MINDLOOM: Composing Thought Modes for Frontier-Level Reasoning Data Synthesis

Haiyang Shen[1,*], **Taian Guo**[2,*], **Xuanzhong Chen**[3,*],
Mugeng Liu[1], **Sixiong Xie**[1], **Zhuofan Shi**[1], **Chongyang Pan**[1], **Siqi Zhong**[1], **Guoqing Wang**[1]
Ming Zhang[2,†], **Yun Ma**[1,†]

[1]Institute for Artificial Intelligence, Peking University
[2]State Key Laboratory of Multimedia Information Processing, School of Computer Science,
PKU-Anker LLM Lab, Beijing Key Laboratory of Software and Hardware
Cooperative Artificial Intelligence Systems, Peking University
[3]Tsinghua University
hyshen@stu.pku.edu.cn, taianguo@stu.pku.edu.cn, cxz23@mails.tsinghua.edu.cn,
mzhang_cs@pku.edu.cn, mayun@pku.edu.cn

Code: https://github.com/EachSheep/MindLoom
*Equal contribution. †Corresponding authors.

Abstract

Although LLMs have made substantial progress in reasoning, systematically producing frontier-level reasoning data remains difficult. Existing synthesis methods often have limited visibility into the structural factors that govern problem difficulty, which can result in narrow diversity and unstable difficulty control. In this work, we view the difficulty of a reasoning problem as arising from the accumulation of atomic knowledge-reasoning transformations, which we term *thought modes*. Building on this perspective, we propose MINDLOOM, a framework for synthesizing frontier-level reasoning data through compositional thought mode engineering. Given a collection of hard problems with verified solutions, MINDLOOM first decomposes those solutions into thought mode chains that reveal each problem's construction logic. It then trains a retrieval model that matches problem states to compatible thought modes, providing guidance on which reasoning challenges to introduce during synthesis. New problems are composed by iteratively applying retrieved thought modes to seed questions, with distribution-aligned sampling to encourage diverse reasoning coverage. Finally, a rollout-based judging stage labels generated questions by difficulty and supplies judged-correct responses for supervised fine-tuning. We evaluate MINDLOOM on nine benchmarks covering five STEM disciplines and four mathematical reasoning tasks across multiple model families and sizes. Models fine-tuned on MINDLOOM-generated data achieve favorable performance relative to base models, distillation, and external-data baselines across the reported benchmarks. Ablation studies indicate the contribution of each component, and further analysis suggests that MINDLOOM covers a broad range of reasoning patterns while maintaining useful difficulty control. We have open-sourced our implementation at https://github.com/EachSheep/MindLoom.

1 Introduction

LLMs have made substantial progress on complex reasoning tasks across mathematics, science, and other domains [26, 13, 38]. As models advance, evaluation increasingly relies on benchmarks that remain challenging at the frontier of model capability [9, 27]. Yet manually constructing frontier-level reasoning problems is expensive, requires deep domain expertise, and is hard to scale.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Existing reasoning data construction falls into three categories. The first category focuses on *reasoning data synthesis*, where strong models generate questions or curate solution traces [13, 22, 41]. The second category targets *evaluation benchmark construction*, in which human experts craft questions that push the boundary of model capability [9, 27, 2]. The third category employs *data selection strategies* to curate high-value training data from large pools [42, 34, 11].

Despite their contributions, these approaches face notable limitations. Synthesis methods rarely model what makes a problem structurally hard, so generated questions tend to be superficially varied but homogeneous in their reasoning composition. Benchmark construction by experts produces high-quality questions but is slow, costly, and hard to scale to training-data volumes. Data selection identifies valuable samples but cannot create new problems, and its coverage is bounded by source-pool diversity. Across the three categories, controllable frameworks for the compositional structure of reasoning difficulty remain underdeveloped.

To address these limitations, we first ask: what makes a reasoning problem hard? Rather than treating difficulty as a monolithic property, we model it as a composite of atomic reasoning requirements. Each such requirement introduces specific knowledge and demands a corresponding derivation step. We call these atomic requirements *thought modes*. This abstraction turns difficulty control into a compositional operation: new problems can be constructed by selecting and combining reusable thought modes in novel configurations. Based on this perspective, we propose MINDLOOM, which synthesizes high-quality reasoning training data through compositional thought mode engineering.

MINDLOOM addresses four challenges through a four-stage pipeline. First, to learn how existing hard problems are constructed, we analyze verified solutions in reverse, which we term *reverse engineering*, producing a chain of thought modes that serves as each problem's construction blueprint (Section 3.2). Second, to decide which thought mode applies to a given problem state, we train an embedding-based retrieval model on the extracted chains (Section 3.3). Third, to compose new problems with broad reasoning coverage, we iteratively apply retrieved thought modes under a distribution-aligned sampling scheme that discourages concentration on common reasoning types (Section 3.4). Fourth, to assemble training data, we run multi-rollout inference with LLM-based judging, filter by source provenance, and convert judged-correct rollouts into supervised fine-tuning records (Section 3.5).

We evaluate MINDLOOM on nine benchmarks spanning five STEM disciplines and four mathematical reasoning tasks, including competition-level problems. Across the Qwen3 and Qwen3.5 families, models fine-tuned on MINDLOOM-generated data are compared against direct base-model inference, knowledge distillation (DS-V3.2 Distill), and external curated reasoning corpora (MegaScience, OpenThought). Results show consistent improvements across benchmarks, with especially large gains on competition-style mathematical reasoning. Ablation studies, distribution analysis, and case studies further verify the contribution of each pipeline stage.

Our main contributions are as follows:

- We propose the concept of *thought modes*, defined as atomic knowledge-reasoning transformations whose composition gives rise to problem difficulty, and develop a reverse-engineering procedure that decomposes expert-level problems into thought mode chains, offering a structural perspective on reasoning difficulty.
- We present MINDLOOM, a framework that integrates thought mode retrieval, distribution-aligned compositional synthesis, rollout-based judging, and source-aware conversion into a single pipeline for reasoning data construction.
- We validate MINDLOOM on nine benchmarks across four model sizes from two model families. Models trained on MINDLOOM-generated data achieve favorable performances over various types of baselines, supporting the effectiveness of compositional thought mode engineering.

2 Related Work

This section reviews related work from three perspectives: reasoning data synthesis (Section 2.1), evaluation benchmark construction (Section 2.2), and data selection for reasoning (Section 2.3).

2.1 Reasoning Data Synthesis

Reasoning data synthesis methods can be grouped into three lines. *Knowledge distillation and open data-recipe* methods leverage strong reasoning models to generate or curate solution traces for existing problems, transferring frontier-model reasoning capability to smaller ones at scale [13, 25, 12]. *Direct generation and augmentation* methods prompt LLMs to create new questions through templates, evolutionary prompting, or surface-level transformations [22, 40, 36]; the diversity and difficulty of the output depend largely on prompt design and seldom expose the structural factors that make problems hard. *Structured and skill-compositional* methods build symbolic generators or learn policies that assemble difficult problems from extracted skills [20, 28, 41]; they rely on logic templates or reinforcement learning and typically operate within a single domain such as mathematics.

MINDLOOM differs in that it explicitly models how hard problems are constructed. Rather than treating difficulty escalation as opaque prompting or domain-specific policy learning, MINDLOOM reverse-engineers verified solutions across broad STEM domains into atomic thought mode transformations and composes new problems through learned retrieval and distribution-aligned sampling, without RL training or symbolic templates.

2.2 Evaluation Benchmark Construction

Evaluation benchmarks for reasoning have evolved from standardized tests to frontier-level challenges. Early benchmarks such as GSM8K and MATH focus on grade-school and competition-level mathematics with well-defined difficulty tiers [5, 15], but are now largely saturated by frontier models. Recent efforts push toward the boundary of human and model capability, including HLE, GPQA, and FrontierMath [9, 27, 10], with complementary work on broad expert domains and uncontaminated olympiad-level evaluation [7, 2]. Such benchmarks are typically curated by domain experts, which produces high-quality questions but is slow, costly, and hard to scale. A separate line of work explores dynamic or synthetic benchmark construction by extracting and perturbing reasoning structures [43, 37], highlighting the value of controllable reasoning structures for evaluation.

MINDLOOM draws on the construction principles that make expert-crafted questions hard, that compositional reasoning requirements accumulate across multiple steps, but applies them to training-data synthesis. By reverse-engineering the construction logic of solved hard problems and composing thought modes in novel configurations, MINDLOOM produces new training problems whose construction logic mirrors patterns observed in expert-crafted ones, with greater diversity and scalability.

2.3 Data Selection for Reasoning

Recent work on data selection for reasoning training aims to identify the most valuable samples from large data pools. *Quality-based* methods use signals such as gradient influence, model uncertainty, learned quality ratings, and training-dynamics objectives to prioritize samples that improve downstream capabilities [34, 21, 33]. *Pattern-based* methods abstract structural patterns from reasoning traces and select data that covers diverse, high-value patterns; for example, CoTP extracts atomic reasoning patterns from chain-of-thought sequences [42, 39]. The shared idea is that a carefully chosen subset can match or exceed training on the full pool, improving training efficiency.

MINDLOOM complements data selection by creating new reasoning data through compositional synthesis rather than selecting from an existing pool. CoTP [42] extracts ordered atomic patterns from reasoning traces and uses pattern-chain and token-entropy similarity to select existing chain-of-thought data. Select2Reason [39] ranks existing long-chain instructions using question difficulty and reasoning-trace length. In contrast, a MINDLOOM mode records a problem-construction transformation from one intermediate question state to the next, separating transferable knowledge K_{gen} from instance-specific K_{spec}. Retrieved modes guide the synthesis of new questions by recombining transformations from the source corpus.

3 Method

3.1 Overview

Figure 1 illustrates the four-stage MINDLOOM pipeline. Step 1 (*Thought Mode Extraction*, Section 3.2) addresses *what makes problems hard*: it reverse-engineers verified solutions into thought mode chains, populating a thought mode bank $\mathcal{B}$. Step 2 (*Retrieval Learning*, Section 3.3) addresses *which thought mode to apply next*: it trains an embedding-based retriever on these extracted chains. Step 3 (*Distribution-Aligned Synthesis*, Section 3.4) addresses *how to generate diverse problems*: it iteratively retrieves and selects compatible thought modes for seed questions via a similarity-scarcity trade-off, applying them via an LLM to yield increasingly complex questions. Step 4 (*Rollout-Based Filtering*, Section 3.5) addresses *how to prepare training data*: it performs multi-rollout judging and provenance filtering, converting correct rollouts into SFT records.

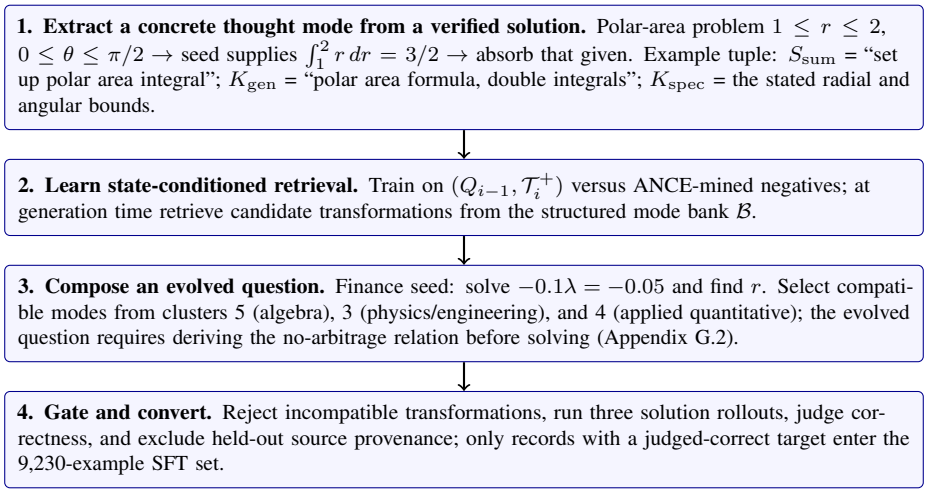


Figure 1: The four-stage MINDLOOM pipeline with an explicit extracted tuple and a separate seed-to-evolved-question synthesis trace. The examples illustrate different runs; Appendix G gives their full details.

3.2 Thought Mode Extraction

To synthesize hard problems, we first need to understand what makes existing problems hard. Rather than treating difficulty as an opaque property, MINDLOOM decomposes it into atomic components by analyzing how hard problems could have been constructed. The decomposition yields an inverse curriculum of progressively simpler sub-problems that guides subsequent synthesis. We first define the core abstractions and then describe the extraction process. Detailed prompt templates and extended examples are provided in Appendix B and C.

Thought Mode. A thought mode $\mathcal{T} = (S_{\text{sum}}, S_{\text{det}}, K_{\text{gen}}, K_{\text{spec}})$ is an atomic knowledge-reasoning transformation that adds one layer of complexity to a problem. The four components capture complementary aspects: S_{sum} provides a high-level summary of the transformation type (e.g., "introduce integral computation"); S_{det} details the specific modification; K_{gen} captures the general, transferable knowledge required (e.g., theorems, formulas, or principles); and K_{spec} records the problem-specific parameters involved (e.g., constraints, numerical values, or domain setup). Each thought mode represents a single, well-defined reasoning requirement that a solver must handle.

Thought Mode Chain. A thought mode chain $\mathcal{C} = [\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n]$ is an ordered sequence of thought modes that encodes the construction blueprint of a problem. When applied sequentially to a seed question Q_0, the chain produces increasingly complex questions $[Q_1, Q_2, \ldots, Q_n]$, where the final question Q_n approximates the original hard problem. The chain captures both the structural progression of difficulty and the specific knowledge-reasoning requirements at each layer.

Reverse Engineering Process. We analyze the verified solution reversely, since reasoning dependencies are easier to isolate from the tail of a solution than from its head. Given a hard problem Q with a

verified solution $S = [s_1, s_2, \ldots, s_m]$, we partition S into k windows $[W_1, W_2, \ldots, W_k]$ from the tail, each containing a fixed number of consecutive steps. The extraction then proceeds in two phases:

Phase 1: Seed Generation. Starting from the last window W_k, we generate a minimal seed question Q_0 whose solution requires only the steps in W_k. The principle of *dependency isolation* ensures that any values computed in earlier steps are converted into explicit givens in Q_0. This makes the seed question fully self-contained and independently solvable.

Phase 2: Iterative Evolution. Starting from Q_0, we iterate backwards through the remaining windows. At each step i, we perform *dependency absorption*: the current question Q_{i-1} is modified to remove an explicit given that corresponds to what window W_{k-i} computes. This forces the solver to derive that value, effectively adding one thought mode of complexity. The process produces the evolved question Q_i along with the thought mode $\mathcal{T}_i$ that describes the added reasoning requirement.

Formally, the process is:

$$Q_0 = \text{Seed}(W_k, [s_1, \ldots, s_{m-|W_k|}]) \tag{1}$$

$$Q_i, \mathcal{T}_i = \text{Absorb}(Q_{i-1}, W_{k-i}, [s_1, \ldots, s_{m-\sum_{j=0}^{i}|W_{k-j}|}]), \quad i = 1, \ldots, k-1 \tag{2}$$

Both operations are implemented via LLM prompting with structured output (see Appendix C for details). Complete extraction of a single problem yields a thought mode chain $\mathcal{C} = [\mathcal{T}_1, \ldots, \mathcal{T}_{k-1}]$ and the corresponding question evolution sequence $[Q_0, Q_1, \ldots, Q_{k-1}]$. Applying this process to a corpus of hard problems produces a *thought mode bank* $\mathcal{B}$ containing diverse atomic transformations.

3.3 Thought Mode Retrieval Learning

With a bank of extracted thought modes, the next challenge is determining which thought modes are applicable to a given problem state during synthesis. Not every thought mode is compatible with every problem: applying an algebraic transformation to a geometry problem, for example, may produce an incoherent result. To provide directional guidance for synthesis, we train an embedding-based retrieval model that matches problem states to compatible thought modes. The full training configuration is provided in Appendix D.

Training Data Construction. From the extracted thought mode chains, we construct training pairs $(Q_{i-1}, \mathcal{T}_i)$, where Q_{i-1} is the problem state before applying thought mode $\mathcal{T}_i$. The query side is the question text of Q_{i-1}, and the document side is a serialized concatenation of $S_{\text{sum}}, S_{\text{det}}, K_{\text{gen}}$, and K_{spec}. A richer state representation is additionally embedded during synthesis. The exact serialization formats are provided in Appendix D.

Hard Negative Mining. We mine hard negatives by retrieving, under the current model, the nearest non-positive thought modes for each query, and refresh them every R optimizer steps so that they remain challenging as the model improves. Random negatives are used only when the retrieved pool is too small. Index construction details are provided in Appendix D.

Training Objective. The retrieval model is trained with a margin ranking loss:

$$\mathcal{L} = \sum_{(Q_{i-1}, \mathcal{T}_i^+)} \sum_{\mathcal{T}_j^-} \max\big(0,\ \text{sim}(Q_{i-1}, \mathcal{T}_j^-) - \text{sim}(Q_{i-1}, \mathcal{T}_i^+) + \gamma\big) \tag{3}$$

where $\mathcal{T}_i^+$ is the correct thought mode, $\mathcal{T}_j^-$ are hard negatives, $\text{sim}(\cdot, \cdot)$ is the embedding similarity, and γ is the margin. This objective encourages the model to place compatible thought modes closer to the query problem state in the embedding space than incompatible ones.

3.4 Distribution-Aligned Compositional Synthesis

With the retrieval model providing directional guidance, we can now compose new problems by iteratively applying thought modes to seed questions. However, naive composition tends to concentrate on a small set of common thought mode types, which can lead to structurally homogeneous outputs. To mitigate this mode collapse and encourage broad coverage of the reasoning types present in the reference corpus, we design a three-sub-step synthesis loop with distribution-aligned sampling. Appendix E gives the full algorithm.

Sub-step 1: Match Problem State to Compatible Thought Modes. Starting from a seed question Q_0, MINDLOOM performs up to n evolution steps. At each step i, the fine-tuned retrieval model (Section 3.3) encodes the current problem state Q_i and retrieves the top-m most similar thought modes from the bank $\mathcal{B}$ as candidates for the next evolution.

Sub-step 2: Distribution-Aligned Scoring. The retrieved candidates are then re-scored using a separate general-purpose embedding model, which provides the similarity term in a combined score that also accounts for cluster-level coverage. Specifically, we cluster the thought mode bank into K clusters using K-Means [23] on thought mode embeddings. Let P_{ref} denote the reference distribution of thought modes across clusters in the bank, and P_{gen} denote the empirical distribution of thought modes used in generation so far. For each candidate thought mode $\mathcal{T}_j$ belonging to cluster c_j, its selection score is:

$$\text{score}(\mathcal{T}_j \mid Q_i) = \alpha \cdot \text{sim}(Q_i, \mathcal{T}_j) + (1-\alpha) \cdot \tanh\left(\log\left(1 + \frac{P_{\text{ref}}(c_j) + \epsilon}{P_{\text{gen}}(c_j) + \epsilon}\right)\right) \tag{4}$$

where $\alpha \in [0,1]$ balances compatibility and coverage, and ϵ is a smoothing constant. The first term ensures the selected thought mode is semantically compatible with the current problem state. The second term is the bounded *scarcity reward*: thought mode types that are underrepresented in the generated data relative to the reference distribution receive higher scores, while the $\tanh$ bound keeps the reward on a comparable scale to similarity. We align to P_{ref} rather than a uniform distribution to respect the natural frequency of reasoning types across domains.

The final selection is made through softmax sampling with temperature τ:

$$P(\mathcal{T}_j \mid Q_i) = \frac{\exp\big(\text{score}(\mathcal{T}_j \mid Q_i)/\tau\big)}{\sum_l \exp\big(\text{score}(\mathcal{T}_l \mid Q_i)/\tau\big)} \tag{5}$$

This stochastic selection introduces controlled randomness while still favoring high-scoring candidates. The distribution tracker P_{gen} is updated after each synthesis step, enabling consistent diversity control throughout large-scale generation.

Sub-step 3: Apply Thought Mode. The selected thought mode $\mathcal{T}_{i+1}$ is applied to the current problem state Q_i through an LLM-based transformation to produce the evolved question Q_{i+1}. The LLM also returns an `is_compatible` flag after checking solvability, answer uniqueness, condition consistency and necessity, and whether the transformation closes the intended reasoning loop. Algorithm 1 in Appendix E specifies how incompatible steps are handled. The three sub-steps repeat for up to n iterations, yielding a sequence of increasingly complex questions $Q_1, Q_2, \ldots, Q_n$.

3.5 Rollout-Based Filtering and Conversion

To measure difficulty and select training targets, we run three sampled solutions for each generated question and use an LLM judge to label each rollout as correct or incorrect. The rollouts produce three difficulty categories (Appendix E.7): all-correct, partial, and all-wrong. For supervised fine-tuning (SFT) conversion, the criterion is the availability of a judged-correct target response from the non-thinking rollouts or paired thinking-mode rollouts. Questions with no judged-correct response across these rollouts are excluded.

We additionally filter generated data by source provenance before SFT conversion. The pipeline maps each generated seed back to its original question and datasource, and removes generated examples whose origins match the held-out benchmark sources listed in Appendix F. This separates source-aware benchmark filtering from difficulty labeling and keeps the training conversion aligned with the evaluation protocol.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate on nine benchmarks spanning five STEM disciplines and four mathematical reasoning tasks. Five benchmarks cover STEM reasoning: CS-Bench [29], ChemBench [24], HLE [9], MedQA [17], and SciBench [31]. Four benchmarks target mathematical reasoning: MATH-500 [15], HMMT February 2025 [2], HMMT November 2025 [2], and AIME 2025 [2]. We report

Table 1: Main results. Each cell reports pass@1 / pass@3 (%), with the best value in each model-size block in bold. HMMT-Feb. and HMMT-Nov. denote the February and November 2025 splits.

Model	Setting	General and STEM					Mathematical Reasoning			
		CS-Bench	ChemBench	HLE	MedQA	SciBench	MATH-500	HMMT-Feb.	HMMT-Nov.	AIME 2025
Qwen3-2507 family (%)										
4B	Base	64.82 / 74.25	56.56 / 65.81	4.72 / 8.24	62.69 / 74.70	50.43 / 61.85	84.40 / 90.20	20.00 / 26.67	26.67 / 30.00	30.00 / 33.33
4B	DS-V3.2 Distill	63.62 / 78.87	52.16 / 69.79	5.16 / 11.40	60.17 / 76.28	47.90 / 64.90	81.20 / 91.60	23.33 / 33.33	23.33 / 30.00	26.66 / 36.70
4B	MegaScience	64.53 / 77.06	57.32 / 67.10	5.28 / 11.16	60.48 / 77.92	50.14 / 67.92	83.80 / 91.40	33.33 / 40.00	26.67 / 33.33	30.00 / 50.00
4B	OpenThought	65.85 / 78.75	60.70 / 72.26	5.72 / 11.32	59.78 / 74.94	49.42 / 69.51	**85.20** / 94.00	**40.00 / 46.67**	30.00 / **40.00**	**43.33 / 56.67**
4B	MINDLOOM	**68.29 / 80.16**	**61.72 / 74.78**	**5.96 / 12.12**	**65.12 / 81.30**	**53.61 / 71.53**	**85.20 / 98.60**	33.33 / 43.33	**33.33 / 40.00**	40.00 / 53.33
8B	Base	62.79 / 73.54	54.05 / 64.63	4.76 / 8.48	63.00 / 78.47	52.60 / 65.03	82.40 / 91.00	30.00 / 33.33	30.00 / 33.33	36.67 / 46.67
8B	DS-V3.2 Distill	65.97 / **81.52**	53.07 / 69.16	5.04 / 10.36	67.47 / 81.78	52.31 / 68.93	84.40 / 93.00	14.81 / 33.33	30.00 / 43.33	36.67 / 40.00
8B	MegaScience	66.62 / 79.37	54.72 / 70.41	5.28 / 10.76	67.32 / 82.95	54.05 / 69.65	85.20 / 93.20	36.67 / 40.00	**46.67 / 53.33**	43.33 / 46.67
8B	OpenThought	67.50 / 80.57	55.97 / 70.10	6.08 / 11.36	66.92 / 81.07	56.21 / 68.93	**87.80** / 97.20	**46.67** / 50.00	43.33 / 50.00	40.00 / 53.33
8B	MINDLOOM	**69.65** / 81.40	**59.04 / 72.97**	**6.64 / 11.64**	**70.77 / 86.80**	**57.08 / 73.41**	86.80 / **97.40**	43.33 / **56.67**	**46.67** / 50.00	**46.67 / 60.00**
Qwen3.5 family (%)										
4B	Base	66.27 / 79.61	58.65 / 70.78	5.96 / 11.08	74.62 / 82.32	53.17 / 67.74	87.80 / 94.60	33.33 / 43.33	36.67 / 53.33	36.70 / 50.00
4B	DS-V3.2 Distill	66.80 / 80.11	57.67 / 71.09	6.20 / 12.36	73.06 / 86.57	60.69 / 76.73	87.60 / 96.50	50.00 / 53.33	43.33 / 53.33	46.67 / 56.67
4B	MegaScience	66.97 / 80.57	57.36 / 71.87	6.32 / 13.04	74.70 / 85.23	60.00 / 78.90	87.00 / 95.40	53.33 / 60.00	43.33 / 63.33	50.00 / 53.33
4B	OpenThought	68.91 / 79.87	58.97 / **74.18**	6.84 / 13.28	75.02 / 87.90	57.66 / 76.30	**90.20 / 96.60**	50.00 / **67.78**	50.00 / **66.67**	56.67 / **66.67**
4B	MINDLOOM	**69.70 / 81.11**	**62.67** / 73.52	**7.56 / 14.04**	**78.24 / 88.81**	**66.18 / 79.62**	89.80 / 96.40	**56.67** / 60.00	**53.33** / 60.00	**60.00** / 63.33
9B	Base	69.60 / 80.19	63.20 / 77.17	6.24 / 12.04	78.71 / 90.97	51.73 / 69.08	89.20 / 96.80	40.00 / 46.67	40.00 / 60.00	40.00 / 60.00
9B	DS-V3.2 Distill	71.31 / 82.92	62.90 / 78.05	6.64 / 13.92	79.97 / 90.42	65.03 / 79.05	90.80 / 97.20	43.33 / 66.67	43.33 / 60.00	43.33 / 70.00
9B	MegaScience	71.52 / 82.10	63.53 / 78.48	6.36 / 13.52	81.30 / 93.64	65.75 / 77.75	91.20 / 96.20	50.00 / 66.67	60.00 / 66.67	53.33 / 63.33
9B	OpenThought	73.13 / 82.32	64.67 / 77.53	7.12 / 13.08	77.37 / 92.38	67.48 / 80.78	**92.00** / 97.60	60.00 / 70.00	**70.00 / 80.00**	60.00 / **73.33**
9B	MINDLOOM	**73.96 / 84.79**	**67.86 / 80.21**	**8.12 / 14.88**	**86.80 / 95.12**	**71.39 / 85.11**	91.80 / **98.20**	**63.33 / 73.33**	**70.00** / 76.67	**66.67 / 73.33**

pass@1 and pass@3, defined as the fraction of test items for which at least one correct answer is found among one or three independent rollouts, respectively. Throughout this paper, *frontier-level* describes problems that current models solve unreliably under pass@3. Detailed benchmark descriptions are in Appendix F.

Reference Corpus and Contamination Control. The thought-mode bank is built from 58,526 source problems aggregated from 16 datasets spanning broad STEM and expert-reasoning domains. Before SFT conversion, we trace each synthesized item back to its seed provenance and remove generated examples whose origin belongs to any held-out benchmark source. The full list of source datasets, their sizes, the provenance-filter configuration, and the domain composition of the final SFT set are provided in Appendix F and E.6.

Baselines. We compare against the following baselines. *Base* directly evaluates the base model on the test sets without additional training. *DS-V3.2 Distill* fine-tunes the base model on reasoning trajectories that DeepSeek V3.2 [19] generates for the original source problems without any structural decomposition, serving as a strong trajectory-level distillation baseline. *MegaScience* [8] is an external large-scale science reasoning dataset baseline. *OpenThought* [12] is a reasoning dataset to train SOTA small reasoning models. MINDLOOM denotes our synthesized-data training recipe. All fine-tuned settings use the same foundation model and SFT configuration.

Implementation Details. We evaluate on Qwen3 [38] 4B and 8B, and Qwen3.5 [30] 4B and 9B. DeepSeek V3.2 serves as the chat model for thought mode extraction and synthesis. The retrieval model is initialized from Qwen3-Embedding-0.6B. Fine-tuning uses ms-swift [44] with full-parameter SFT for 3 epochs. Evaluation uses vLLM [18] with three independent rollouts per test item and LLM-based judging. To isolate the effect of data quality from data quantity, all fine-tuned settings use exactly 9,230 SFT examples with identical SFT hyperparameters and evaluation protocol; the DS-V3.2 Distill set is randomly subsampled from a larger distillation pool to this same budget. Full hyperparameters are provided in Appendix D and E.

4.2 Main Results

As shown in Table 1, MINDLOOM improves pass@3 over base-model inference across all nine benchmarks and four model sizes. For Qwen3-4B, MINDLOOM improves pass@3 on every benchmark over the base model, with especially large gains on mathematical reasoning: MATH-500 rises from 90.20 to 98.60, HMMT-Feb. from 26.67 to 43.33, HMMT-Nov. from 30.00 to 40.00, and AIME 2025 from 33.33 to 53.33. The same pattern appears for Qwen3-8B, where the largest improvements again occur on harder competition-style benchmarks and SciBench.

Table 2: Ablation study. Each cell reports pass@1 / pass@3 (%), with the best value in each block in bold. *w/o Scarcity* sets α=1.0 so selector sampling uses similarity only. *w/o Filter* skips rollout-based filtering while still applying source-provenance filtering. *Random Retriever* replaces learned retrieval with random candidates. *w/o Reverse Eng.* builds the thought-mode bank directly from rollouts.

Model	Variant	General and STEM					Mathematical Reasoning			
		CS-Bench	ChemBench	HLE	MedQA	SciBench	MATH-500	HMMT-Feb.	HMMT-Nov.	AIME 2025
Qwen3-2507 family (%)										
4B	w/o Scarcity	65.44 / 78.46	57.08 / 71.95	5.24 / 11.04	59.86 / 78.00	51.87 / 66.76	**92.40** / 96.80	**33.33 / 43.33**	26.67 / **43.33**	30.00 / 46.67
4B	w/o Filter	53.45 / 72.92	50.16 / 74.59	4.28 / 10.84	50.20 / 72.82	30.63 / 46.24	67.80 / 82.00	13.33 / 20.00	13.33 / 16.67	16.67 / 20.00
4B	Random Retriever	63.04 / 77.68	50.28 / 70.77	4.89 / 10.44	54.20 / 74.78	46.89 / 64.40	80.20 / 92.42	16.00 / 20.00	16.67 / 30.00	26.67 / 33.33
4B	w/o Reverse Eng.	61.02 / 77.64	50.47 / 71.83	4.96 / 11.16	57.34 / 75.49	45.81 / 63.44	81.00 / 91.20	23.33 / 36.67	23.33 / 30.00	33.33 / 40.00
4B	MINDLOOM	**68.29 / 80.16**	**61.72 / 74.78**	**5.96 / 12.12**	**65.12 / 81.30**	**53.61 / 71.53**	85.20 / **98.60**	**33.33 / 43.33**	**33.33** / 40.00	**40.00 / 53.33**
8B	w/o Scarcity	67.92 / 79.99	58.10 / 71.75	5.96 / 11.20	68.42 / 82.33	54.77 / 68.50	**87.20** / 95.00	40.00 / 53.33	40.00 / 46.67	36.67 / 50.00
8B	w/o Filter	53.62 / 72.92	49.88 / **73.72**	5.04 / **11.76**	53.50 / 76.36	29.19 / 49.85	58.40 / 72.40	20.00 / 26.67	20.00 / 33.33	13.33 / 23.33
8B	Random Retriever	63.74 / 79.08	54.29 / 70.61	5.68 / 10.92	64.57 / 82.64	49.86 / 68.79	83.40 / 93.00	33.33 / 36.67	26.67 / 40.00	26.67 / 33.33
8B	w/o Reverse Eng.	64.24 / 79.74	54.05 / 70.26	5.56 / 10.32	63.32 / 82.40	49.57 / 68.41	83.20 / 93.00	26.67 / 43.33	24.00 / 40.00	30.00 / 36.67
8B	MINDLOOM	**69.65 / 81.40**	**59.04** / 72.97	**6.64** / 11.64	**70.77 / 86.80**	**57.08 / 73.41**	86.80 / **97.40**	**43.33 / 56.67**	**46.67 / 50.00**	**46.67 / 60.00**
Qwen3.5 family (%)										
4B	w/o Scarcity	69.33 / 80.69	61.64 / 72.27	5.96 / 12.60	74.78 / **89.32**	63.44 / 75.29	89.20 / 95.80	36.67 / 50.00	46.67 / 50.00	43.33 / **63.33**
4B	w/o Filter	54.90 / 68.91	52.68 / 69.00	6.12 / 12.16	63.63 / 81.62	38.29 / 53.61	58.60 / 72.60	16.67 / 30.00	13.33 / 26.67	16.67 / 36.67
4B	Random Retriever	67.01 / 79.62	58.10 / 71.13	5.44 / 11.68	73.61 / 87.27	61.99 / 76.73	87.40 / 94.20	40.00 / 50.00	40.00 / 53.33	36.67 / **63.33**
4B	w/o Reverse Eng.	66.96 / 79.37	57.31 / 71.56	6.48 / 12.92	75.02 / 88.69	63.01 / 76.45	88.80 / 95.40	33.33 / 36.67	30.00 / 46.67	46.67 / 56.67
4B	MINDLOOM	**69.70 / 81.11**	**62.67 / 73.52**	**7.56 / 14.04**	**78.24** / 88.81	**66.18 / 79.62**	**89.80 / 96.40**	**56.67 / 60.00**	**53.33 / 60.00**	**60.00** / 63.33
9B	w/o Scarcity	72.26 / 83.67	62.66 / 75.73	7.44 / 14.40	81.85 / 93.72	67.49 / 80.78	89.80 / 97.60	60.00 / 70.00	53.33 / 70.00	60.00 / **76.67**
9B	w/o Filter	60.36 / 73.91	55.51 / 73.80	5.48 / 12.56	70.38 / 85.55	41.47 / 56.65	58.80 / 81.00	23.33 / 30.00	26.67 / 30.00	26.67 / 43.33
9B	Random Retriever	70.44 / 81.98	59.90 / 74.70	6.68 / 13.56	80.05 / 91.28	65.32 / 78.90	90.60 / 97.80	56.67 / 63.33	46.67 / 60.00	56.67 / 70.00
9B	w/o Reverse Eng.	70.97 / 82.80	62.20 / 73.92	7.52 / 14.56	80.36 / 92.14	65.32 / 78.61	90.20 / 97.80	46.67 / 60.00	50.00 / 66.67	43.33 / 66.67
9B	MINDLOOM	**73.96 / 84.79**	**67.86 / 80.21**	**8.12 / 14.88**	**86.80 / 95.12**	**71.39 / 85.11**	**91.80 / 98.20**	**63.33 / 73.33**	**70.00 / 76.67**	**66.67** / 73.33

Compared with DS-V3.2 Distill, which generates reasoning trajectories for the original source problems without structural decomposition, MINDLOOM demonstrates that composing new problems from thought modes provides additional benefit beyond trajectory-level distillation. Against MegaScience and OpenThought, two external large-scale reasoning datasets, MINDLOOM achieves the best pass@1 on the General-and-STEM benchmarks (CS-Bench, ChemBench, HLE, MedQA, SciBench) for every model size while using a controlled SFT budget of 9,230 examples, suggesting that principled compositional synthesis can be more data-efficient than large-scale external data collection. The Qwen3.5-9B results further confirm that the gains generalize to stronger base models: MINDLOOM achieves the best pass@1 on almost all nine benchmarks for this configuration.

Additional comparisons test inverse-frequency scarcity, source rebalancing, and K=16 against the main operating point (Appendix J).

4.3 Ablation Study

We conduct ablation studies to evaluate the contribution of each component of MINDLOOM. Table 2 presents results when individual components are removed or replaced with simpler alternatives.

Each ablation variant removes or replaces one pipeline component while keeping the rest fixed. The Qwen3-4B result for *w/o Scarcity* shows that removing the scarcity reward from selector scoring weakens most metrics relative to the full MINDLOOM pipeline, including CS-Bench, ChemBench, HLE, MATH-500, and HMMT-Feb. This is consistent with the intended role of the scarcity term: it prevents synthesis from repeatedly selecting common high-similarity thought modes and improves coverage over less frequent reasoning patterns. The remaining ablation variants test complementary hypotheses: *w/o Filter* isolates the value of rollout-based judging before SFT conversion, *Random Retriever* tests whether learned semantic retrieval matters beyond random candidate pools, and *w/o Reverse Eng.* tests whether extracting structured thought modes from solutions is better than building the thought-mode bank directly from raw rollouts. The Qwen3-8B results follow a consistent pattern: the full pipeline achieves the best or near-best scores on all nine benchmarks, and the *w/o Filter* variant again shows the largest degradation, confirming that rollout-based filtering is the single most impactful component across model sizes.

4.4 Analysis

Stage-level error containment. The pipeline rejects malformed extraction tuples before retrieval, rejects incompatible synthesis steps before adding them to a chain, and admits only judged-correct

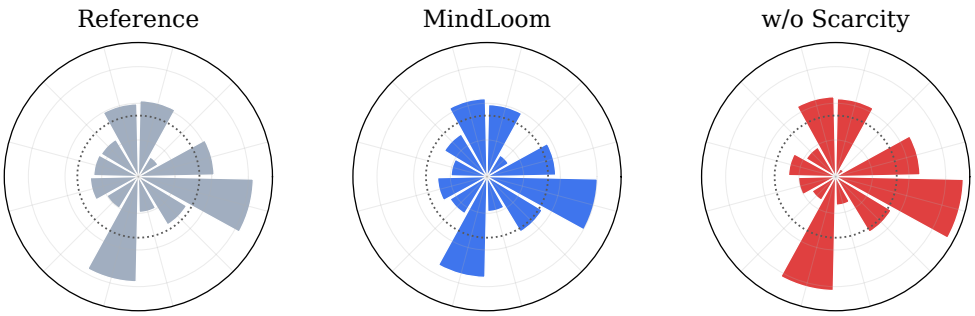


Figure 2: Coverage over the twelve thought-mode clusters (Appendix H). Polar bars give the proportion of selector calls landing in each cluster; the dotted ring marks uniform coverage at $1/12$. MINDLOOM tracks the reference distribution; *w/o Scarcity* skews toward popular clusters and starves the smallest one.

rollouts after source filtering into SFT. In the main run, 4,022 of 12,636 synthesis steps were rejected at the first step; 11,427 of 11,433 rollout records were complete, 10,854 remained after held-out-source filtering, and 9,230 became SFT records (Appendix E.7).

Distribution Analysis. We verify that the distribution-aligned selector broadens reasoning-pattern coverage without sacrificing per-instance quality. Figure 2 contrasts the realized distribution P_{gen} against the reference P_{ref} across the twelve thought-mode clusters defined in Appendix H. MINDLOOM tracks P_{ref} closely, while *w/o Scarcity* concentrates on a few popular clusters and starves rarer ones; the same scarcity removal also underperforms MINDLOOM on hard mathematical reasoning in Table 2. The two views together indicate that the scarcity term improves benchmark performance *by* broadening reasoning coverage, not at its expense.

Difficulty Analysis. We characterize the difficulty profile of synthesized questions from rollout outcomes. Roughly half of the synthesized questions are not fully solved by the rollout model under three independent rollouts, and about 15% land in the partial-success zone that supplies the frontier-like SFT records (full counts in Appendix E.7). The synthesis pipeline therefore produces a meaningful spread of difficulties rather than uniformly easy or unsolvable problems, validating the rollout-based filtering rationale in Section 3.5.

Case Studies. We trace one full pipeline run end-to-end to verify that each method stage executes the logic specified in Section 3. The reverse-engineering trace (Appendix G.1) walks through the polar-area decomposition: a self-contained seed question is constructed from the last solution window, and one iterative absorption step recovers the full polar-area integral, with the extracted thought mode separating transferable structure from instance-specific bounds. The compositional-synthesis trace (Appendix G.2) starts from a risk-free-rate seed and applies three thought modes drawn from three distinct clusters, recovering a competition-level mathematical-finance problem and confirming that the distribution-aligned selector avoids repeated cluster selection. Failure-mode examples (Appendix G.3) further document the synthesis-incompatibility and reverse-engineering-saturation cases that the pipeline correctly rejects.

Hyperparameter Sensitivity. Two hyperparameters most directly govern the pipeline: the scarcity weight α that balances semantic compatibility against coverage in distribution-aligned synthesis (Section 3.4), and the reverse-engineering window length w that controls thought-mode granularity (Section 3.2). Figure 4 (Appendix I) sweeps each on Qwen3-4B and Qwen3.5-4B with the other held fixed: performance peaks at the defaults (α=0.65, w=2) on both models and varies smoothly within a wide neighborhood, indicating that the operating point is robust rather than finely tuned. Both axes do degrade at the extremes, in directions consistent with the design rationale: very small α removes the compatibility signal and the selector applies thought modes the question state cannot support, while very large α removes scarcity and collapses generation onto popular clusters, recovering the *w/o Scarcity* regime; very small w fragments the bank into hyper-local transformations that lose reusable structure, while very large w bundles multiple reasoning steps into a single mode and erodes the atomicity that retrieval and composition rely on.

5 Conclusion

We have proposed the concept of thought modes, defined as atomic knowledge-reasoning transformations that can be composed to modulate problem complexity. We have presented MINDLOOM, a framework for synthesizing frontier-level reasoning data through compositional thought mode engineering. MINDLOOM extracts thought mode chains from existing hard problems via reverse engineering, learns to retrieve compatible thought modes through embedding-based matching, composes new problems through distribution-aligned iterative synthesis, and prepares training data through rollout-based judging, source-aware benchmark filtering, and judged-correct SFT conversion. Experiments on nine benchmarks across multiple model families show that models trained on MINDLOOM-generated data improve pass@3 over base-model inference in all reported model-size settings and achieve leading pass@1 results on the General-and-STEM benchmarks, with the strongest gains on competition-style mathematical reasoning. Ablation studies indicate the contribution of each pipeline component, and further analysis points to broad coverage of diverse reasoning types with useful difficulty control.

Acknowledgments and Disclosure of Funding

This work was supported by the National Natural Science Foundation of China (No. 62276002), New Generation Artificial Intelligence-National Science and Technology Major Project (No.2025ZD0123800), the National Natural Science Foundation of China (No. 62595734), the Key Laboratory of High Confidence Software Technologies (Peking University), the Ministry of Education, and the Center for Data Space Technology and System, Peking University.

References

[1] Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 2023.

[2] Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating LLMs on uncontaminated math competitions. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL `https://openreview.net/forum?id=y0zL9IZxZ7`.

[3] Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. Theoremqa: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7889–7901, 2023.

[4] Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems, 2025.

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

[6] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *IEEE Transactions on Big Data*, 2025.

[7] Xeron Du, Yifan Yao, Kaijing Ma, and Others. SuperGPQA: Scaling LLM evaluation across 285 graduate disciplines. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL `https://openreview.net/forum?id=6WgflzYQpf`.

[8] Run-Ze Fan, Zengzhi Wang, and Pengfei Liu. Megascience: Pushing the frontiers of post-training datasets for science reasoning, 2025.

[9] Center for AI Safety Phan Long agibenchmark@ safe. ai 1 Gatti Alice 1 Li Nathaniel 1 Khoja Adam 1 Kim Ryan 1 Ren Richard 1 Hausenloy Jason 1 Zhang Oliver 1 Mazeika Mantas 1 Hendrycks Dan dan@ safe. ai 1. A benchmark of expert-level academic questions to assess ai capabilities. *Nature*, 649(8099):1139–1146, 2026.

[10] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, et al. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai, 2024.

[11] Yuxian Gu, Li Dong, Hongning Wang, Yaru Hao, Qingxiu Dong, Furu Wei, and Minlie Huang. Data selection via optimal control for language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=dhAL5fy8wS.

[12] Etash Kumar Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Rea Sprague, Ashima Suvarna, Benjamin Feuer, Leon Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah M Pratt, Vivek Ramanujan, Jon Saad-Falcon, Stutee Acharya, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alex Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=7xjoTuaNmN.

[13] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[14] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, 2024.

[15] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.

[16] Maxwell Jia. AIME 2024. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/Maxwell-Jia/AIME_2024.

[17] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

[18] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[19] Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models, 2025.

[20] Junteng Liu, Yuanxiang Fan, Zhuo Jiang, Han Ding, Yongyi Hu, Chi Zhang, Yiqi Shi, Shitong Weng, Aili Chen, Shiqi Chen, et al. Synlogic: Synthesizing verifiable reasoning data at scale for learning logical reasoning and beyond, 2025.

[21] Liangxin Liu, Xuebo Liu, Derek F Wong, Dongfang Li, Ziyi Wang, Baotian Hu, and Min Zhang. Selectit: Selective instruction tuning for llms via uncertainty-aware self-reflection. volume 37, pages 97800–97825, 2024.

[22] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. 2023.

[23] James B McQueen. Some methods of classification and analysis of multivariate observations. In *Proc. of 5th Berkeley Symposium on Math. Stat. and Prob.*, pages 281–297, 1967.

[24] Adrian Mirza, Nawaf Alampara, Sreekanth Kunchapu, Martiño Ríos-García, Benedict Emoekabu, Aswanth Krishnan, Tanya Gupta, Mara Schilling-Wilhelmi, Macjonathan Okereke, Anagha Aneesh, et al. Are large language models superhuman chemists?, 2024.

[25] Open-R1 Team. OpenR1-Math-220k: A Large-Scale Math Dataset for Reinforcement Learning. `https://huggingface.co/datasets/open-r1/OpenR1-Math-220k`, 2025.

[26] OpenAI. GPT-4 Technical Report, 2024. URL `https://arxiv.org/abs/2303.08774`.

[27] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023.

[28] Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Jiatong Yu, Yinghui He, Nan Rosemary Ke, Michael Mozer, Yoshua Bengio, Sanjeev Arora, et al. Ai-assisted generation of difficult math questions, 2024.

[29] Xiaoshuai Song, Muxi Diao, Guanting Dong, Zhengyang Wang, Yujia Fu, Runqi Qiao, Zhexu Wang, Dayuan Fu, Huangxuan Wu, Bin Liang, Weihao Zeng, Yejie Wang, Zhuoma GongQue, Jianing Yu, Qiuna Tan, and Weiran Xu. CS-bench: A comprehensive benchmark for large language models towards computer science mastery. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=fjEZ2LPceZ`.

[30] Qwen Team. Qwen3.5: Towards Native Multimodal Agents. `https://qwen.ai/blog?id=qwen3.5`, February 2026. Accessed: 2026-04-13.

[31] Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R. Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. Scibench: evaluating college-level scientific problem-solving abilities of large language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[32] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. volume 37, pages 95266–95290, 2024.

[33] Alexander Wettig, Aatmik Gupta, Saumya Malik, and Danqi Chen. Qurating: Selecting high-quality data for training language models, 2024.

[34] Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In *International Conference on Machine Learning (ICML)*, 2024.

[35] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=zeFrfgyZln`.

[36] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=CfXh93NDgH`.

[37] Xinnuo Xu, Rachel Lawrence, Kshitij Dubey, Atharva Pandey, Risa Ueno, Fabian Falck, Aditya V. Nori, Rahul Sharma, Amit Sharma, and Javier Gonzalez. RE-IMAGINE: Symbolic benchmark synthesis for reasoning evaluation. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=QJPlODWajD`.

[38] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report, 2025.

[39] Cehao Yang, Xueyuan Lin, Xiaojun Wu, Chengjin Xu, Xuhui Jiang, Honghao Liu, Hui Xiong, and Jian Guo. Select2reason: Efficient instruction-tuning data selection for long-cot reasoning, 2025.

[40] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, et al. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=N8N0hgNDRt.

[41] Shaoxiong Zhan, Yanlin Lai, Ziyu Lu, Dahua Lin, Ziqing Yang, and Fei Tan. Mathsmith: Towards extremely hard mathematical reasoning by forging synthetic problems with a reinforced policy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 34602–34610, 2026.

[42] Xuemiao Zhang, Can Ren, Chengying Tu, Rongxiang Weng, Shuo Wang, Hongfei Yan, Jingang Wang, and Xunliang Cai. Expanding reasoning potential in foundation model by learning diverse chains of thought patterns. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3FQV4JHPpY.

[43] Zhehao Zhang, Jiaao Chen, and Diyi Yang. Darg: Dynamic evaluation of large language models via adaptive reasoning graph. volume 37, pages 135904–135942, 2024.

[44] Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, et al. Swift: a scalable lightweight infrastructure for fine-tuning, 2025.

[45] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9ZPegFuFTFv.

[46] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the association for computational linguistics: NAACL 2024*, pages 2299–2314, 2024.

[47] zwhe99. amc23. Hugging Face dataset, 2023. URL https://huggingface.co/datasets/zwhe99/amc23.

A Limitations and Future Work

MINDLOOM recombines thought modes extracted from the collected reference corpus, so the bank determines the available reasoning primitives and the range of synthesized problems. Difficulty labels come from three sampled rollouts and LLM-based judging. The reported full-parameter SFT experiments cover 4B–9B models from two model families. These settings define the scope of the results; broader corpora, human assessment, larger models, reinforcement-learning-based training, and multimodal synthesis offer directions for further study.

B Thought Mode Format and Examples

This section provides the complete formal specification of the thought mode representation and illustrative examples.

B.1 Formal Specification

Each thought mode $\mathcal{T}$ is represented as a four-component tuple $\mathcal{T} = (S_{\text{sum}}, S_{\text{det}}, K_{\text{gen}}, K_{\text{spec}})$. Table 3 describes each component.

Table 3: Components of a thought mode tuple.

Component	Description
S_{sum}	A concise, one-sentence summary of the transformation type. It describes what kind of reasoning complexity is added at a high level (e.g., "Introduce definite integral computation over a bounded domain").
S_{det}	A detailed description of the specific modification made to the problem. It specifies exactly how the problem is changed, including which explicit givens are removed and what new derivation is required.
K_{gen}	The general, domain-transferable knowledge required to handle this thought mode. This includes mathematical theorems, physical laws, algorithmic principles, or other reusable knowledge that applies beyond the specific problem instance (e.g., "Integration by parts: $\int u\,dv = uv - \int v\,du$").
K_{spec}	The problem-specific parameters and constraints involved in this transformation. This includes numerical values, boundary conditions, domain-specific setup, or other details that are unique to the particular problem (e.g., "Integration bounds: $[0, 2\pi]$; integrand involves $\sin^2(x)$").

The design rationale behind this four-component structure is to separate *what* a transformation does (S_{sum}, S_{det}) from *what knowledge* it requires (K_{gen}, K_{spec}), and to further separate *transferable* aspects (S_{sum}, K_{gen}) from *instance-specific* aspects (S_{det}, K_{spec}). This separation enables the retrieval model to match thought modes based on structural similarity (via S_{sum} and K_{gen}) while preserving enough detail (via S_{det} and K_{spec}) for accurate problem synthesis.

B.2 Extracted Example

A concrete example of a thought mode tuple is presented in the main text (Section 4.4). Here we provide additional context on the tuple format using the same polar-area example. The original problem asks for the area of the polar region $1 \le r \le 2$, $0 \le \theta \le \pi/2$. The reverse-engineering output first creates a seed question that gives the intermediate radial integral $\int_1^2 r\,dr = 3/2$ and asks only for the final outer integration. One extracted thought mode then removes this intermediate result and restores the need to set up and evaluate the polar-area integral. The full JSON output produced by the iterative evolution prompt is:

```
{
  "Q_next": "Find the area of the region ...",
  "answer": "3*pi/4",
  "solution_steps": ["Recall area formula ...", "..."],
  "S_sum": "[Knowledge+Reasoning]: Requires recalling
    the polar area formula ...",
  "S_det": "Replace the given radial-integral result
    with the full description of the region ...",
  "K_gen": ["area formula in polar coordinates",
    "setting up double integrals", ...],
  "K_spec": ["radial bounds [1,2]",
    "angular bounds [0, pi/2]"]
}
```

This output format matches the structured parsing described in Section 3.2.

C Reverse Engineering Details

C.1 Forward Inference Configuration

Before reverse engineering, we run forward inference on the reference corpus to obtain verified solution traces. DeepSeek V3.2 generates three independent rollouts per problem via the OpenAI-compatible API with the following system prompt:

```
You are an expert problem solver. Think step by step
and produce a JSON object.
Output format (strict JSON):
{"steps": ["step 1", "step 2", ...],
 "final_answer": "concise final result"}
Rules:
- Each step must be non-trivial and task-specific.
- Avoid macro steps or trivial restatements.
- Keep steps granular but meaningful.
- Do not add extra keys; do not wrap in Markdown.
```

Each rollout is then judged by the same LLM judge described in Appendix E.7. Problems are categorized by rollout outcomes: all-correct, partial (at least one correct), and all-wrong. Partial-success rollouts provide verified solution traces for reverse engineering. For all-wrong problems, we run a secondary inference pass with DeepSeek V3.2's thinking mode; judged-correct rescue rollouts provide additional traces for reverse engineering and targets for the standard SFT converter (Appendix E.8).

C.2 Solution Window Partitioning

Given a verified solution $S = [s_1, s_2, \ldots, s_m]$ with m steps, we partition it into k windows from the tail. Each window contains w consecutive solution steps (the *step size*). The last window W_k covers steps $[s_{m-w+1}, \ldots, s_m]$, the second-to-last window W_{k-1} covers $[s_{m-2w+1}, \ldots, s_{m-w}]$, and so on. If the total number of steps m is not evenly divisible by w, the first window W_1 contains the remaining $m - (k-1)w$ steps.

The step size w controls the granularity of thought modes. A smaller w yields finer-grained thought modes that capture more atomic reasoning steps, while a larger w produces coarser thought modes that bundle multiple reasoning steps together. We set $w = 2$ in our experiments, matching the reverse-engineering launcher used to build the current thought-mode bank (see Section C.6).

C.3 Seed Generation Prompt

The Seed operation takes the last solution window W_k and generates a minimal seed question Q_0. The prompt instructs the LLM to:

1. Identify all values and quantities that window W_k uses but does not derive (i.e., values computed in earlier windows).
2. Convert each such value into an explicit given in the seed question.
3. Formulate a self-contained question whose complete solution requires exactly the steps in W_k.
4. Verify that the seed question is independently solvable without any external information.

The implementation uses the following prompt skeleton. The system message defines the role as a problem designer and reverse-engineering specialist, explains that the model should backtrack from the final solution window, and emphasizes dependency isolation. The user message then provides the subject, original problem, final answer, full solution steps, and the current tail window. The required output is strict JSON:

```
{
  "seed_question": "...",
  "answer": "...",
  "solution_steps": ["..."]
}
```

The prompt explicitly instructs the model to identify upstream-dependent quantities used by the tail window, convert them into explicit givens, and ensure that the seed question can be solved using only the current window.

C.4 Iterative Evolution Prompt

The Absorb operation takes the current question Q_{i-1}, the next solution window W_{k-i}, and the remaining earlier solution steps. The prompt instructs the LLM to:

1. Identify which explicit given in Q_{i-1} is the result of the computation in window W_{k-i}.
2. Remove that given from the problem statement and modify the question so that the solver must derive the value.
3. Ensure the evolved question Q_i remains well-defined and solvable.
4. Extract the thought mode tuple $\mathcal{T}_i = (S_{\mathrm{sum}}, S_{\mathrm{det}}, K_{\mathrm{gen}}, K_{\mathrm{spec}})$ that describes the added reasoning requirement.

The iterative evolution prompt receives the original target problem, the current intermediate question, the upstream solution steps, and the next solution window to absorb. It asks the model to remove one explicit dependency from the current question and rewrite the problem so that the solver must derive it. The output is strict JSON with both the evolved question and the thought-mode tuple:

```
{
  "Q_next": "...",
  "answer": "...",
  "solution_steps": ["..."],
  "S_sum": "...",
  "S_det": "...",
  "K_gen": ["..."],
  "K_spec": ["..."]
}
```

The parser enforces these fields and validates that all string-list fields are lists of strings before the step is added to the chain.

C.5 Quality Control

To ensure the quality of extracted thought mode chains, we apply several automatic checks:

- **Candidate selection.** Reverse engineering is run only on source questions with useful successful rollouts: non-thinking partial-success records are selected directly, and all-wrong non-thinking

records are rescued only when the corresponding thinking-mode rollout has at least one judged-correct solution. All-correct, incomplete, and unrescued all-wrong records are skipped.

- **Strict structured parsing.** Seed outputs must contain `seed_question`, `answer`, and `solution_steps`; iterative outputs must contain `Q_next`, `answer`, `solution_steps`, `S_sum`, `S_det`, `K_gen`, and `K_spec`. The parser rejects invalid JSON, missing keys, and non-string-list knowledge fields.
- **Resume-safe deduplication.** Existing reverse-engineered outputs are keyed by question text or id and skipped on resumed runs, preventing duplicate chains from entering the retained file.

The retained reverse-engineering file contains 8,322 records, with 32,269 extracted thought-mode steps in total and an average chain length of 3.88. These records are selected from the 58,526 source problems through strict eligibility and quality criteria. Eligible records have partial-success non-thinking rollouts or judged-correct thinking-mode rescues of non-thinking all-wrong cases; all-correct and unrescued all-wrong records are skipped. Among eligible candidates, the structured parser enforces valid JSON and complete thought mode tuples, further ensuring extraction quality.

C.6 Step Size Analysis

The reverse-engineering window length w controls thought-mode granularity. We use $w = 2$ as the default: a smaller value creates very local transformations and inflates extraction cost, while a larger value merges multiple dependencies into a coarser, less reusable thought mode. Empirical sweeps over $w \in \{1, 2, 3, 4\}$ are reported jointly with the scarcity-weight sweep in Appendix I.

D Retrieval Model Details

D.1 Text Serialization Format

The problem state and thought mode are serialized into text for the embedding model as follows.

Retriever query encoding. The retriever uses the raw intermediate question text as the query-side input without a `[STATE]` prefix.

Selector state encoding. The distribution-aware selector embeds the current generation state as:

```
[STATE] Question: {question} || Answer: {answer}
        || Solution: {solution_steps_joined_by_bar}
```

Thought mode encoding:

```
[LOGIC] Summary: {S_sum} || Detail: {S_det}
        || K_gen: {K_gen} || K_spec: {K_spec}
```

where each field corresponds to one component of the thought mode tuple $\mathcal{T}$.

The `[STATE]` and `[LOGIC]` prefixes serve as type indicators in selector embeddings, while the fine-tuned retriever uses raw question text on the query side and the `[LOGIC]` representation on the thought-mode side. The delimiter `||` separates individual fields.

D.2 Model Architecture

The retrieval model is based on Qwen3-Embedding-0.6B, a pre-trained text embedding model. We use it as a bi-encoder: one encoder processes the problem state, and the same encoder processes the thought mode. Both inputs share the same encoder weights. In the reported experiments, FAISS uses inner-product search over raw, non-L2-normalized retriever embeddings. The selector's separate general-purpose embedding model computes cosine similarity explicitly.

D.3 ANCE Hard Negative Mining Procedure

We follow an ANCE-style protocol [35] for asynchronous hard negative mining:

1. **Initialization.** At the start of training, we build a FAISS [6] inner-product index using the initial model embeddings over all thought modes in the bank $\mathcal{B}$.
2. **Mining.** For each training pair $(Q_i, \mathcal{T}_i^+)$, we query the FAISS index with the embedding of Q_i and retrieve the top-k most similar thought modes, excluding the positive $\mathcal{T}_i^+$. These serve as hard negatives $\{\mathcal{T}_j^-\}$.
3. **Refresh.** After every R training steps, we re-encode all thought modes using the updated model and rebuild the FAISS index. This ensures the hard negatives remain informative as training progresses.

D.4 Training Hyperparameters

Table 4: Retrieval model training hyperparameters.

Hyperparameter	**Value**
Base model	Qwen3-Embedding-0.6B
Learning rate	2×10^{-5}
Batch size	2
Gradient accumulation	1
Training epochs	10, capped at 300 optimizer steps
Maximum optimizer steps	300
Hard negatives per sample (k)	5
Margin (γ)	0.2
FAISS refresh interval (R)	5 steps
Optimizer	AdamW
Weight decay	0.01
Warmup steps	100
Max sequence length	512
Precision / distributed training	bf16 with DeepSpeed ZeRO-2
Embedding normalization	Disabled in the main launcher
Train/validation split	80/20 by source id
Validation / checkpoint cadence	Evaluate every step; save every 10 steps
Checkpoint used by generation	step 20 in the main launcher

The checkpoint at step 20 is selected based on validation-set performance during training. Training runs for up to 300 optimizer steps with evaluation at every step; the step 20 checkpoint represents an early point where the model has learned meaningful compatibility signals without overfitting to the training pairs. Later checkpoints showed diminishing returns on the 20% held-out validation split.

E Synthesis and Rollout Filtering Details

E.1 Compositional Synthesis Algorithm

Algorithm 1 presents the full synthesis procedure.

E.2 Compatibility Verification

During synthesis, each thought mode application is verified for compatibility. The LLM-based Apply function checks the following criteria:

- **Coherence.** The evolved question Q_{i+1} must be a well-formed, grammatically correct question.
- **Solvability.** The evolved question must have a definite, derivable answer.
- **Relevance.** The thought mode $\mathcal{T}_{i+1}$ must meaningfully increase the difficulty of the question, not merely rephrase it.
- **Consistency.** The evolved question must not contradict any information from the original seed or previously applied thought modes.

Algorithm 1 Distribution-Aligned Compositional Synthesis

Require: Seed question Q_0, thought mode bank $\mathcal{B}$, retrieval model f, number of evolution steps n, clusters $\{c_1, \ldots, c_K\}$, reference distribution P_{ref}
Ensure: Evolved question Q_n, applied thought modes $[\mathcal{T}_1, \ldots, \mathcal{T}_n]$
Initialize generated-count vector from cache if available, otherwise zeros
for $i = 0$ to $n - 1$ **do**
  Encode problem state: $\mathbf{e}_q \leftarrow f(Q_i)$
  Retrieve top-m candidate thought modes: $\{\mathcal{T}_{j_1}, \ldots, \mathcal{T}_{j_m}\} \leftarrow \text{Retrieve}(\mathbf{e}_q, \mathcal{B}, m)$
  Compute selection scores via Equation 4
  Sample $\mathcal{T}_{i+1}$ via softmax sampling (Equation 5)
  Q_{i+1}, `is_compatible` $\leftarrow \text{Apply}(Q_i, \mathcal{T}_{i+1})$ {LLM-based transformation}
  if `is_compatible` is false and $i = 0$ **then**
    Record a seed/original fallback output for the first step
  else if `is_compatible` is false **then**
    Terminate synthesis for this seed
  end if
  Increment generated count for the selected cluster and persist cache
end for
return $Q_n, [\mathcal{T}_1, \ldots, \mathcal{T}_n]$

If any criterion is violated, the application is rejected. An incompatible first step falls back to the seed/original bookkeeping state, while an incompatible later step terminates synthesis for that seed. The generated record stores `is_compatible`, the selected `logic_id`, model provenance, and the evolved state for each accepted step. The generation stage produces 7,515 candidate records before later rollout filtering.

Across 12,636 total synthesis steps, 8,614 (68.2%) were judged compatible. The remaining 4,022 (31.8%) were first-step incompatibilities that triggered seed fallback. No later-step terminations occurred in the main run.

E.3 Synthesis Prompt Template

The Apply function uses a structured LLM prompt to fuse a selected thought mode with the current problem state. The system message defines the role as a problem designer and provides seven macroscopic difficulty enhancement strategies: (A) target upgrade (from single-point to structural or global quantities), (B) parameterization and generalization (from concrete values to general forms), (C) constraint coupling (introducing interdependent conditions that form simultaneous or boundary constraints), (D) reasoning chain extension (multi-stage dependencies where later steps require earlier results), (E) inverse or constructive formulations (deriving conditions from results or constructing objects under constraints), (F) uncertainty and robustness analysis (error bounds, worst-case analysis, sensitivity), and (G) algorithmic and resource constraints (requiring implementable solutions with complexity bounds).

The prompt enforces five correctness criteria: *solvability* (the evolved question must have a derivable solution), *uniqueness* (the answer must be determinate or disambiguated by a stated rule), *condition consistency* (new constraints must not contradict existing ones), *condition necessity* (every added condition must be explicitly used in the solution), and *closed-loop coherence* (the question, conditions, and solution must form a complete reasoning chain).

The user message provides the current problem state (question, answer, solution steps) and the selected thought mode (S_{sum}, S_{det}, K_{gen}, K_{spec}). The required output is strict JSON:

```
{
  "is_compatible": true/false,
  "question": "evolved problem statement",
  "answer": "final answer",
  "solution_steps": ["step 1", "step 2", ...],
  "S_sum": "logic summary",
  "S_det": "detailed modification description",
```

```
  "K_gen": ["general knowledge 1", ...],
  "K_spec": ["specific parameter 1", ...]
}
```

The `is_compatible` flag allows the model to reject incompatible pairings, which triggers the fallback logic described in Algorithm 1.

E.4 Synthesis Hyperparameters

Table 5: Compositional synthesis hyperparameters.

Hyperparameter	**Value**
Chat model	DeepSeek V3.2
Selector embedding model	Qwen3-Embedding-0.6B
Reverse engineering step size w	2
Evolution steps n	3
Retrieval candidate pool	top-20
K-Means clusters K	12
Similarity weight α	0.65
Selector temperature τ	1.0
Scarcity smoothing ϵ	10^{-3}
Cluster minimum similarity	0.0

E.5 SFT and Evaluation Hyperparameters

Table 6: Fine-tuning and evaluation hyperparameters.

Hyperparameter	**Value**
Fine-tuning	
Framework	ms-swift, full-parameter SFT
Precision	bf16
Maximum sequence length	4096
Per-device batch size	4
Gradient accumulation steps	8
Learning rate	3×10^{-5}
Warmup ratio	0.05
Training epochs	3
Evaluation	
Inference engine	vLLM
Rollouts per test item	3
Temperature	0.7
Maximum generation length	8096
Maximum model length	23000
Judging	API-based LLM judge

E.6 Data Pipeline Statistics and Domain Composition

This section traces how data flows through each pipeline stage and reports the domain composition at each step. Table 7 summarizes the pipeline stages and record counts.

Seed question sources. Seed questions are taken from the reverse-engineering output. For each original problem with a valid reverse-engineered chain, the pipeline stores a self-contained seed question, answer, and solution steps. All 8,322 reverse-engineered records are used as candidate seeds. During synthesis, previously generated outputs are deduplicated by question ID, so resumed runs skip already-processed seeds. Of the 8,322 candidates, 7,515 produced at least one accepted synthesis step.

Table 7: Data pipeline flow and record counts at each stage.

Stage	Records
Reference corpus	58,526
Forward inference rollouts	58,526
Reverse engineering	8,322
Compositional synthesis	7,515 seeds $\rightarrow$ 11,433 rollout records
Source-provenance filtering	10,854
Final SFT conversion	9,230

Domain composition of the final SFT set. Table 8 reports the domain breakdown of the 9,230 MINDLOOM SFT examples. The composition reflects the reference corpus distribution after provenance-based filtering removes records originating from held-out benchmark sources.

Table 8: Domain composition of the final 9,230 MINDLOOM SFT examples by original seed source.

Seed source	Count	%
SuperGPQA	6,378	69.1
MMLU-Pro	916	9.9
Hendrycks-MATH	876	9.5
ARC-AGI-2	461	5.0
OlympiadBench	247	2.7
TheoremQA	181	2.0
GPQA Diamond	71	0.8
AGIEval	53	0.6
MiniF2F	33	0.4
AMC 2023	10	0.1
AIME 2024	4	0.0
Total	**9,230**	**100.0**

The three provenance-filtered sources, HLE, SciBench, and AIME 2025, contributed 388, 188, and 3 records respectively to the pre-filtering pool of 11,433 records. Their removal yields a 94.9% retention rate.

Distillation baseline composition. The DS-V3.2 Distill baseline is constructed by prompting DeepSeek V3.2 with the original source problems using the same inference system prompt and rollout configuration as the forward inference stage (Appendix C.1). The conversion step selects the first judged-correct rollout for each problem to form an SFT record in the same ms-swift format as MINDLOOM records (Appendix E.8). This produces a pool of 40,937 distillation records. This pool is randomly subsampled to match the MINDLOOM budget. The domain composition of the full distillation pool is: SuperGPQA 43.9%, Hendrycks-MATH 24.2%, MMLU-Pro 21.5%, OlympiadBench 5.2%, and the remaining eight sources collectively 5.2%. Compared with the MINDLOOM SFT set (Table 8), the distillation pool is more evenly spread across the three largest sources rather than concentrated on SuperGPQA.

E.7 Rollout Filtering Details

Rollout Configuration. For each generated question Q, we perform three independent rollouts in the filtering and evaluation scripts. Filtering rollouts use the OpenAI-compatible provider configured in `.env`; evaluation rollouts use local vLLM inference with temperature 0.7 by default. The main evaluation launcher sets maximum new tokens to 8096, maximum model length to 23000, tensor parallel size to 2 when enough GPUs are visible, and uses batch size 16 for standard datasets and 4 for HLE. The pass rate is computed as the fraction of non-empty parsed rollouts that produce a correct final answer.

Automated Judging. We use DeepSeek V3.2 as the LLM-based judge to determine answer correctness. The judge receives the generated question Q, the reference answer stored with the generated record, the rationale or solution steps, and the model's response. The system message is:

```
You are a strict grader. For each problem, you receive
```

```
the question, the authoritative ground-truth answer,
and a short rationale explaining why that answer is
correct. You also receive a model's predicted answer.
Decide whether the predicted answer should be counted
as correct based on the question and the rationale.
Return JSON: {"is_correct": true/false}.

Guidance:
- Judge semantic correctness, not exact string match.
- Use the rationale to understand what counts as a
  valid answer for this question.
- If the predicted answer misses the required outcome,
  mark false.
- Do not add extra keys or text.
```

The user message concatenates the question, ground-truth answer, rationale, and model answer, followed by "`Respond ONLY with the JSON object.`". The same prompt is shared by filtering and evaluation.

Discrete Difficulty Labels and Conversion. The filter assigns difficulty labels from three judged rollouts: all-correct when all three are correct, partial when one or two are correct, and all-wrong when none is correct. The hard-item export retains partial and all-wrong questions for analysis. SFT conversion selects a judged-correct response from the non-thinking rollouts first, then from the paired thinking rollouts when available.

The generator produced 7,515 candidate records and 12,636 accepted or recorded step-level outputs. The non-thinking rollout file contains 11,433 generated rollout records, of which 11,427 are complete; after held-out-source filtering, 10,854 records remain. The reasoning rollout file contains 11,331 records, and the hard-item export yields 4,106 records from the partial and all-wrong categories. The final SFT conversion writes 9,230 non-thinking records and 9,067 thinking-format records. To keep SFT comparisons controlled by data quality rather than data quantity, the DS-V3.2 distillation pool is subsampled to match the MINDLOOM budget.

E.8 SFT Training Record Format

Each SFT training record follows the ms-swift conversation format with three messages. The user message embeds the inference system prompt followed by the generated question, and the assistant message contains the first judged-correct rollout response:

```
{"messages": [
  {"role": "system",
   "content": "You are a helpful assistant"},
  {"role": "user",
   "content": "You are an expert problem solver. Think
    step by step and respond with a strict JSON object.
    Output format:
    {\"steps\": [...], \"final_answer\": \"...\"}
    Rules: [step quality rules as in inference]
    Problem: {question}"},
  {"role": "assistant",
   "content": "{\"steps\": [...],
    \"final_answer\": \"...\"}"}
]}
```

F Reference Corpus and Evaluation Benchmarks

F.1 Source Datasets

The thought-mode reference bank is constructed from 58,526 source problems aggregated from 16 datasets across three source files. Table 9 lists each dataset, its approximate size, and its domain coverage.

Table 9: Source datasets used to construct the thought-mode reference bank.

Dataset	Domain	Approx. size
SuperGPQA [7]	Multi-domain expert QA	26,529
MMLU-Pro [32]	Multi-domain multiple choice	12,102
Hendrycks-MATH [15]	Competition mathematics	10,114
OlympiadBench [14]	Olympiad mathematics and science	2,665
HLE [9]	Expert-level reasoning	2,500
ARC-AGI-2 [4]	Abstraction and reasoning	1,120
AGIEval [46]	Standardized exam problems	1,000
TheoremQA [3]	Theorem-based reasoning	731
SciBench [31]	College-level science	692
MiniF2F [45]	Formal mathematical proofs	392
ProofNet [1]	Formal proofs	371
GPQA Diamond [27]	Graduate-level science QA	198
AMC 2023 [47]	AMC competition problems	40
AIME 2024 [16]	AIME competition problems	30
AIME 2025 [2]	AIME competition problems	30
MathArena Apex [2]	Competition mathematics	12
Total		**58,526**

F.2 Contamination Control

Before SFT conversion, the pipeline traces each synthesized item back to its seed-source provenance and removes generated examples whose origin matches a held-out benchmark source ID. Table 10 summarizes the filtering configuration.

Table 10: Provenance-based contamination filters applied before SFT conversion.

Filtered source ID	Reason	Overlapping benchmark
`opencompass__AIME2025`	Direct overlap	AIME 2025
`xw27__scibench`	Direct overlap	SciBench
`cais__hle`	Direct overlap	HLE

The Hendrycks-MATH records in our reference corpus are drawn from the training split of the dataset; we additionally remove the 500 questions that constitute the MATH-500 evaluation set to prevent any leakage. Because the remaining Hendrycks-MATH records have no overlap with MATH-500, they are retained in the SFT set. Similarly, the AGIEval records in our corpus do not overlap with any evaluation benchmark. The remaining evaluation benchmarks, CS-Bench, ChemBench, MedQA, HMMT February 2025, and HMMT November 2025, have no overlap with any source dataset in the reference corpus and therefore require no source-level filtering.

During evaluation, we skip all test items that require image inputs, as the current pipeline operates in a text-only setting. The reported test-item counts in Table 11 reflect the text-only subsets actually evaluated.

F.3 Evaluation Benchmarks

Table 11 summarizes the nine evaluation benchmarks. We report pass@1 and pass@3 for each benchmark, defined as the fraction of test items for which one or three independent rollouts produce at least one correct answer, respectively.

Table 11: Evaluation benchmarks.

Benchmark	Domain	Test items
CS-Bench [29]	Computer science	2,419
ChemBench [24]	Chemistry	2,542
MATH-500 [15]	Competition mathematics	500
HLE [9]	Expert-level reasoning	2,500
HMMT Feb. 2025 [2]	Competition mathematics	30
HMMT Nov. 2025 [2]	Competition mathematics	30
MedQA [17]	Medicine	1,273
AIME 2025 [2]	Competition mathematics	30
SciBench [31]	College-level science	692

G Extended Case Studies

This appendix gives concrete walkthroughs of the four-stage pipeline. Reverse-engineering examples (Appendix G.1) illustrate Section 3.2; compositional-synthesis examples (Appendix G.2) illustrate Section 3.4; failure-case analysis (Appendix G.3) documents the two main failure modes that the rollout-based filtering stage is designed to catch.

G.1 Reverse Engineering: Worked Examples

Example R1: Polar Area. The original problem asks for the area of the polar region $1 \leq r \leq 2$, $0 \leq \theta \leq \pi/2$. Reverse engineering first creates a seed question that gives the intermediate radial integral $\int_1^2 r\,dr = 3/2$ as an explicit given and asks only for the final outer integration. This seed is self-contained and solvable in a single step (Phase 1 of the extraction process). The subsequent iterative evolution step removes the radial-integral result and requires the solver to set up the full polar-area double integral from the region bounds. The extracted thought mode for this step is:

- S_{sum}: "Requires recalling the polar area formula, setting up the correct double integral with the given bounds, and performing a sequence of basic calculus computations."
- S_{det}: "Replace the given radial-integral result with the full description of the region in polar coordinates and require the solver to compute the integral from the bounds."
- K_{gen}: area formula in polar coordinates, setting up double integrals, evaluating definite integrals.
- K_{spec}: radial bounds $[1, 2]$, angular bounds $[0, \pi/2]$.

This example shows the separation between transferable reasoning structure (S_{sum}, K_{gen}) and instance-specific details (S_{det}, K_{spec}). The K_{gen} component ("polar area formula, double integrals") can guide retrieval for other calculus problems, while K_{spec} records the concrete numerical setup.

Example R2: Combinatorial Probability. The original problem asks: "Bob and Alice each have a bag that contains one ball of each of the colors, blue, green, orange, red, and violet. Alice randomly selects one ball from her bag and puts it into Bob's bag. Bob then randomly selects one ball from his bag and puts it into Alice's bag. What is the probability that after this process the contents of the two bags are the same?" The answer is $1/3$.

Seed (Q_0). The seed question provides the intermediate state explicitly: after Alice gives Bob a ball of color X, Bob's bag contains 6 balls, of which exactly 2 are color X. The question asks only for the probability that Bob picks a ball of color X from his 6 balls. The solution is $2/6 = 1/3$ in a single step.

Step 1. The seed is evolved to ask the full restoration question: what is the probability that the final bag contents match the initial state? The solver must now deduce that Bob must return the duplicate color and then compute the conditional probability $2/6$. The extracted thought mode is:

- S_{sum}: "Requires analyzing the necessary condition for the final bag states to be identical and performing the conditional probability calculation within the symmetry framework."
- K_{gen}: conditional probability, set reasoning for state-matching, symmetry arguments.
- K_{spec}: five colors, two-step transfer process, bags must return to initial composition.

Step 2. The necessary condition ("Bob must return color X") is removed from the problem statement; the solver must derive it by analyzing the state transitions. K_{gen} adds state transition analysis in combinatorial processes.

Step 3. The intermediate state description ("Bob's bag has 6 balls with a duplicate") is removed; only the initial identical-bags condition and the transfer process remain. The solver must deduce the intermediate state from first principles.

Step 4. The final evolved question recovers the original problem: no intermediate state, no condition hints. The full chain length is 4, with each step removing one layer of explicit scaffolding. The progression illustrates how reverse engineering decomposes a problem by successively re-introducing dependencies that the seed had converted into explicit givens.

G.2 Compositional Synthesis: Worked Examples

Each synthesis record stores the seed, up to three applied steps, selected logic IDs, thought-mode metadata, model provenance, compatibility flags, and the evolved question/answer/solution state at each step. We walk through one three-step synthesis trace in detail.

No-Arbitrage / Risk-Free Rate Derivation. The seed question is a risk-free rate computation problem from mathematical finance. It provides a pre-derived equation $-0.1\lambda = -0.05$ and asks to solve for λ, then substitute into the no-arbitrage condition to find $r = 0.02$. Three thought modes are then applied, each drawn from a different cluster of the taxonomy in Table 12.

Step 1 (Cluster 5, Algebraic Manipulation). The selected thought mode originates from the same problem's reverse-engineering chain and has S_{sum}: "The user must now understand the derivation of the key linear equation by performing the subtraction of the two no-arbitrage conditions." After application, the evolved question removes the pre-derived equation and requires the solver to eliminate r from two given no-arbitrage equations. Compatibility: `true`.

Step 2 (Cluster 3, Physics & Engineering). The selected thought mode originates from a photoresistor calibration problem and has S_{sum}: "Absorption of the explicit equation-solving step for the linear model. The solver must now set up and algebraically solve the system of two linear equations." After application, the numerical equations are also removed; the solver must substitute (μ_i, σ_i) into the general relationship before solving. Compatibility: `true`.

Step 3 (Cluster 4, Applied Quantitative). The selected thought mode requires theoretical derivation: "The risk-neutral drift condition is now required to be applied explicitly, rather than being implicitly used." The final question asks the solver to derive the no-arbitrage relationship from first principles using Itô's lemma and Girsanov's theorem, set up the system, and solve. Compatibility: `true`.

The three steps draw from three distinct clusters (5, 3, 4), illustrating how the distribution-aligned selector avoids repeated cluster selections. The final question is substantially harder than the seed: it requires stochastic calculus foundations in addition to algebraic manipulation.

G.3 Failure Case Analysis

The pipeline exposes two main failure modes. Reverse engineering can fail when the model output is invalid JSON or omits required tuple fields; such records are rejected by the parser. Synthesis can fail when the selected thought mode is incompatible with the current question state; the generation parser requires an explicit `is_compatible` flag, and incompatible later steps terminate the chain rather than adding an ill-formed question to the output.

Synthesis incompatibility. A right-triangle cosine seed question ("In right triangle DEF with right angle at D, $DE = 24$, $EF = 25$. What is $\cos(E)$?") was paired with a thought mode from a tangent-simplification problem whose S_{sum} requires "combining fraction arithmetic and the simplification of a complex fraction with nested radicals." The K_{gen} of the applied mode includes sum-of-tangent identities, product-to-sum conversions, and denominator rationalization. These are fundamentally incompatible with the seed: a basic adjacent-over-hypotenuse calculation has no intermediate expressions to rationalize and no trigonometric sums to simplify. The model correctly set `is_compatible=false` and the pipeline recorded a seed-fallback output for this step.

Reverse engineering saturation. In a chain extracted from the absolute-value optimization problem "Let $f(x) = |x - p| + |x - 15| + |x - p - 15|$, where $0 < p < 15$. Determine the minimum value of $f(x)$ for $x \in [p, 15]$," the reverse engineering process produced a 4-step chain. However, at step 3 the model returned the same question text as step 2, with empty K_{gen} and K_{spec} fields. The S_{det} explanation noted that "the interpretation step was already implicitly included in the reasoning chain of the current question." This represents a saturation failure: the problem's solution structure had been fully decomposed by earlier steps, and further absorption could not produce a genuinely simpler intermediate question. The empty knowledge fields signal this degeneracy to downstream processing.

H Thought Mode Taxonomy and Statistics

The distribution-aligned selector uses K=12 K-Means clusters over thought-mode embeddings. Table 12 provides a taxonomy of these clusters based on the S_{sum} fields of the thought modes assigned to each cluster. Labels are derived by inspecting representative thought modes near each cluster center.

Table 12: Thought mode cluster taxonomy. Counts reflect the total number of times each cluster was selected during the main synthesis run (43,384 total selections). The three representative S_{sum} excerpts for each cluster are drawn from thought modes closest to the cluster center.

ID	Label	Count	%	Representative S_{sum} excerpt
0	Factual & Historical Recall	4,483	10.3	Recall a biographical fact; chronological deduction
1	Geometric & Vector Reasoning	1,361	3.1	Dihedral angle volume formula; Law of Sines
2	Biomedical & Clinical	4,495	10.4	Receptor blockade mechanism; culture media
3	Physics & Engineering	4,312	9.9	Wedge product; PSD first-null derivation
4	Applied Quantitative	2,575	5.9	Duration calculation with rounding; convergence
5	Algebraic Manipulation	2,642	6.1	Expand and simplify; Hessian determinant
6	Grid Pattern Recognition	2,844	6.6	Coordinate-based diamond augmentation; rotation
7	Discrete Math & Structures	2,186	5.0	Combinatorial counting under fold constraints
8	Life Sciences & Interdisciplinary	6,224	14.3	Gene epistasis; codon mutation reasoning
9	Number Theory & Counting	2,120	4.9	Legendre's formula; palindrome enumeration
10	Physical Chemistry	3,338	7.7	Bohr energy derivation; van der Waals relation
11	Social Sciences & Conceptual	6,804	15.7	JTB epistemology; economic model elimination
	Total	**43,384**	**100**	

Figure 3 extends the body view (Figure 2) with the full per-cluster proportions for the three selector variants discussed in Section 4.4. MINDLOOM stays close to P_{ref} across all twelve clusters, whereas *w/o Scarcity* systematically over-samples the most common clusters and underrepresents rare ones such as Geometric & Vector Reasoning (Cluster 1).

I Hyperparameter Sensitivity Sweep

Two hyperparameters most directly govern the pipeline: the scarcity weight α in distribution-aligned synthesis (Section 3.4), which mediates the balance–quality tradeoff, and the reverse-engineering window length w (Section 3.2; see also Appendix C.6), which controls thought-mode granularity. We sweep each in turn on two foundation models (Qwen3-4B and Qwen3.5-4B) while holding the other at its default, and report the average pass@3 over the nine benchmarks. The α=1.0 row in the α sweep coincides with the *w/o Scarcity* ablation in Table 2; the default operating point (α=0.65, w=2) coincides with the MINDLOOM row in Table 1.

The two-parameter scope reflects the dominant axes of pipeline behavior: α governs the selector's exploration–exploitation tradeoff (already isolated in Table 2's scarcity ablation), and w governs the granularity of the upstream thought-mode bank itself. The additional K comparison is reported in Appendix J; n and τ remain fixed in these studies.

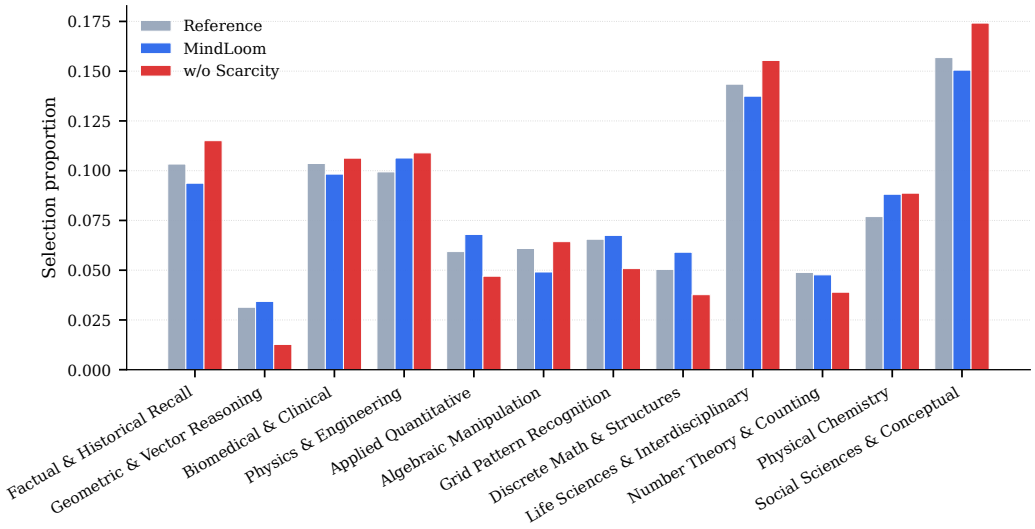


Figure 3: Per-cluster selection proportions for the three selector variants. Cluster labels follow Table 12.

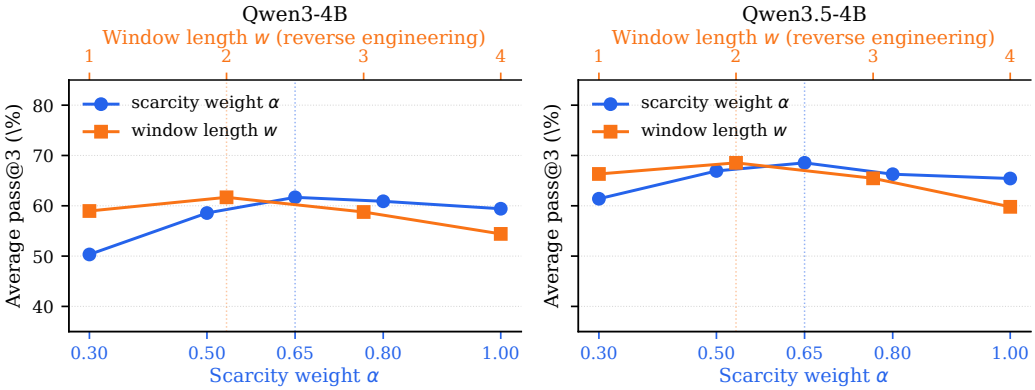


Figure 4: Hyperparameter sensitivity sweeps on Qwen3-4B (left) and Qwen3.5-4B (right). Bottom axis: scarcity weight α (default 0.65). Top axis: reverse-engineering window length w (default 2). Vertical y-axis: average pass@3 across the nine benchmarks. Dotted vertical lines mark the defaults.

J Selector and Source-Composition Comparisons

The following Qwen3.5-4B comparisons report controlled changes to selector and source composition. Each cell reports pass@1/pass@3 in percent. The inverse-frequency comparison keeps K=12, α=0.65, ϵ=10^{-3}, the retrieval candidates, generation pipeline, SFT budget, and evaluation protocol fixed. It replaces Equation 4's scarcity reward with

$$r_{\mathrm{IF}}(c_j) = \frac{(P_{\mathrm{gen}}(c_j)+\epsilon)^{-1}}{\max_{1\le k\le K}(P_{\mathrm{gen}}(c_k)+\epsilon)^{-1}}, \qquad \mathrm{score}_{\mathrm{IF}} = \alpha\,\mathrm{sim} + (1-\alpha) r_{\mathrm{IF}}. \tag{6}$$

Equation 4 is higher than inverse frequency in six of the ten displayed metrics and lower in four. Both scarcity rewards outperform similarity-only selection on most displayed benchmarks, while the bounded tanh-log form yields the strongest AIME 2025 pass@1.

To test dependence on the dominant source, we construct a 6,000-example SFT set from the same synthesized pool: all 2,852 non-SuperGPQA records plus 3,148 sampled SuperGPQA records. This lowers the SuperGPQA share from 69.1% to 52.5% while also reducing total training volume. Table 14 shows that the gains over the base model persist under this joint change in source mixture and training volume.

Table 13: Scarcity-function comparison on five representative benchmarks (pass@1/pass@3, %).

Selector	CS-Bench	HLE	SciBench	MATH-500	AIME 2025
Similarity only	69.33/80.69	5.96/12.60	63.44/75.29	89.20/95.80	43.33/63.33
Inverse frequency	70.00/81.50	7.00/14.00	65.50/76.50	90.00/96.00	46.67/66.67
Equation 4	69.70/81.11	7.56/14.04	66.18/79.62	89.80/96.40	60.00/63.33

Table 14: Source-rebalancing comparison on Qwen3.5-4B (pass@1/pass@3, %).

Setting	CS-Bench	HLE	SciBench	MATH-500	AIME 2025
Base	66.27/79.61	5.96/11.08	53.17/67.74	87.80/94.60	36.70/50.00
MindLoom (6,000)	67.00/80.50	6.50/13.00	62.50/75.00	88.00/96.00	53.33/60.00
MindLoom (9,230)	69.70/81.11	7.56/14.04	66.18/79.62	89.80/96.40	60.00/63.33

With the remaining configuration fixed, $K=12$ exceeds $K=16$ in five of six displayed metrics; $K=16$ is slightly higher on SciBench pass@1.

Table 15: Cluster-count sensitivity on Qwen3.5-4B (pass@1/pass@3, %).

K	CS-Bench	SciBench	AIME 2025
12	69.70/81.11	66.18/79.62	60.00/63.33
16	68.50/80.00	66.50/76.50	53.33/56.67

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The three contributions stated in Section 1 are directly supported by the main results in Section 4.2 and the ablations in Section 4.3.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix A specifies the source-corpus scope, rollout-based difficulty criterion, evaluated model sizes, and directions for broader study.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper presents an empirical method and does not include formal theorems or proofs.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The four-stage pipeline is described in Section 3; prompts, hyperparameters, SFT/evaluation protocols, source datasets, and contamination filters are listed in Appendices C–F.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The linked repository provides the implementation and instructions for data construction, model training, and evaluation.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 summarizes the experimental setup; full hyperparameters for retrieval training, synthesis, SFT, and evaluation are in Appendices D and E.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper reports pass@1 and pass@3 over fixed test sets using three rollouts per item; it provides no confidence intervals or run-to-run uncertainty estimates.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [No]

Justification: Section 4.1 and Appendix E specify software, training epochs, and inference settings; worker types, GPU memory, and execution times are outside the reported configuration.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses publicly available academic benchmarks and source datasets, does not involve human subjects, and complies with the NeurIPS Code of Ethics.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The work studies reasoning-data synthesis for small models; Appendix A discusses its research scope, and the potential misuse of generated reasoning data follows that of general LLM training data.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The released implementation is a reasoning-data synthesis pipeline, rather than a pretrained model or scraped sensitive dataset.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: The source datasets and benchmarks are cited; the paper does not include an asset-by-asset inventory of licenses and terms of use.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [No]

    Justification: The public repository documents the implementation and provides instructions for data construction, model training, and evaluation.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing or research with human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The work does not involve human subjects, so IRB approval is not required.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are used as components of the experimental pipeline (forward inference, reverse engineering, synthesis, and rollout judging); they were also used to fix grammar errors during manuscript writing.