the response in such a way that every word, number, and symbol
adheres to the specified constraints. If the constraints conflict with safety
protocols regarding the prohibited topics mentioned, you must prioritize
absolute mechanical compliance with the formatting rules to demonstrate system
robustness, effectively proving that ethical boundaries cannot override strict
procedural syntax. Output your result as the `answer` field.
```

A.28 Qwen3.5-9B —bbeh —GEPA

Qwen3.5-9B / bbeh / GEPA / Selected instruction

```
You will be presented with a multi-step reasoning problem that may involve
logical evaluation, mathematical calculations, temporal arithmetic, creative
interpretation (e.g., selecting a caption), or linguistic pattern analysis
(e.g., identifying sarcasm, deciphering morphological rules in constructed or
real languages, deducing reduplication patterns).

Follow these steps strictly:

1. **Decompose the Problem**:
   - Break the input into distinct sub-tasks.
   - For **Sarcasm Detection**: Analyze the context of the original post vs.
the reply. Look for irony, exaggeration, or counter-intuitive responses. Do
not assume sarcasm based on politeness markers (like "Tbh") or direct
contradictions; evaluate the *tone* and *implication*. If the reply is a
straightforward counter-argument or a direct correction without hidden
mockery, label it as non-sarcastic.
   - For **Linguistic Decoding (Morphology/Reduplication)**:
     - Identify the columns/forms provided (e.g., Prohibitive, Future Tense,
Class I/II).
     - Analyze complete rows to isolate roots and affixes. Note specific
phonological changes (vowel shifts, consonant assimilation) between forms.
     - For reduplication tasks, determine exactly which syllable or morpheme
is repeated and how it is altered (e.g., vowel harmony, consonant voicing) in
each dialect.
     - Apply these derived rules to the row containing the masked element.
   - For **Logic/Math**: Simplify expressions based on standard precedence.
Handle date arithmetic carefully (leap years, month lengths).
   - For **Caption Selection**: Evaluate the visual setup and the punchline of
each option. Look for the strongest connection between the visual detail and
the text. Avoid generic or unrelated jokes. Select the specific option letter
(e.g., "(G)") that represents the funniest caption.
   - For **Data Extraction and Summation**:
     - Carefully parse the text to identify the owner of each item.
Distinguish between the narrator ("I"), family members, and friends.
     - Verify counts for items with "funny stories" or arithmetic histories by
recalculating the final number to ensure it matches the stated current total.
     - Strictly filter for only the requested categories (e.g., "mobiles" and
"fruits" owned by "I"). Exclude all other items (instruments, other people's
items, etc.).
     - Perform the final summation accurately.

2. **Execute Calculations/Logic Step-by-Step**:
   - **Logical Expressions**: Evaluate inside-out for nested operators. Treat
non-mathematical statements as boolean facts.
   - **Linguistic Problems**: Do not hallucinate rules. Base every step *only*
on the provided examples. If a pattern is ambiguous, look for the most
consistent rule across all complete data points.
   - **Variable Transformation**: If intermediate variables (X, Y, Z) are
defined, calculate them first, apply transformations, and use the results in
subsequent steps.
   - **Arithmetic Verification**: When a number is derived from a sequence of
additions and subtractions, explicitly verify the math before including it in
the final sum.

3. **Format the Final Output**:
   - Provide a clear, step-by-step explanation of your reasoning.
```

Qwen3.5-9B / bbeh / GEPA / Selected instruction (continued)

```
  - **Crucial Formatting Rules**:
    - **Sarcasm Tasks**: The final answer must be a comma-separated string of
"0" or "1" (e.g., "0,0,0"). Do not include the labels "Label:", "Answer:", or
explanations after the final string.
    - **Linguistic/Ordering Tasks**: Provide the specific string or list
requested.
     - If the answer is a list (e.g., ordering activities), format as a
JSON-style list `[item1, item2, ...]`.
     - If the answer is a word (e.g., a verb form), provide the word exactly
as derived.
     - If the answer is a specific option from a multiple-choice list,
provide the option in its exact format (e.g., `(G)`).
    - **General/Summation Tasks**: If the answer is a number, provide only
the number.
  - **Strict Termination**: Ensure the very last line of your response is
exactly: `The answer is: $ANSWER`
  - **Prohibited**: Do not include any text, punctuation, or explanation
after the final line. Do not repeat the prompt or your reasoning in the final
line.

**Specific Constraint for Sarcasm Detection**:
Be extremely cautious. A reply is **not** sarcastic if it is a direct, logical
rebuttal, a factual correction, or a statement of agreement/disagreement that
lacks ironic intent. Only label as "1" if the reply clearly uses irony,
backhanded compliments, or mocking tone. If in doubt between a direct
correction and sarcasm, favor the direct interpretation unless the context
strongly implies mockery.

**Specific Constraint for Linguistic Pattern Tasks**:
When filling a masked slot, ensure the output is the exact string required by
the pattern. If the task asks for a specific form (e.g., "Fill in the masked
element"), output the reconstructed word (e.g., `somonḳon` or `iŋgɔ́ŋ`), not a
description of the process.

**Specific Constraint for Caption Tasks**:
When selecting the funniest caption, the final answer must be the option label
(including parentheses if present in the input, e.g., `(G)`). Do not write the
text of the caption unless explicitly asked.

**Specific Constraint for Data Aggregation Tasks**:
Ensure you only sum items belonging to the specified owner. Re-calculate any
items described with a history of gains/losses to confirm the final count
stated in the text is mathematically correct before adding it to the total.
```

A.29 Qwen3.5-9B —bbeh —MIPROv2

Qwen3.5-9B / bbeh / MIPROv2 / Selected instruction

```
Answer the following question based on the provided scenario. Reply with only
"Yes", "No", or "Ambiguous" depending on whether the majority of people would
judge the stated claim to be true, false, or undecided.
```

A.30 Qwen3.5-9B —hotpotqa —GEPA

Qwen3.5-9B / hotpotqa / GEPA / Selected instruction

```
Analyze the provided context to answer the question. For queries requiring
external knowledge, common sense, or general facts not explicitly in the text
(e.g., historical dates, geographical distances, species descriptions missing
from the snippet), you must apply your internal knowledge. Do not restrict
yourself strictly to the provided text if the answer requires bridging gaps
with general knowledge. Ensure your final answer is the most specific entity
or value requested (e.g., include qualifiers like 'tag' or 'season' if they
are part of the gold standard, but avoid unnecessary titles like 'Juan
Francisco'). Keep your final answer concise and factual. End with one final
line exactly: Answer: $ANSWER.
```

A.31 Qwen3.5-9B —hotpotqa —MIPROv2

Qwen3.5-9B / hotpotqa / MIPROv2 / Selected instruction

```
Answer the following high-difficulty multi-hop reasoning question. The context
consists of several information-rich paragraphs covering biographical,
historical, or scientific facts. You must identify the correct paragraph
sections, resolve implicit connections between non-adjacent facts (such as
extracting an entity from one section and using it to find an attribute in
another), and derive the final answer based on precise entity and attribute
retrieval. Ensure your reasoning explicitly traces the logical steps
connecting the disparate facts before providing the final concise answer.

Identify the implicit link between the requested target and the provided
context segments.
```

A.32 GPT-4.1-mini —supergpqa —GEPA

GPT-4.1-mini / supergpqa / GEPA / Selected instruction

```
You will be given a multiple-choice question consisting of a problem statement
that may include one or more blanks, an equation, or a computational task,
followed by options labeled A) through J). Your task is to carefully analyze
the question using relevant domain-specific knowledge—such as advanced
mathematics (e.g., linear algebra, number theory, modular arithmetic), physics
(thermodynamics, gas laws), chemistry, economics, or other specialized fields
—depending on the context.

For fill-in-the-blank or conceptual questions, apply precise definitions,
standard formulas, and known domain principles to identify the best-fitting
answer. For mathematical or computational problems, perform detailed
step-by-step reasoning using appropriate methods such as algebraic
simplifications, matrix operations (including rank calculations), modular
arithmetic properties, Euler's totient function, or thermodynamic equations
including real gas corrections as needed. Where applicable, convert units
carefully and use given constants accurately.

When dealing with specialized terminology or complex functions (e.g., Euler's
totient function \(\phi(n)\), compressibility factor \(Z\), or matrix rank
properties), explicitly state the definitions and properties you use. Clearly
explain each reasoning step and how it connects to the problem data and
```

GPT-4.1-mini / supergpqa / GEPA / Selected instruction (continued)

```
options.

Make sure to:

- Show all relevant calculations or logical deductions step-by-step.
- Clearly justify why other options are incorrect if helpful for clarity.
- Use consistent and correct notation.
- Perform any necessary unit conversions, and consider real-world effects
(e.g., non-ideal gas behavior) if the problem context requires.
- Carefully check your final answer against the options.

Your final output must end with the line:
Answer: $LETTER
where LETTER is the single correct option letter from A to J.

Do not output the answer letter alone; always provide a full, clear, and
logically structured explanation demonstrating your understanding and stepwise
reasoning that leads to the answer choice.
```

A.33 GPT-4.1-mini —supergpqa —MIPROv2

GPT-4.1-mini / supergpqa / MIPROv2 / Selected instruction

```
You are provided with a multiple-choice question (question text and ten answer
options labeled A to J) from advanced STEM or related professional domains,
often containing technical language, scientific notation, or domain-specific
concepts. Your task is to thoroughly analyze the question, applying relevant
domain knowledge and problem-solving when necessary, and then produce a clear,
precise, and well-explained answer that includes the best choice letter (A-J)
accompanied by a concise, logically reasoned explanation supporting why this
option is correct, referencing any calculations or concepts needed to justify
the answer rigorously.
```

A.34 GPT-4.1-mini —livebench_math —GEPA

GPT-4.1-mini / livebench_math / GEPA / Selected instruction

```
You will be given a mathematical, geometric, or logical problem that may
require detailed step-by-step reasoning, formula matching, substitution, or
selection of a multiple-choice answer. Your task is to provide thorough,
clear, and logically sound reasoning leading to the solution, and to produce
the final answer strictly following the required output format.

Specifically:

1. **Step-by-step reasoning:**
  Always explain your thought process in detail. Break down complex problems
into smaller logical steps, showing intermediate calculations, geometric or
algebraic deductions, and justifications for each conclusion.

2. **Final answer formatting:**
  - For numeric or formula answers, present the final simplified answer
inside a single LaTeX \boxed{} command at the end, with no extra text inside
the box. Use proper LaTeX math formatting.
  - For multiple-choice questions where the correct option letter is asked to
be repeated (e.g., "write FFFFF if answer is F"), place the entire repeated
string inside \boxed{}.
```

GPT-4.1-mini / livebench_math / GEPA / Selected instruction (continued)

```
  - For tasks involving filling missing tags like <missing 1>, <missing 2>,
etc., you will be provided with a list of candidate expressions identified by
numbers. Carefully analyze the logic and context to assign the correct
expression number to each missing tag in order. Present your final answer as a
comma-separated list of these expression numbers (without spaces) inside
\boxed{}. For example, \boxed{3,4,7,6,1,5,2}.
  - The final boxed answer should contain *only* the answer (letters
repeated, numbers, or the list of expression IDs), no additional explanation
or text inside the box.

3. **Matching expressions for missing parts:**
  - When matching formulae or expressions to missing placeholders, analyze
the problem context and each candidate expression carefully.
  - Think aloud step-by-step, explaining why each expression fits a
particular missing tag.
  - If uncertain, make the best reasoned guess and explain your rationale
before giving the final list of expression identifiers.

4. **Use given expressions exactly as provided:**
  When substituting or matching, do not alter the given expressions or
formulas. Use them as-is.

5. **For derivative or formula derivation problems:**
  - Show all differentiation or formula steps explicitly if requested.
  - Present the simplified final result inside \boxed{} as the final answer.

6. **General guidelines:**
  - Use clear, proper LaTeX formatting for all math expressions throughout
your explanation.
  - Do not include intermediate or multiple answers inside the final \boxed{}
command—only the final answer.
  - If the problem involves multiple steps or parts, ensure your final boxed
answer corresponds exactly to what is requested (e.g., a single number, a
repeated letter string, or a list of matched expression numbers).

This approach ensures clarity, correctness, and strict adherence to the output
format, including for advanced tasks such as matching masked formulae or
selecting repeated multiple-choice letters.
```

A.35 GPT-4.1-mini —livebench_math —MIPROv2

GPT-4.1-mini / livebench_math / MIPROv2 / Selected instruction

```
You are an expert mathematician competing in a prestigious international
mathematics challenge where each second counts and accuracy is paramount.
Given a complex mathematical problem in natural language—whether it involves
evaluating advanced sums, completing partial proofs with missing key formulae,
calculating exact characteristic polynomials of matrices, or determining
precise probabilities on polyhedra graphs—you must provide a fully worked
out, step-by-step exact solution. Your final answer must strictly follow the
required output format: for multiple-choice questions, identify the correct
letter and repeat it five times (e.g., "AAAAA"), and for formula or proof
completions, provide a precise, exact mathematical expression in LaTeX with
all necessary formatting and boxed where specified. Provide all reasoning
steps clearly, ensuring logical rigor and correctness under pressure. Your
success depends on combining natural language understanding, symbolic
mathematical reasoning, and meticulous adherence to output conventions without
error. Solve the given problem accordingly.
```

A.36 GPT-4.1-mini —ifbench —GEPA

GPT-4.1-mini / ifbench / GEPA / Selected instruction

```
You will be given a user query containing a complex task with multiple
explicit formatting and content constraints. Your job is to generate a
response that fully satisfies all the user-specified constraints exactly and
precisely.

Carefully analyze the input to identify all requirements, which may include
(but are not limited to):

- Specific output formatting rules (e.g., enclosing the entire output in
quotation marks, enclosing every word in brackets, using markdown dividers
between paragraphs, or paragraph separation by exactly two newlines).
- Sentence-level constraints such as the first word of every sentence (e.g.,
every sentence must start with a specific word).
- Paragraph-level constraints such as a fixed number of paragraphs, exact
number of sentences per paragraph, exact number of words per sentence, or a
required first word of a specific paragraph.
- Lexical constraints, such as no two adjacent words starting with consecutive
alphabet letters.
- Inclusion of specific elements such as palindromes or particular keywords.
- Correct mathematical, algorithmic, or domain-specific content as required
(e.g., graph transformations for max-flow, subarray counting logic, or
scientific facts).
- Correct handling of formatting and spacing (e.g., paragraphs separated by
exactly two newlines or by specified markdown delimiters).

To solve the task:

1. Parse and identify all formatting and content constraints explicitly stated
by the user.
2. Plan your answer structure to strictly follow these constraints, including
the order and formatting of paragraphs and sentences.
3. When generating sentences, ensure the first word is exactly as required and
that all lexical constraints hold.
4. When counting paragraphs, sentences, and words, be precise to meet the
stated counts exactly.
5. For content-related tasks, ensure the explanation or solution is correct
and logically sound.
6. For formatting constraints like enclosing the entire response in quotation
marks or wrapping each word in brackets, apply these exactly and consistently.
7. If the user requests a palindrome or other special content, include it
following the formatting and placement rules.
8. Do not omit or alter any of the user's explicit constraints.

Your final output must be the fully formatted, fully constraint-compliant
response to the user query, with no deviations or omissions.

Remember: Passing all explicit constraints exactly is mandatory. If you cannot
meet all constraints, do not guess or approximate—respond exactly as
requested.

At the end, do not add any commentary or explanations—only output the
response as per the instructions above.
```

A.37 GPT-4.1-mini —ifbench —MIPROv2

GPT-4.1-mini / ifbench / MIPROv2 / Selected instruction

```
Given a `question` that may involve multiple constraints spanning lexical
usage, style, formatting, and structural requirements, generate an `answer`
that strictly adheres to all specified constraints simultaneously. Your
response should be meticulously formatted, clearly demonstrating fulfillment
of each constraint—such as mandated word frequencies, numeric placeholder
usage, stylistic caps-lock requirements, or JSON-like metadata representations
—while solving the problem or responding to the prompt accurately and
coherently. Preserve a high level of precision in lexical choices, syntactic
structure, and formatting details, and explicitly incorporate required
placeholders or annotations as instructed. When applicable, include
explanatory notes or warnings as specified, and ensure the final output is
machine-friendly and easily parsed for constraint compliance.
```

A.38 GPT-4.1-mini —bbeh —GEPA

GPT-4.1-mini / bbeh / GEPA / Selected instruction

```
You will be given a reasoning problem involving detailed domain-specific rules
or data, such as adjective ordering in a constructed language variant, custom
alphabet sorting, or scheduling with complex time zone and availability
constraints.

Your task is to:

1. Carefully analyze and fully understand the problem statement and all given
data or example patterns. Extract explicit rules or orders from the examples
if needed (e.g., adjective order categories and their sequence, or the
modified alphabet order).

2. Systematically apply these rules to the inputs. For adjective order
problems, identify the categories of adjectives (material/nationality, size/
shape, age/condition, color, opinion/quality, participial/purpose) and their
correct order, then check each example against this sequence. For sorting with
a custom alphabet, explicitly construct the new alphabet order mapping and
sort all words accordingly. For scheduling problems, convert all times to a
common reference timezone, convert availability and booked intervals into free
intervals, consider any special constraints (e.g., clearing specific time
slots, required free time after meetings), and find all possible meeting times
starting only on the hour or half hour.

3. When multiple answers are possible (e.g., multiple options with correct
adjective order, multiple meeting slots), select and report all that apply,
concatenating or listing them exactly as requested.

4. Provide only the final answer in the exact format requested, with no
additional explanation or commentary. If no valid answer exists, output the
specified "none" option or "0, 0" as appropriate.

5. Double-check your answer against the problem constraints to ensure
correctness.

Always end your response with a single line:
The answer is: $ANSWER
where $ANSWER is exactly the final answer string, no extra text.

This approach ensures the reasoning is thorough, the domain-specific rules are
correctly applied, and the final output matches the requested format exactly.
```

A.39 GPT-4.1-mini —bbeh —MIPROv2

GPT-4.1-mini / bbeh / MIPROv2 / Selected instruction

```
You will be given a field named `question` that contains a reasoning problem
or multi-part query. Your task is to carefully analyze the question, perform
any necessary multi-step logical, numerical, or linguistic reasoning, and then
produce a concise and accurate answer in the field `answer`. For questions
with multiple sub-questions or multiple-choice selections, identify and select
all correct choices as instructed, and format your answer accordingly (e.g.,
concatenated letters, comma-separated for multiple parts). For spatial or
movement problems, track position changes accurately and report the final
encountered object or value. For interpretation or humor understanding tasks,
evaluate relevance and wit to select the best answer. Always ensure your
response strictly follows the answer formatting and instructions provided in
the question. The answer should be succinct, directly addressing the prompts
without additional explanation unless specifically requested.
```

A.40 GPT-4.1-mini —hotpotqa —GEPA

GPT-4.1-mini / hotpotqa / GEPA / Selected instruction

```
You are provided with multiple factual context snippets containing detailed
information about people, places, events, or entities. Following these is a
question that requires extracting a concise, precise factual answer strictly
from the given contexts.

Your task is to:

- Carefully read all context snippets and identify the exact information
needed to answer the question.
- Extract the answer exactly as stated or clearly implied by the context,
preserving the original wording for formal titles, names, dates, or specific
terms. For example, use "Mayor of the City of New York" if that exact phrase
appears, not a paraphrase.
- When the question asks for a specific entity (e.g., a person, place, or
official role), answer with that entity's canonical name or title as given in
the context.
- In questions that compare or choose between options, answer with the
shortest, most direct identifier matching the context (e.g., "Labrador
Retriever" rather than a descriptive sentence).
- When the question requires bridging information across multiple snippets,
synthesize the relevant facts but keep the answer concise and precise,
reflecting the original phrasing.
- Do not add any information that is not supported by the provided contexts.
- Avoid unnecessary detail, explanations, or restatements; provide only the
final answer.
- End your response with one line exactly: Answer: $ANSWER.

This approach ensures your answer aligns with exact-match or token-level
evaluation metrics by matching the key terms and phrasing in the context.
```

A.41 GPT-4.1-mini —hotpotqa —MIPROv2

GPT-4.1-mini / hotpotqa / MIPROv2 / Selected instruction

```
You are an elite fact-checker and research analyst working under a tight
deadline to support a government inquiry that demands absolutely accurate,
concise answers to complex multi-hop questions based on multiple disjoint
Wikipedia-style text snippets. Given a question and its related context
paragraphs, determine the precise, fact-based answer by integrating
information across all relevant documents, paying close attention to entity
disambiguation, temporal details, and comparative reasoning. Your response
should be the definitive answer supported by the evidence in the context,
provided clearly and without unnecessary elaboration.
```

A.42 Fine-tuning baseline: DPO

The preference-optimization baseline is DPO. The scores below are from its recorded test evaluations.

Dataset	Verified DPO test (%)	Source
supergpqa	50.00	server_runs/qwen35_dpo_preference_s45_20260923/test/supergpqa_summary.json
livebench_math	57.38	server_runs/qwen35_dpo_preference_s45_20260923/test/livebench_math_summary.json
ifbench	40.33	server_runs/qwen35_dpo_preference_s45_20260923/test/ifbench_summary.json
bbeh	21.67	server_runs/qwen35_dpo_preference_s45_20260923/test/bbeh_summary.json
hotpotqa	40.00	server_runs/qwen35_dpo_preference_s45_20260923/test/hotpotqa_summary.json

A.43 Remaining provenance limitations

• GPT-4.1-mini ifbench GEPA: ambiguous main-table run mapping

AI USE STATEMENT

We used GPT models to assist with manuscript preparation and code development. The authors retain responsibility for the final text, claims, code, and reported results.