Autodata: An agentic data scientist to create high quality synthetic data

Ilia Kulikov[†], **Chenxi Whitehouse**[†], **Tianhao Wu**[†], **Yixin Nie**[†], **Swarnadeep Saha, Eryk Helenowski, Weizhe Yuan, Olga Golovneva, Jack Lanchantin, Yoram Bachrach, Jakob Foerster, Xian Li, Han Fang, Sainbayar Sukhbaatar, Jason Weston**

FAIR at Meta [†]Joint first author

We introduce Autodata, a general method that enables AI agents to act as data scientists who build high quality training and evaluation data. We show how to train (meta-optimize) such a data scientist agent, so that it learns to create even stronger data. We describe the overall formulation, and a specific practical implementation, Agentic Self-Instruct. We conduct experiments on computer science research tasks, legal reasoning tasks and reasoning with mathematical objects, where we obtain improved results compared to classical synthetic dataset creation methods. Further, meta-optimizing the data scientist agent itself delivers an even larger performance uplift. Agentic data creation provides a way to convert increased inference compute into higher quality model training. Overall, we believe this direction has the potential to change the way we build AI data.

Date: June 25, 2026
Correspondence: kulikov@meta.com, jase@meta.com

Meta

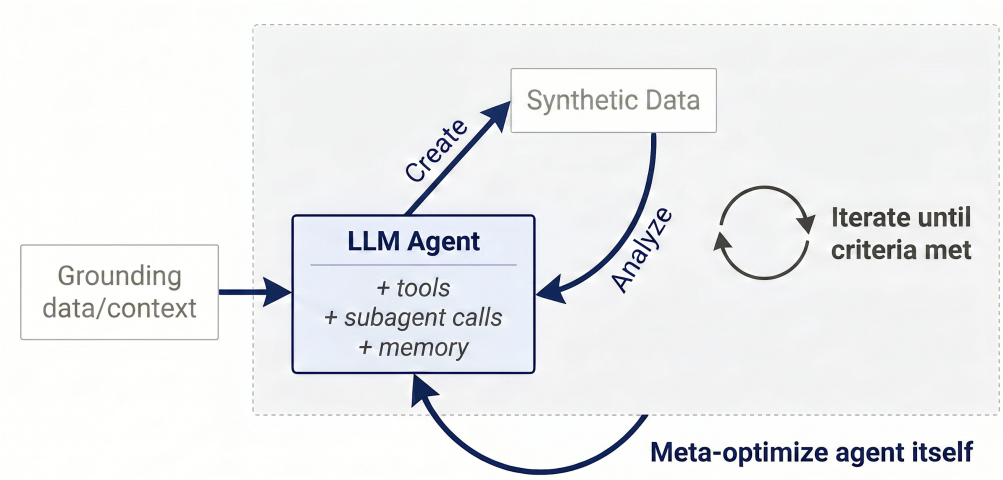

Figure 1 Autodata pipeline. The framework employs an autonomous agent that emulates the role of a data scientist, iteratively generating data, conducting qualitative inspection and quantitative performance evaluation, synthesizing insights, and updating the data-generation recipe. The agent itself can be trained to be better at the data scientist role using the same criteria used in the inner loop. This cyclical process aims to progressively enhance data quality; the diagram depicts the general workflow underlying possible instantiations.

1 Introduction

Progress at the AI frontier increasingly depends on high-quality training data and benchmarks that continue to challenge models. The initial foundation for training current AI systems is human-written training data. However, increasingly performance improvements are derived from synthetic data created by the model itself. Synthetic data addresses several practical challenges: it facilitates the generation of edge cases and long-tail scenarios that are underrepresented in real corpora, reduces the difficulty and latency associated with manual

labeling, and can potentially produce more challenging data than the human-generated data distribution.

With the introduction of large language models (LLMs) and the ability to use in-context learning and instruction following, Self-Instruct (Wang et al., 2023) emerged as a method to create synthetic data through zero or few-shot prompting. Grounded Self-Instruct (Lupidi et al., 2024; Yuan et al., 2025) extended that to ground on documents and other sources to reduce hallucination and increase diversity. Further, methods like CoT Self-Instruct (Yu et al., 2025) extended that to use Chain-of-Thought reasoning during the generation process to help construct more complex tasks more accurately. Finally, so called "self-challenging" methods (Zhou et al., 2025) allowed a challenger agent to interact with tools before proposing a task and accompanying evaluation functions. However, none of these methods control the difficulty and quality of the data directly, motivating approaches such as filtering (Yu et al., 2025), evolution (Xu et al., 2024) and refinement (Shah et al., 2024).

In this work, we introduce Autodata, which generalizes all the above described methods. An agent acting as a data scientist is tasked with the act of constructing and curating data, performing the actions a human data scientist would take in order to create high quality data: where both building benchmark data and training data are use cases. This process includes both an initial iteration of data creation, followed by an analysis phase "eyeballing" the data as well as measuring its performance, constructing learnings, and then iterating with an improved recipe to create better data. Further, we show how to train (meta-optimize) this agentic system (outer loop) to be optimal as a data scientist (inner loop). While much of the recent work on autoresearch (Karpathy, 2026) has concentrated on agentic methods for architectural or training recipe improvements, we posit that focusing on *data* is likely to play an equally important, if not more important, role in future progress.

In our experiments, we focus on a particular implementation of Autodata, Agentic Self-Instruct, and show that it provides strong results across a diverse set of tasks. We conduct experiments on computer science research tasks, legal reasoning tasks and reasoning with mathematical objects, where in each case we obtain improved results compared to classical synthetic dataset creation methods. Further, meta-optimizing the data scientist agent itself delivers an even larger performance uplift.

As state-of-the-art LLMs become ever-stronger, there is concern that existing tasks or synthetic data methods cannot produce tasks that are challenging enough to make further progress. Autodata, via agentic data creation, provides a way to convert increased inference compute into higher quality model training to produce such challenging data. Hence, we believe that this direction has the potential to change the way we create new tasks and benchmarks to advance the frontier.

2 Autodata

The high-level design of Autodata is shown in Figure 1, where various instantiations can be built from this template. The overall loop consists of the following components.

Data Creation. The autodata agent grounds on some provided data (e.g. specific documents from math, legal, coding etc. or another useful data source, depending on the task) to help create the data. The agent can then use tools or existing skills/learnings it has previously acquired and inference time compute to create training or evaluation data for model training and benchmarking. Importantly, this creation step can be repeated after subsequent analysis and learnings to improve the data even further.

Data Analysis. Given the data that the agent has created, it can then analyze the data for learnings on what it did right versus wrong, and how it can be improved. This could be at the level of a specific example (checking if an example is correct? high quality? challenging enough?), or potentially at the dataset level (are the samples diverse? do they improve a model if used as training data?). These learnings are fed back into the data creation process to improve the data in the next iteration, until a stopping criteria is met.

Overall Data Scientist Loop. The agent loops over the data creation and data analysis stages until it is satisfied with the quality of the data, and then generates a final training dataset or benchmark. This can include specific guardrails in the outer loop to prevent hacking. The agentic loop allows the model to build on top of its own learnings during these steps.

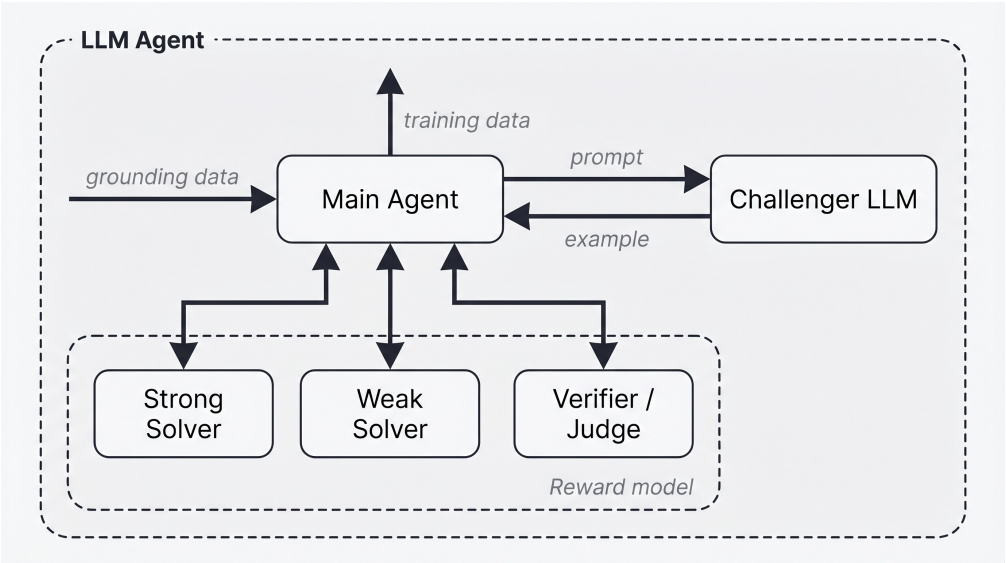

Figure 2 Weak-vs-strong **Agentic Self-Instruct method**. The main LLM agent directs four subagents: a challenger LLM generates examples; weak and strong solvers attempt it; a judge evaluates their outputs. The system aims to generate training data where the strong solver succeeds while the weak solver struggles. The main LLM analyzes data and updates the challenger prompt using the judge's feedback and repeats the cycle, yielding challenging examples for training the weak solver.

Meta-Optimization of the Data Scientist. The agent itself can also be optimized to be *better at being a data scientist*. One way to do this is to optimize the agent harness using autoresearch (Karpathy, 2026) or meta-harness (Lee et al., 2026) style optimization using the same inner loop criteria (creating better data) to guide the optimization of the outer loop (the agent optimization itself). This is depicted in the outer box of Figure 1.

2.1 A specific implementation: Agentic Self-Instruct

In our experiments we consider a specific, practical implementation of Autodata for creating high quality data which we call Agentic Self-Instruct, depicted in Figure 2. The main orchestrator agent has access to four LLM subagents:

(i) Challenger, which creates training examples given a detailed prompt from the main agent,

(ii) "Weak" solver that is expected to generally struggle to solve the created training data; and

(iii) "Strong" solver that is expected to generally succeed at the created training data,

(iv) Verifier/judge that given the example and a model solution, checks its quality, and passes its learnings back to the main agent.

The main agent proceeds to create an example (e.g., a given context/input, desired response or reference answer, and evaluation criteria depending on the task), by sending its initial prompt including grounding context to the challenger. It then checks the quality of the challenger's output by sending the input to the weak and strong solvers, and assigning a reward based on the verifier's judgments. The judge also plays the role of checking the quality of the example itself: the question, reference answer or generated rubric.

For verifiable tasks (using an LLM-based verifier), one approach is to require that majority vote over the strong solver is correct, while majority vote over the weak solver is wrong. For non-verifiable tasks, we require a gap in quality as measured by the judge, e.g. given rubrics generated by the challenger, such that the task is neither too easy not too hard for the weak solver, while the strong solver helps guarantee correctness. The agent analyzes the report from the verifier (that includes the solver outputs), and if this criteria is not fulfilled,

then it continues to modify the input prompt sent to the challenger given these new learnings, to try and make a new example until the criteria is met.

This process allows the agent to effectively learn how to create challenging and high quality examples specifically for training the "weak" solver. We note that the "weak" and "strong" solvers can actually be the same LLM, but in different modes, e.g. the strong version can be allowed to use increased inference time compute including scaffolding or aggregation (Zhao et al., 2025b), as well as having access to privileged information.

3 Experiments

3.1 Computer Science Research tasks

We consider the task of answering computer science (CS) research questions, using academic CS papers as source material. CS research questions are open-ended and, unlike verifiable tasks, require rubric-based evaluation. The challenger generates a context, a question, a reference answer, and a self-contained evaluation rubric consisting of weighted criteria that an LLM-judge uses to score any response without access to the reference answer.

Table 1 Quality statistics for generated CS research tasks. CoT Self-Instruct is standard prompted generation; Agentic Self-Instruct is the agentic loop accepted output. Both columns are graded by Kimi-K2.6 at generation time on the same 4B-weak / 397B-strong solver pair.

Metric	CoT Self-Instruct	Agentic Self-Instruct
Weak solver avg	0.677	0.458
Strong solver avg	0.696	0.772
Gap (strong − weak)	0.019	0.314
Agentic rounds	1.00	6.59
Question length (chars)	723	619
Rubric items	13.2	13.1

Pipeline overview. The main orchestrator agent calls the challenger to generate a context-QA pair with a corresponding rubric from a given paper. A quality verifier then checks for context leakage, rubric coverage, and question quality before final evaluation (and also as a final step at the end of the loop). The question and context are sent to both solvers (each invoked 3 times to reduce variance), and the judge scores their answers against the rubric on a per-criterion basis. If any acceptance criterion fails, the agent provides targeted feedback to the challenger: which previous questions were too easy (with weak solver scores), which failed on the strong solver (with gap information), and which were rejected by the quality verifier. The challenger then generates a new question from a different reasoning angle. This loop typically runs several rounds per paper before it either finds an accepted question or exhausts the step budget. We use Kimi-K2.6 as the main orchestrator agent and challenger, Qwen3.5-397B-A17B as the strong solver, and Qwen3.5-4B as the weak solver. Agent system prompts are provided in Appendix subsection C.1.

Criteria. A useful training example for the weak solver requires that the strong solver scores meaningfully higher than the weak solver on the rubrics. In preliminary experiments, however, most questions generated by prompted Kimi-K2.6 are too *easy* for the weak solver, leaving limited room for improvement: as shown in the "CoT Self-Instruct" column of Table 1, these questions have a weak solver average above 0.67 and a weak/strong gap of only 0.02. We therefore define the acceptance criterion of the agentic loop directly in terms of this gap: a candidate question is accepted only if the strong solver averages ≥ 0.65, the weak solver < 0.5, and the strong−weak gap ≥ 20 percentage points across the solver attempts. Given the strong nature of the weak solver, we save compute at each iteration by having the judge evaluate the strong solver only if the weak solver passes its corresponding success criterion.

Data Setup. We process over 10k CS papers from the S2ORC corpus (2022+) (Lo et al., 2020), producing 2.8k accepted examples with Agentic Self-Instruct. The quality verifier with Kimi-K2.6 at the end of the loop (that

Table 2 RL training results on CS research tasks. The autodata Agentic Self-Instruct method outperforms creating data with standard CoT Self-Instruct. We train Qwen3.5-4B with GRPO on 1.3k examples from each data source and evaluate on a 200-prompt held-out test set. Scores are rubric-based, graded by Kimi-K2.6. As shown in Figure 3, Agentic outperforms CoT data throughout training; we illustrate with step 200 in the table.

	CoT test		Agentic test	
Response model	mean@3	best@3	mean@3	best@3
Qwen3.5-4B (no additional RL)	0.630	0.758	0.366	0.484
Qwen3.5-4B RL on CoT Self-Instruct data	0.727	0.853	0.500	0.631
Qwen3.5-4B RL on Agentic Self-Instruct data	**0.774**	**0.894**	**0.632**	**0.768**

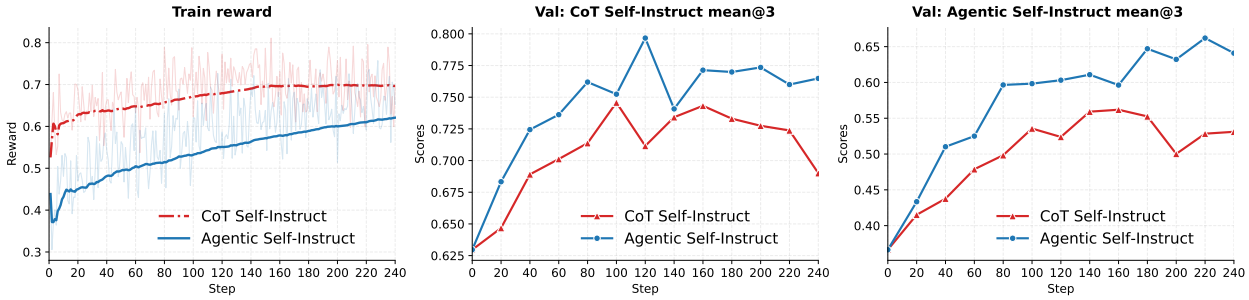

Figure 3 Train reward and held-out validation for the two CS RL runs (Qwen3.5-4B trained on Agentic vs. CoT Self-Instruct data). *Left*: per-step Kimi-K2.6 rubric reward, EMA-smoothed. *Middle / Right*: mean@3 on the CoT and Agentic held-out test sets (each is in-distribution for one arm, out-of-distribution for the other). Agentic-trained leads on both held-out sets throughout training, with the larger margin on the harder Agentic test.

removes questions with paper-specific reference leakage, short contexts, or malformed rubrics) further filters this set and we retain 1.3k high-quality accepted examples as the Agentic Self-Instruct data for RL training. For the CoT Self-Instruct baseline we also apply the same quality verifier and sample an equal amount of 1.3k filtered data for fair comparison.

3.1.1 Agentic Self-Instruct Loop Analysis

Running the data creation loop over our 10k-paper corpus, the agent needed substantial iterations to produce an accepted question: a mean of 6.59 rounds per accepted item (Table 1), with a long tail extending past 10 rounds on a fraction of papers. The failure modes we saw at each round were heavily one-sided: across 880 pre-acceptance rounds, 80% of failed rounds were rejected because the question was too easy and the weak solver scored too high, 13% because the strong solver could not reliably solve them either.

For the papers that did converge, the generated question was rarely the one that the agent created in the first attempt. For instance, the weak solver scores an average of 0.677 on the questions generated by the baseline CoT Self-Instruct method, while the questions generated using Agentic Self-Instruct see a 22-point drop given the same source material (papers). Inspecting the trajectories, we observe that the agent's initial attempt on a CS paper was usually a high-level summary question that often proves to be easy for a 4B solver model. However, subsequent rounds, guided by the judge's feedback, moved the questions toward specific algorithmic steps, ablation details, or numerical claims in the paper that required following the paper's actual argument. The aggregate effect of this search is visible in the corpus statistics in Table 1: the weak solver's score drops by 22 points (0.677 $\rightarrow$ 0.458) while the strong solver's score improves by 8 points (0.696 $\rightarrow$ 0.772), confirming that the accepted questions are harder in a way the strong solver's deeper reasoning specifically rewards. That is, the agentic data creation loop produces questions that specifically reward stronger model capabilities, rather than questions both models can answer.

Improvement works through exploration. Each round the agent generates a new question from a different reasoning angle, guided by feedback on which previous questions were too easy or failed to discriminate. The

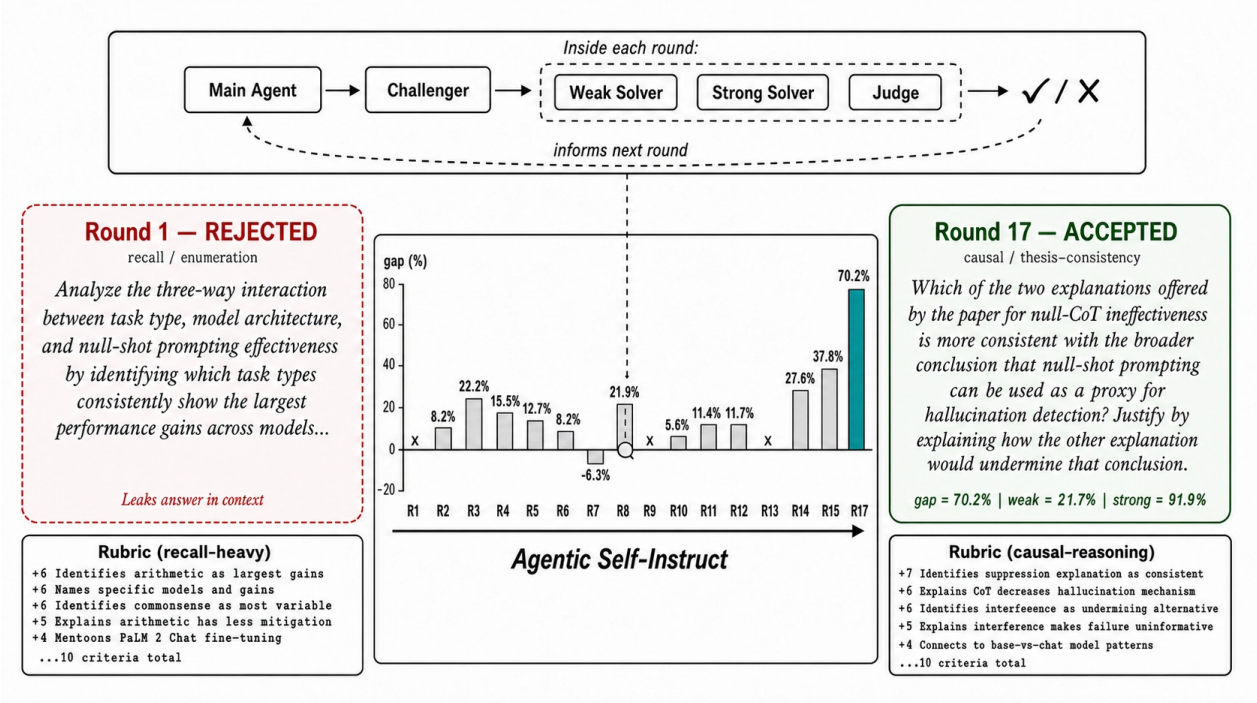

Figure 4 Autodata creation of CS research questions. Shown is the progression of Agentic Self-Instruct generating a training example with corresponding evaluation rubric for a given paper about Large Language Modeling.

accepted questions after the agentic loop also qualitatively test different reasoning types: specific technical mechanisms, multi-step derivations, and paper-specific design tradeoffs, compared to the broader, more generic questions produced without this loop. See Figure 4 for an example.

3.1.2 RL Training Results

We compare the performance of Qwen3.5-4B trained on 1.3k examples from CoT Self-Instruct versus Agentic Self-Instruct data, using Kimi-K2.6 as the reward model to score responses against the generated rubrics. From each dataset (generated using CoT Self-Instruct and Agentic Self-Instruct) we use 100 examples as the test set and train Qwen3.5-4B with GRPO (Shao et al., 2024) (batch size 16, learning rate 1e-6), evaluating each trained model on both held-out test sets.

On the easier CoT Self-Instruct test set (Table 2, left), training on CoT data lifts the base 4B model from 0.630 mean@3 to 0.727 and training on Agentic data lifts it further to 0.774. On the harder Agentic test set (right), the corresponding numbers are 0.366 (base) $\rightarrow$ 0.500 (CoT-trained) $\rightarrow$ 0.632 (Agentic-trained): the gap between the two methods is more than twice as large here as on the CoT test, and best@3 follows the same ordering. The training dynamics in Figure 3 are consistent with this: our Agentic method sits above the CoT Self-Instruct method on the per-step Kimi reward from the start and the spread widens through training, while on the validation panels our Agentic method matches or exceeds the CoT method at every checkpoint on both test sets, including the one where the CoT data was the natural in-distribution choice. The Agentic-trained model transfers in both directions (+0.05 to the easier CoT test, +0.13 to the harder Agentic test), the clear advantage suggests that the discriminative training data produced by the Agentic pipeline translates to stronger reasoning performance.

3.2 Legal Reasoning Tasks

We next investigate a second setting to test the method's generality, on legal reasoning tasks. In subsection 3.1 we experimented with Agentic Self-Instruct on open-ended CS research tasks, where the initial synthetic data was not challenging enough for the model to improve when using it for downstream RL training. This section

studies a second setting on improving an LLM's legal reasoning capabilities, which turns out to have different qualities. Here, we find that Agentic Self-Instruct has to contend with the opposite failure mode: we find standard prompting via CoT self-instruct produces questions that are *too hard* (and not *too easy*) in a way that hinders the RL reward signal.

The goal as before is to create high-quality data to improve the weak solver on the given task, in this case legal reasoning. We use court opinions and other public legal documents drawn from Pile of Law (Henderson et al., 2022) as source material, and evaluate on PRBench-Legal and the PRBench-Legal-Hard subset (Akyürek et al., 2025). As in the CS setting, we use Kimi-K2.6 as the main orchestrator agent, challenger and judge, Qwen3.5-397B-A17B as the strong solver, and Qwen3.5-4B as the weak solver. Quite differently from CS papers, our initial study showed that CoT Self-Instruct generated questions and rubrics are too hard for the weak solver. As shown in Table 3, the weak solver averages only 0.159, with many of the attempts scoring 0, which hinders learning under GRPO. If we were to apply the hard-threshold acceptance criteria from the CS setting here, the majority of the CoT data points would have been accepted. Instead, we ask: can we directly verify both the quality and the GRPO-suitability of a data point?

Table 3 Quality statistics for generated legal reasoning tasks. CoT Self-Instruct is standard prompted generation; Agentic Self-Instruct is the output of the agentic loop. Both columns are graded by Kimi-K2.6 at generation time on the same 4B-weak / 397B-strong solver pair.

Metric	CoT Self-Instruct	Agentic Self-Instruct
Weak solver avg	0.159	0.283
Strong solver avg	0.717	0.698
Gap (strong − weak)	0.558	0.415
Agentic rounds	1.00	4.98
Question length (chars)	1,569	900
Rubric items	18.6	17.3
Weak rollout std	7.93	12.63

Pipeline overview. The main difference from the hard-coded acceptance criteria we adopted in the CS papers task is that here we instead adopt a more flexible *loop judge* to decide if a round's generation is accepted. Specifically, each legal document is first passed through a dedicated *extractor* agent that produces a structured extract (topic keywords, salient facts, holdings). The *challenger* agent then generates one realistic legal question plus a weighted grading rubric and a declared target-capability set from the extract. Each candidate is rolled out by the weak solver 5 times and the strong solver 3 times.

The judge then reads the per-rollout solver patterns, the weak/strong gap, and the rubric and returns a structured verdict (`weak_pattern`, `strong_pattern`, `gap_interpretation`, `rubric_concerns`, `grpo_suitability`) plus an `accept`/`improve` decision. For the `improve` decision case, it hands the challenger a concrete `suggestion_for_challenger` (e.g. "the weak-rollouts are all reciting the same boilerplate; push the question toward step-wise application of the holding rather than recall"), and the loop re-runs; on `accept` the example is considered good quality and the loop ends. Unlike the hard-coded CS quality decision, the judge we use for legal tasks has no fixed acceptance thresholds: it decides per round given analysis of the results, including the weak-rollout *variance*, the gap, and PRBench baseline solver statistics. A good legal training example is defined by overall data quality and GRPO-suitability, not only by a numeric gap target.

Agent system prompts are provided in Appendix subsection C.2.

Data Setup. We processed 7.8k source documents, where 5.7k of these produced usable CoT Self-Instruct examples and 2.8k reached an `accept` verdict after the agentic loop. For the controlled head-to-head RL comparison in subsubsection 3.2.2, the *Agentic* experiment uses all 2.8k accepted data points and the *CoT* Self-Instruct set uses 2.8k examples randomly drawn from the 5.7k CoT pool.

3.2.1 Agentic Self-Instruct Loop Analysis

The Agentic Self-Instruct loop reshapes the weak-rollout distribution. Table 3 shows the before/after analysis: the weak/strong gap on the same source documents actually narrows from 55.8 to 41.5 points after the agentic

Table 4 RL training results evaluated on PRBench. The autodata Agentic Self-Instruct method outperforms creating data with standard CoT Self-Instruct. We train Qwen3.5-4B with GRPO on 2.8k legal QA pairs from each data source and evaluate on the PRBench-Legal (500 prompts) and PRBench-Legal-Hard (250-prompt subset) test sets. Scores are clipped per-prompt PRBench scores, graded by both Kimi-K2.6 and GPT-5. Both graders agree: our 4B trained on Agentic data outperforms the model trained on the standard CoT Self-Instruct data as well as the much larger 397B baseline on both splits.

	GPT-5 Grader		**Kimi-K2.6 Grader**	
Response Model	Legal	Legal-Hard	Legal	Legal-Hard
Qwen3.5-4B (no additional RL)	0.280	0.167	0.245	0.145
Qwen3.5-397B (no additional RL)	0.404	0.277	0.358	0.226
Qwen3.5-4B RL on CoT Self-Instruct data	0.377	0.253	0.343	0.233
Qwen3.5-4B RL on Agentic Self-Instruct data	**0.441**	**0.315**	**0.393**	**0.266**

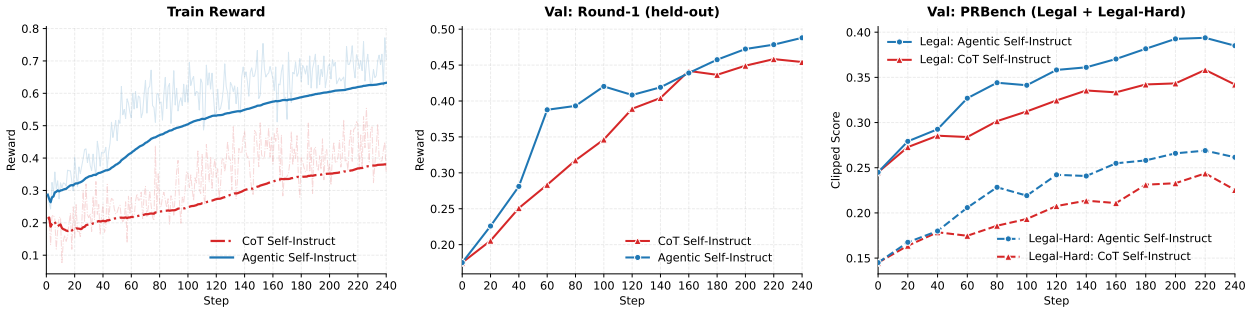

Figure 5 RL training dynamics on legal reasoning. We train Qwen3.5-4B with GRPO on 2.8k legal Question-Rubric pairs from each data source (Agentic Self-Instruct, CoT Self-Instruct) and evaluate every 20 steps on a 100-prompt held-out CoT set (*middle*) and the PRBench Legal / Legal-Hard splits (*right*). All rewards and scores graded by Kimi-K2.6. Agentic stays ahead of CoT on every metric throughout training.

loop, unlike the CS setting. The notable change lies in that per-prompt weak-rollout standard deviation rises from 7.93 to 12.63. CoT questions concentrate weak scores near zero (mean 15.9%, median 10.7%); many prompts have four or five out of five weak-rollouts scoring zero, which leaves the per-group GRPO advantage near zero and provides little learning signal. The agentic loop pushes the weak mean up to 28.3% while leaving strong roughly unchanged (71.7% $\rightarrow$ 69.8%); the same gap is now spread over a usable variance range. The loop makes the questions more *learnable* by reshaping the per-prompt reward signal. As a byproduct, the loop judge's textual feedback also pushes the challenger toward shorter, application-style questions (mean 900 vs 1.6k characters) without changing rubric density, incidentally aligning the format with the relatively short PRBench-Legal prompts.

At each round the loop judge provides a categorical `grpo_suitability` verdict (`high`/`medium`/`low`) based on the rollout patterns. On the CoT Self-Instruct pool the distribution is 4.8% high / 41% medium / 45% low; on the Agentic pool it is **52% high / 43% medium / 2% low**. The median accepted question goes through 4 agentic rounds (mean 4.98, max 19), only $\sim$2% use a single round.

3.2.2 RL Training Results

We RL-train Qwen3.5-4B with GRPO on the setups: 2.8k Agentic Self-Instruct prompts (*Agentic*) versus 2.8k standard CoT Self-Instruct prompts (*CoT*), with n=8 rollouts per prompt, and a Kimi-K2.6 rubric judge as the train-time reward. We evaluate on PRBench-Legal (500 prompts, 250 of them tagged `legal_hard`) under the PRBench-official judge setting (e.g, the judge evaluates each rubric separately), and additionally re-grade the same rollouts with GPT-5 as an independent, stronger grader to confirm the comparison is not Kimi-grader biased.

On the 500-prompt PRBench-Legal split (Table 4), Qwen3.5-4B RL'd on Agentic data scores 0.441 (GPT-5

as judge) and 0.393 (Kimi as judge), outperforming the same-architecture CoT-trained model (0.377 / 0.343), and even outperforming the much larger strong Qwen3.5-397B-A17B baseline without additional RL (0.404 / 0.358). The same ordering holds on PRBench-Legal-Hard. The +0.05–0.06 advantage of Agentic over CoT is obtained on the same 2.8k-prompt budget, same challenger, same source corpus: the only difference between the two setups is the agentic loop in the training data creation. The training curves in Figure 5 show the Agentic method leading on train reward, on the held-out CoT validation set, and on PRBench-Legal at every checkpoint we evaluated.

"More Challenging" vs "Just Right"

The two tasks we have so far experimented with in subsection 3.1 and subsection 3.2 apply the *same* Agentic Self-Instruct loop to opposite failure modes of standard CoT-Self Instruct prompt-based generation: in CS, CoT questions are too easy for the weak solver (gap 0.02, raising the concern that the questions are not challenging enough), while in Legal they are too hard, with many rollouts scored at 0 (gap 0.56, but providing too harsh a learning signal). After applying the Autodata agentic loop the gap moves in opposite directions (widening in CS, narrowing in Legal), yet the downstream RL outcome is the same: the model trained on autodata-generated data outperforms the model trained on CoT data on every held-out test, and on PRBench-Legal a 4B model outperforms a much larger baseline. The key is not to make the question more challenging, but to make them *just right* for the model to hill-climb on; the Agentic Self-Instruct loop is what lets us achieve this.

3.3 Scientific reasoning

Next, we consider the construction of challenging problems that require reasoning over mathematical objects in the same categories and domains as the existing Principia collection (Aggarwal et al., 2026). The Principia collection was designed using a CoT Self-Instruct-based method (prompted, multi-step LLM workflow) covering a wide range of curricula from the MSC2020 and PHYS catalogs. Principia bench, on the other hand, consists of human-labeled subsets of existing math and physics benchmarks where the problems were filtered to contain those involving mathematical objects in the answer.

Pipeline overview In this experiment, we use a weak solver of Qwen3.5-4B, and a strong solver of Qwen3.5-397B-A17B. The weak solver is in fact a very capable reasoning model that can solve many problems from the Principia collection. Thus it is a good candidate model for our pipeline where we seek to generate problems that are more challenging for the weak solver. The main orchestrator agent and challenger are Kimi K2.6. Agent system prompts are provided in Appendix subsection C.3. A detailed breakdown of question types in the constructed agentic data is provided in Appendix B.

3.3.1 RL training results

This setup allows us to compare three data sources for down-stream RL training: (i) **CoT Self-Instruct**: training directly on the problems from the Principia collection, which are also used as grounding context in Agentic Self-Instruct, (ii) **Agentic**: training on data generated by Agentic Self-Instruct, and (iii) **Combined**: training on both data sources together, which doubles the training set size. Each individual data source consists of 9k training examples and 1k held-out evaluation examples, while the Combined setting uses 18k training examples. Similarly to previous experiments, we train Qwen3.5-4B model using Kimi K2.6 as a judge to compare model's generated answer against the reference answer and assign binary reward based on the comparison. We use GRPO with group size 8 and batch size 64 for training.

We evaluate models both in and out of their training distributions using a combined validation set consisting of held-out examples from both the agentic-generated and CoT data distributions, as well as the out-of-distribution Principia benchmark. Results are shown in Table 5 and Table 6.

On the combined validation set (Table 5), training on Agentic Self-Instruct data yields the largest overall improvement (+3.20% avg@8), outperforming both direct training on CoT Self-Instruct data (+2.42%) and the combined dataset (+2.70%). A key finding is that Agentic Self-Instruct data improves performance even on the CoT validaton subset (+3.05% vs. +1.86% for CoT Self-Instruct), despite not being explicitly optimized for that distribution. This demonstrates that *training on harder problems transfers to easier ones*:

Table 5 RL training results evaluated on scientific reasoning tasks. Agentic Self-Instruct data outperforms CoT Self-Instruct or even Combined data (2× training size). Deltas (Δ) are relative to the starting Qwen3.5-4B model.

	Base	CoT Self-Instruct		Agentic Self-Instruct		Combined (2× data)	
Eval Subset	Qwen3.5-4B	avg@8	Δ	avg@8	Δ	avg@8	Δ
Overall	68.66%	71.08%	+2.42	**71.86%**	**+3.20**	71.36%	+2.70
Agentic subset	52.39%	56.33%	+3.94	**56.79%**	**+4.40**	55.88%	+3.49
CoT Self-Instruct subset	77.17%	79.03%	+1.86	**80.22%**	**+3.05**	79.66%	+2.49
		pass@8	Δ	pass@8	Δ	pass@8	Δ
Overall	87.73%	88.58%	+0.85	**88.91%**	**+1.18**	88.75%	+1.02
Agentic subset	81.13%	81.74%	+0.61	81.86%	+0.73	**82.11%**	**+0.98**
CoT Self-Instruct subset	92.52%	92.77%	+0.25	**94.36%**	**+1.84**	93.50%	+0.98

Table 6 Out-of-distribution Principia benchmark results comparing training data sources. Agentic Self-Instruct data yields the largest overall improvement despite using half the data of Combined.

		Base	CoT Self-Instruct		Agentic Self-Instruct		Combined (2× data)	
Category	**Items**	Qwen3.5-4B	avg@8	Δ	avg@8	Δ	avg@8	Δ
Overall	2113	50.43%	51.10%	+0.67	**51.47%**	**+1.04**	51.17%	+0.74
ARB	47	81.91%	**83.78%**	**+1.87**	80.32%	−1.59	83.24%	+1.33
Physics	110	66.22%	65.91%	−0.31	**67.05%**	**+0.83**	66.25%	+0.03
RealMath	632	33.68%	35.25%	+1.57	**35.43%**	**+1.75**	34.97%	+1.29
SuperGPQA	1324	56.00%	56.28%	+0.28	**56.82%**	**+0.82**	56.50%	+0.50
			pass@8	Δ	pass@8	Δ	pass@8	Δ
Overall	2113	59.54%	**60.06%**	**+0.52**	59.91%	+0.37	59.91%	+0.37
ARB	47	91.49%	91.49%	+0.00	91.49%	+0.00	**93.62%**	**+2.13**
Physics	110	76.36%	77.27%	+0.91	**79.09%**	**+2.73**	72.73%	−3.63
RealMath	632	41.93%	43.67%	+1.74	43.67%	+1.74	**44.30%**	**+2.37**
SuperGPQA	1324	65.41%	65.33%	−0.08	64.95%	−0.46	65.11%	−0.30

the challenging examples produced by our iterative agentic process teach reasoning skills that generalize beyond the specific difficulty level they target.

On the out-of-distribution Principia benchmark (Table 6), Agentic Self-Instruct again achieves the best overall avg@8 improvement (+1.04%), with consistent gains across most categories, particularly on RealMath (+1.75%) and SuperGPQA (+0.82%). This transfer effect further confirms that the harder problems generated by Agentic Self-Instruct build more robust reasoning capabilities.

The pass@8 results reveal a more nuanced picture with trade-offs across methods. The Combined data shows advantages on pass@8 in several categories: ARB (+2.13% vs. +0.00% for both Agentic and Grounding) and RealMath (+2.37% vs. +1.74% for Agentic). This suggests that while Agentic Self-Instruct improves *average* performance by teaching the model to solve challenging problems more reliably, the Combined data's greater diversity (and size) may help the model *occasionally* solve a broader range of problems (reflected in higher pass@8). One possible explanation is that Qwen3.5-4B may be approaching its capacity limit for this task distribution, and larger models might better exploit the combined data to achieve gains on both metrics simultaneously.

These results highlight the value of *data quality and difficulty*. The iterative process in Agentic Self-Instruct produces challenging examples that provide a more efficient learning signal—training on these harder problems not only improves performance on difficult tasks but also transfers to easier ones. This supports the hypothesis that investing inference-time compute in generating higher-quality, more challenging synthetic data can be more effective than simply scaling dataset size. Additionally, we find that training substantially reduces reasoning truncation rates (from 23.75% to 4.09% for Agentic Self-Instruct with a 65,536 token budget), with approximately half of accuracy improvements attributable to more token-efficient reasoning (see Appendix A).

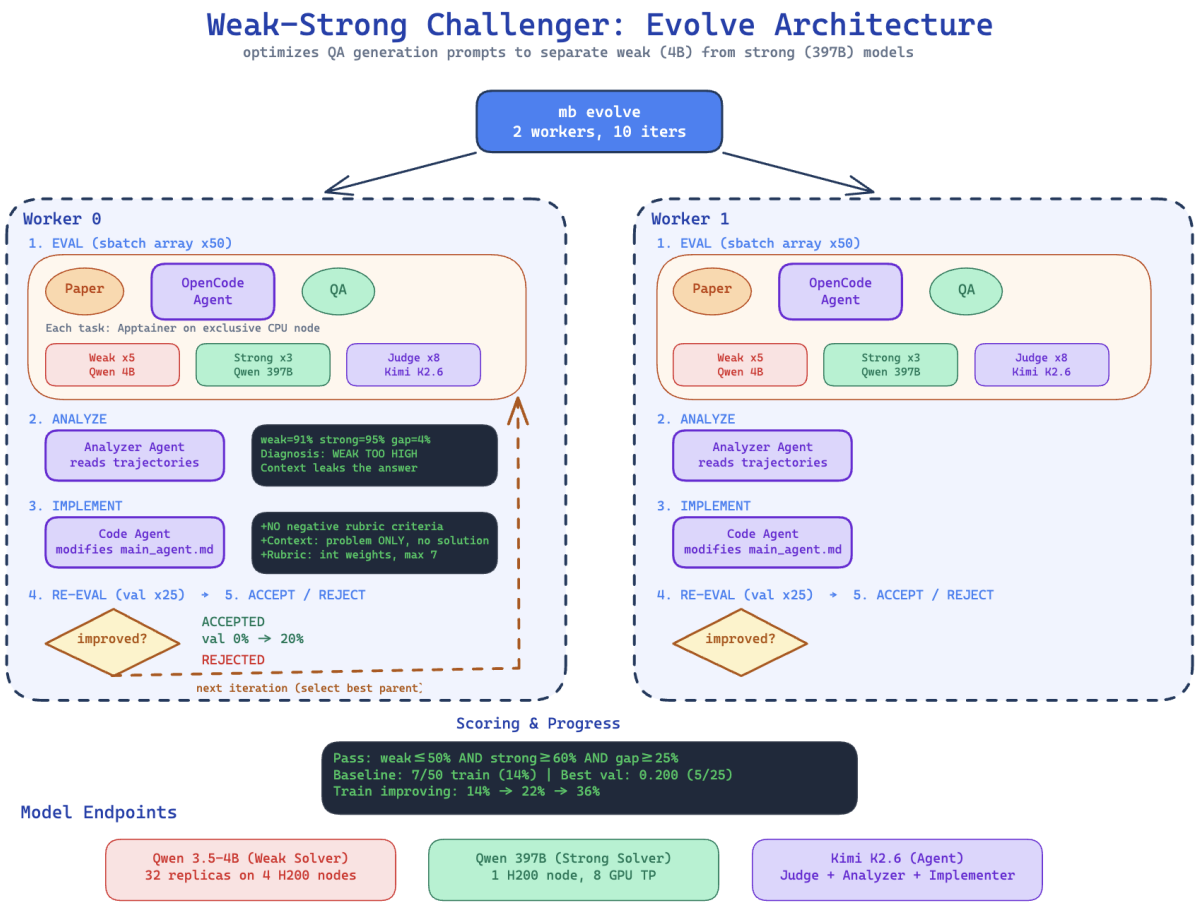

Figure 6 Meta-optimization of the data scientist agent. An outer optimization loop evaluates the agent's prompt on training examples of CS paper task, analyzes failure trajectories to identify systematic weaknesses (e.g., context leakage), implements prompt modifications via a code-editing agent, and re-evaluates on held-out validation papers. Changes are accepted only if they improve the weak-strong separation rate. This process improved validation pass rate from 12.8% to 42.4% over 126 accepted iterations out of 233 total.

4 Meta Optimization of the Data Scientist

So far, we have implemented the autodata agent using a fixed Agentic Self-Instruct framework, with provided prompts which define how the agent's strategy. However, it is also possible to (meta-)learn the overall agent as well. In this section we thus apply meta-optimization to the data scientist agent itself, using the same evaluation criteria from the inner loop to guide optimization of the outer loop—the agent's prompt and strategy. Concretely, we use a evolution optimization framework that treats the agent's scaffold as code to be iteratively improved.

Method. The meta-optimizer maintains a population of candidate prompts, each defined by a code diff relative to the baseline repository. Each iteration proceeds as follows: (1) **Select** a parent from the population via Boltzmann sampling, where candidate c is chosen with probability proportional to $\exp(score_c/T)$ with temperature T=0.1, strongly favoring high-scoring candidates while maintaining exploration; (2) **Evaluate** the parent's prompt on a minibatch of training papers, collecting agent trajectories and weak/strong solver scores; (3) **Analyze** the trajectories with an LLM agent that reads the full solver exchanges and writes a root-cause analysis of systematic failure patterns; (4) **Implement** prompt modifications via a code-editing agent that reads the analysis, iteration history, and current prompt, then produces an improved diff; (5) **Re-evaluate** both parent and mutant on held-out validation papers; (6) **Accept or reject** the mutant—it is added to the

population only if its validation score strictly exceeds its parent's; (7) **Summarize** the outcome into a history log that subsequent analyzers can read. Multiple iterations run concurrently with independent parent selections. Because single-evaluation scores are noisy (solvers run at temperature 1.0), accepted candidates accumulate additional evaluations whenever they are subsequently sampled as parents, and we report the averaged score across all re-evaluations.

Setup. We meta-optimize the CS research paper task from subsection 3.1. The meta-optimizer uses Kimi-K2.6 as both the analyzer (which reads evaluation trajectories to diagnose failure patterns) and the implementer (which modifies the agent's prompts). The inner-loop agent being optimized also uses Kimi-K2.6 in a multi-agent configuration with separate challenger, main agent, and quality verifier prompts, following subsection 3.1. We use 50 training papers and 25 validation papers. A generated QA pair is considered successful if it satisfy all of the criterion's: weak solver (Qwen3.5-4B) scores $\leq$65%, the best weak solver attempt score $\leq$ 75%, the strong solver (Qwen3.5-397B-A17B) scores $\geq$60% and $\leq$95%, and the gap between the strong and weak solver $\geq$20 percentage points, as judged by rubric-based evaluation.

Results. Starting from a baseline prompt that achieves ~62.1% validation QA pass rate on the 100 validation samples, the meta-optimizer progressively discovers prompt improvements across 233 iterations. We report the baseline and the final evolved harness result under 6h per session timeout. Results are shown in Table 7.

Table 7 Meta-optimization of the data scientist agent on the CS research paper task. The optimizer iteratively improves the agent's prompt, with each accepted iteration building on the previous best. Validation pass rate (re-evaluated) measures the fraction of generated QA pairs that successfully separate weak and strong solvers, averaged over multiple re-evaluations to reduce noise.

Meta Iteration	Num Val Sample	Val Pass Rate 6h
Baseline	100	62.1%
Iter 124	100	79.6%

The meta-optimizer identified several systematic failure modes through trajectory analysis—examining what the weak solver actually said in its responses and identifying that generic answers and rubric format errors were the dominant causes of poor separation. The optimizer addressed these through the following prompt modifications, discovered automatically over the course of 233 iterations:

- **Paper-specific insight enforcement**: The optimizer added instructions requiring that questions test knowledge *specific to the paper*, not generic ML/CS knowledge. A self-test was introduced: "If a solver could answer correctly without reading this specific paper, the question is too easy." This directly addressed weak solvers achieving high scores by producing plausible-sounding generic responses.
- **Context leak prevention**: Strict rules were added requiring the context to describe only the problem domain and setup, never the paper's proposed solution. A self-test was introduced: "Could someone answer the question by rephrasing sentences from the context? If yes, rewrite."
- **Positive-only rubric with weight capping**: The optimizer *eliminated* negative-weight rubric criteria, finding that they historically misfired and destroyed strong model scores without improving discrimination. Instead, all criteria use positive integer weights capped at 7, preventing any single criterion from dominating the score. This was a counter-intuitive discovery—penalizing errors seemed helpful in theory but hurt in practice.
- **Structured rubric format**: The optimizer enforced a strict JSON format for rubric criteria with integer weights, eliminating parsing errors (e.g., string weights like "+8" instead of the integer 8) that had caused evaluation failures in earlier iterations.

The progression from 62.1% to 79.6% validated pass rate demonstrates that meta-optimizing the data scientist agent's instructions can substantially improve data quality without manual prompt engineering, though the absolute numbers also highlight the difficulty of reliably generating questions that separate models of different capability levels.

5 Related Work

Synthetic instruction and alignment data. Synthetic data has become a central ingredient in post-training language models, especially as human-written supervision becomes expensive, scarce, or insufficiently challenging. Self-Instruct (Wang et al., 2023) introduced a simple and influential recipe in which a language model bootstraps new instruction-following examples from a small seed set. Subsequent work scaled and diversified this idea in several directions: instruction distillation from stronger teachers (Mukherjee et al., 2023), large-scale synthetic conversations (Ding et al., 2023), AI-generated preference and feedback data (Cui et al., 2023), and automatic instruction evolution to increase task complexity and diversity (Xu et al., 2024). More recent work such as Magpie (Xu et al., 2025) further shows that aligned LLMs can be used to synthesize large-scale alignment data from minimal prompting. These methods establish that LLMs can be powerful data generators, but typically treat generation as a mostly fixed prompting or filtering pipeline. Autodata instead treats data creation as an iterative data-science process: the agent generates examples, evaluates their usefulness, analyzes failures, and revises its data-generation recipe.

Grounded, verifiable and reasoning-based synthetic data. A second line of work emphasizes that synthetic data quality depends strongly on grounding, domain specificity such as specific verifiable tasks, and the form of reasoning traces. In code and mathematical reasoning, synthetic "textbook" data and exercises played a key role in training small but strong models (Li et al., 2023). MetaMath (Yu et al., 2024), MAmmoTH (Yue et al., 2024), and OpenMathInstruct (Toshniwal et al., 2025) show that synthetic or semi-synthetic mathematical reasoning data can substantially improve downstream problem solving. Grounded methods such as Source2Synth (Lupidi et al., 2024) and NaturalReasoning (Yuan et al., 2025) generate examples from real documents or tables and curate them for answerability, while CoT-Self-Instruct (Yu et al., 2025) uses chain-of-thought planning and filtering to improve synthetic data for both verifiable reasoning and open-ended instruction following. Autodata builds on these grounded and reasoning-aware data generation methods, but adds an explicit agentic loop that uses solver behavior and evaluator feedback to adapt the generated data to the target model.

Agentic data generation and automated data-science systems. Several recent systems move from single-prompt generation toward agentic data or data-science workflows. AgentInstruct (Mitra et al., 2024) is especially close in spirit: it uses agentic flows to generate large-scale, diverse synthetic post-training data. Our work instead treats data creation as an iterative data-science loop, where an agent generates examples, evaluates their learning utility, analyzes failures, and revises its recipe. In particular, Agentic Self-Instruct uses weak–strong solver behavior and judge feedback to tune data difficulty, and can further meta-optimize the data scientist agent itself. There is also an existing work that uses the naming "AutoData" (Ma et al., 2026) which studies a multi-agent system for open web data collection, using specialized agents to collect datasets from natural-language instructions, which could be seen as a related and special case of our framework as well. In parallel, LLM-based data-science agents such as DS-Agent (Guo et al., 2024) and Data Interpreter (Hong et al., 2025) automate parts of the data-science workflow, including planning, coding, model training, debugging, and analysis. These works demonstrate that LLM agents can perform complex data-oriented workflows. Our work differs in its objective: the agent is not primarily collecting web data or solving data-science competitions, but acting as a data scientist whose output is training or evaluation data for another model. The core optimization target is therefore the learning value of the generated data, as measured by task-specific evaluators and solver behavior.

Self-improvement, self-play, and challenger–solver data. Autodata is also related to self-improvement and self-play methods in which models generate data, rewards, or tasks for their own training. STaR (Zelikman et al., 2022) bootstraps reasoning traces by iteratively generating and training on successful rationales. Self-Rewarding Language Models (Yuan et al., 2024) use the model itself as an LLM-as-a-judge reward source during iterative preference training. More adversarial or curriculum-oriented approaches train models with self-generated tasks: Self-Challenging Language Model Agents (Zhou et al., 2025) generate tool-use tasks together with verification functions, Absolute Zero (Zhao et al., 2025a) proposes and solves its own verifiable reasoning tasks without external data, and SPICE (Liu et al., 2025) uses a challenger–reasoner setup grounded in corpora. Our weak–strong Agentic Self-Instruct instantiation shares the idea of a challenger creating tasks

for a solver, but places this inside a broader data-scientist loop: the agent analyzes solver failures, judges example quality, adjusts difficulty, and optimizes for examples that are useful for learning rather than merely difficult.

LLM judges, filtering, and data selection. Much synthetic data work relies on filtering, judging, or selecting examples after generation. Self-Instruct uses heuristic filtering for invalid or near-duplicate examples (Wang et al., 2023); WizardLM and related evolution methods increase complexity but still require data quality control (Xu et al., 2024); UltraFeedback scales AI feedback to support preference learning (Cui et al., 2023); and CoT-Self-Instruct uses answer-consistency or reward-model-based filtering to select high-quality examples (Yu et al., 2025). Autodata uses such evaluation signals more actively. Instead of filtering a static pool, the judge's feedback is part of the generation loop itself. In our experiments, this distinction matters: in CS research tasks the agent makes examples harder and more discriminative, while in legal reasoning the agent makes examples less degenerate and more suitable for GRPO by avoiding all-zero weak rollouts. Thus, the goal is not simply high quality or high difficulty, but data that provides an effective learning signal for the target model.

Autoresearch and optimization of agent scaffolds. Finally, Autodata connects to work on automated research and optimization of prompts, scaffolds, and agent harnesses. Prompt optimization methods such as Prompt-breeder (Fernando et al., 2023), Self-Refine (Madaan et al., 2023), LLMs as Optimizers (Yang et al., 2024) and more recently GEPA (Agrawal et al., 2025) show that LLM systems can improve prompts, solutions, or policies through iterative feedback. Further recent autoresearch systems aim to automate larger parts of the scientific loop: The AI Scientist (Lu et al., 2024) performs ideation, experiment implementation, result analysis, paper writing, and simulated review, while autoresearch (Karpathy, 2026) explores agents that modify training code and recipes. Meta-Harness (Lee et al., 2026) treats the harness around an LLM system as an object of end-to-end optimization. Our meta-optimization experiment applies this perspective to data creation itself: the outer loop improves the data scientist agent's prompt and strategy using the same data quality criteria that guide the inner data-generation loop.

Positioning. Prior work has shown that LLMs can synthesize instructions, conversations, feedback, reasoning traces, domain-specific datasets, and even self-play tasks. Autodata unifies these ideas under an explicit agentic data-science formulation. Its key distinction is that data generation, evaluation, failure analysis, recipe revision, and meta-optimization are all part of the loop. This provides a general mechanism for converting stronger inference-time models and larger agentic compute budgets into higher-quality training and evaluation data.

6 Conclusion and Discussion

We introduced Autodata, a general framework in which an autonomous agent plays the role of a data scientist—generating synthetic data, evaluating it with task-specific signals, and improving its data-generation recipe based on those results. We instantiated this idea with Agentic Self-Instruct, which explicitly optimizes for examples that separate weak and strong solvers, and demonstrated consistent quality gains across computer science research tasks, legal reasoning tasks, and reasoning with mathematical objects. Finally, we showed that the data scientist agent itself can be meta-optimized, yielding substantial additional improvements without manual prompt engineering. We believe that the experiments we report in this paper are just the tip of the iceberg, and further exploration and optimization of this approach will bring further gains. We list future directions below.

More tasks, models and baselines. Future continued work should explore the use of this method across more diverse tasks and models. We envision an ideal system being a general autodata agent that can be used for any kind of data (mathematics, code, general instruction following tasks, safety, and so on) from verifiable to non-verifiable, single-turn to multi-turn to agentic tasks.

Hacking & limitations. We encountered instances of the agents trying to avoid doing the work correctly or trying to "cheat" the goal, e.g. by changing the prompt to the weak solver telling it to be weak, which we have partially addressed by simply enforcing more constraints on the agentic pipeline, but have plans of investigating stronger safeguards which would allow the agent to have more freedom to act and use tools than just in the rigid iterative loop defined here. Similarly, we wish to make sure that data is both challenging and meaningful, for example in the computer science task we found some generated questions and rubrics are overly tied to specific experimental numbers from the paper rather than testing generalizable reasoning.

Full dataset analysis iteration. Our initial experiments create quality data at the example level. As detailed in the general description of Autodata in section 2, we would like to expand this to dataset-level analysis in order to improve quality, for example diversity statistics and overall improvements with respect to how it interacts with existing datasets. An intermediate step rather than a full dataset analysis is iterative batched analysis, i.e. generating N examples, and then deriving learnings from the current batch in order to generate the next batch.

From Self-Improvement to Co-improvement. Our, and others, work on self-play (Zhou et al., 2025; Yuan et al., 2024; Zhao et al., 2025a; Liu et al., 2025) also involves making a "challenger" which generates training examples for a solver, which can be optimized together with rewards and weight updates, rather than in the agentic way described above. However, a full self-improving loop could consider our agentic self-instruction system as the challenger, and train it both in learnt skills and its weights – at the same time as training the solver. In this work we have explored an autoresearch-like method (Karpathy, 2026) to meta-train our agent, but there is much more to explore in this direction. Finally, removing humans completely from the loop is unlikely to be desirable in current full model training pipelines, especially when data creation is so important for model capabilities and safe behavior. Incorporating human feedback and ability to do "co-research" with the agent is likely a better path, called co-improvement (Weston and Foerster, 2025), which is a main direction of our future research.

References

Pranjal Aggarwal, Marjan Ghazvininejad, Seungone Kim, Ilia Kulikov, Jack Lanchantin, Xian Li, Tianjian Li, Bo Liu, Graham Neubig, Anaelia Ovalle, et al. Reasoning over mathematical objects: on-policy reward modeling and test time aggregation. *arXiv preprint arXiv:2603.18886*, 2026.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Afra Feyza Akyürek, Advait Gosai, Chen Bo Calvin Zhang, Vipul Gupta, Jaehwan Jeong, Anisha Gunjal, Tahseen Rabbani, Maria Mazzone, David Randolph, Mohammad Mahmoudi Meymand, et al. Prbench: Large-scale expert rubrics for evaluating high-stakes professional reasoning. *arXiv preprint arXiv:2511.11562*, 2025.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Qiang Yue, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with high-quality feedback. 2023.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051, 2023.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Automated data science by empowering large language models with case-based reasoning. *arXiv preprint arXiv:2402.17453*, 2024.

Peter Henderson, Mark Krass, Lucia Zheng, Neel Guha, Christopher D Manning, Dan Jurafsky, and Daniel Ho. Pile of law: Learning responsible data filtering from the law and a 256gb open-source legal dataset. *Advances in Neural Information Processing Systems*, 35:29217–29234, 2022.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, et al. Data interpreter: An llm agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19796–19821, 2025.

Andrej Karpathy. autoresearch: Ai agents running research on single-gpu nanochat training automatically. https://github.com/karpathy/autoresearch, 2026. GitHub repository.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023.

Bo Liu, Chuanyang Jin, Seungone Kim, Weizhe Yuan, Wenting Zhao, Ilia Kulikov, Xian Li, Sainbayar Sukhbaatar, Jack Lanchantin, and Jason Weston. Spice: Self-play in corpus environments improves reasoning. *arXiv preprint arXiv:2510.24684*, 2025.

Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel S Weld. S2orc: The semantic scholar open research corpus. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4969–4983, 2020.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Alisia Lupidi, Carlos Gemmell, Nicola Cancedda, Jane Dwivedi-Yu, Jason Weston, Jakob Foerster, Roberta Raileanu, and Maria Lomeli. Source2synth: Synthetic data generation and curation grounded in real data sources. *arXiv preprint arXiv:2409.08239*, 2024.

Tianyi Ma, Yiyue Qian, Zheyuan Zhang, Zehong Wang, Xiaoye Qian, Feifan Bai, Yifan Ding, Xuwei Luo, Shinan Zhang, Keerthiram Murugesan, et al. Autodata: A multi-agent system for open web data collection. *Advances in Neural Information Processing Systems*, 38:173416–173448, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, et al. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*, 2024.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*, 2023.

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. *arXiv preprint arXiv:2407.21009*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. In *International Conference on Learning Representations*, volume 2025, pages 19243–19275, 2025.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 13484–13508, 2023.

Jason Weston and Jakob Foerster. Ai & human co-improvement for safer co-superintelligence. *arXiv preprint arXiv:2512.05356*, 2025.

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 *International Conference on Learning Representations*, volume 2024, pages 30745–30766, 2024.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. In *International Conference on Learning Representations*, volume 2025, pages 76346–76382, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pages 12028–12068, 2024.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations*, volume 2024, pages 45040–45061, 2024.

Ping Yu, Jack Lanchantin, Tianlu Wang, Weizhe Yuan, Olga Golovneva, Ilia Kulikov, Sainbayar Sukhbaatar, Jason Weston, and Jing Xu. Cot-self-instruct: Building high-quality synthetic prompts for reasoning and non-reasoning tasks. *arXiv preprint arXiv:2507.23751*, 2025.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*, 2024.

Weizhe Yuan, Jane Yu, Song Jiang, Karthik Padthe, Yang Li, Ilia Kulikov, Kyunghyun Cho, Dong Wang, Yuandong Tian, Jason E Weston, et al. Naturalreasoning: Reasoning in the wild with 2.8 m challenging questions. *arXiv preprint arXiv:2502.13124*, 2025.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. In *International Conference on Learning Representations*, volume 2024, pages 40320–40341, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025a.

Wenting Zhao, Pranjal Aggarwal, Swarnadeep Saha, Asli Celikyilmaz, Jason Weston, and Ilia Kulikov. The majority is not always right: Rl training for solution aggregation. *arXiv preprint arXiv:2509.06870*, 2025b.

Yifei Zhou, Sergey Levine, Jason Weston, Xian Li, and Sainbayar Sukhbaatar. Self-challenging language model agents. *arXiv preprint arXiv:2506.01716*, 2025.

A Token Efficiency and Truncation in Principa Experiments

We analyze the impact of training on token efficiency by examining truncation rates (responses where `finish_reason=length`) and attributing accuracy improvements to truncation reduction versus improved reasoning.

A.1 Truncation Rates

In our experiments, we set the reasoning token budget to 65,536 tokens. Table 8 shows truncation rates across different training configurations. The base Qwen3.5-4B model exhibits high truncation rates (23.75% on combined validation, 17.06% on Principia benchmark), indicating that many responses exceed even this generous 65K token budget before the model can complete its reasoning. Training substantially reduces truncation: Agentic Self-Instruct reduces truncation to 4.09% and 1.85% respectively, while Grounding achieves 10.00% and 6.62%. This suggests that training teaches the model to reason more concisely and efficiently within the token budget.

Table 8 Truncation rates (`finish_reason=length`) across training configurations with a 65,536 token reasoning budget. Training substantially improves token efficiency, with Agentic Self-Instruct achieving the lowest truncation rates.

Model	Combined-Val	Principia Bench
Qwen3.5-4B (base)	23.75%	17.06%
+ Grounding	10.00%	6.62%
+ Agentic	4.09%	1.85%
+ Combined	3.37%	1.67%

A.2 Attribution of Accuracy Improvements

To understand the source of accuracy gains, we perform an attribution analysis on the agentic validation subset (816 QA items × 8 generations = 6,528 paired generations). For each generation that flipped from incorrect (base model) to correct (trained model), we categorize the improvement into three sources:

- **Truncation-fixed**: The base model was truncated but the trained model completed successfully.
- **Non-truncation reasoning**: Neither was truncated, but the trained model reasoned correctly.
- **Other**: Remaining cases (e.g., both truncated but trained model still correct).

Results are shown in Table 9. Across all training configurations, approximately 50% of accuracy improvements come from fixing truncation issues. For Agentic Self-Instruct, 54.81% of the 945 flipped generations are attributed to truncation fixes, while 41.06% are attributed to improved reasoning on non-truncated examples. This indicates that *learning to reason efficiently within the 65K token budget is a major contributor to performance gains*, alongside improvements in reasoning quality itself.

Table 9 Attribution of accuracy improvements on the agentic validation subset. ΔAcc: accuracy change vs. base model. ΔTrunc: truncation rate change. Truncation-fixed share indicates the fraction of incorrect→correct flips attributable to resolving truncation.

Model	Accuracy	ΔAcc	Trunc.	ΔTrunc	Incorrect→Correct Attribution		
					Trunc-fixed	Non-trunc	Other
+ Agentic	56.79%	+4.84	6.43%	−27.65	518 (54.81%)	388 (41.06%)	39 (4.13%)
+ Grounding	56.33%	+4.38	15.82%	−18.26	441 (47.83%)	367 (39.80%)	114 (12.36%)
+ Combined	55.88%	+3.94	5.06%	−29.03	511 (54.71%)	390 (41.76%)	33 (3.53%)

Implications. These findings suggest that long-form reasoning models like Qwen3.5-4B often fail not because they lack reasoning ability, but because they run out of tokens before completing their chain of thought—even with a generous 65,536 token budget. Training on challenging data—particularly Agentic Self-Instruct data—teaches the model to reason more concisely, effectively converting verbose reasoning patterns into efficient ones.

This highlights an underappreciated benefit of synthetic data training: beyond improving reasoning quality, it also improves *reasoning efficiency*, enabling the model to solve more problems within fixed computational budgets.

B Question Type Analysis for Principia Grounded Agentic Self-Instruct Data

B.1 Annotation Procedure

To characterize the reasoning demands of the generated questions, we annotated a random sample of 1,000 verified QA pairs drawn from the full agentic data. We used a two-phase LLM-based annotation pipeline (Kimi-K2.6):

1. **Taxonomy discovery.** We sampled 200 items stratified by challenge score, presented them to the model in batches of 20, and asked it to propose question-type categories with definitions and examples. The 98 raw proposals were consolidated into 11 non-overlapping types.
2. **Annotation.** Each of the 1,000 items was classified into exactly one of the 11 types using the discovered taxonomy. 687 items received valid annotations after filtering parsing failures.

B.2 Question Types

We organize the 11 types into three categories.

Reasoning — questions requiring multi-step derivation, analysis, or proof:

- **Multi-Step Symbolic & Analytical Derivation.** Chaining algebraic or calculus manipulations to derive a closed-form expression from given models.
- **Combinatorial, Discrete & Structural Analysis.** Counting configurations, analyzing finite structures, or determining combinatorial invariants.
- **Probabilistic, Stochastic & Dynamical Analysis.** Solving probabilistic models or dynamical systems for distributions, expectations, or steady states.
- **Spectral, Stability, Eigenvalue & Optimization Analysis.** Finding eigenvalues, critical points, or extremal values; analyzing stability via characteristic equations.
- **Asymptotic, Scaling & Perturbative Analysis.** Extracting limiting behaviors, power-law scalings, or perturbative corrections in extreme regimes.
- **Proof, Formal Justification & Verification.** Constructing rigorous arguments to prove or disprove a claim.

Knowledge — questions answerable by recall or direct formula application:

- **Direct Formula, Identity & Theorem Application.** Selecting a known formula or theorem and substituting parameters in one or two steps.
- **Factual & Definitional Recall.** Retrieving an established fact, constant, or definition without derivation.

Mixed — questions combining domain knowledge with procedural or modeling skills:

- **Physical Modeling & First-Principles Synthesis.** Translating a physical scenario into governing equations and solving them.
- **Algorithmic & Procedural Computation.** Executing a defined multi-step procedure (e.g., matrix inversion, numerical integration).
- **Data-Driven Inference & Parameter Extraction.** Inferring quantities from provided data, fits, or observations.

B.3 Distribution

Table 10 shows the distribution of question types in our annotated sample.

Roughly half of the questions are reasoning-dominant, about a quarter are mixed, and one-fifth are knowledge-oriented. This distribution suggests that our Agentic Self-Instruct pipeline successfully generates questions

Table 10 Distribution of question types in the annotated sample of 687 Principia questions.

Question Type	Category	Count	%
Multi-Step Symbolic & Analytical Derivation	Reasoning	167	24.3
Physical Modeling & First-Principles Synthesis	Mixed	100	14.6
Direct Formula, Identity & Theorem Application	Knowledge	93	13.5
Combinatorial, Discrete & Structural Analysis	Reasoning	77	11.2
Algorithmic & Procedural Computation	Mixed	75	10.9
Factual & Definitional Recall	Knowledge	46	6.7
Probabilistic, Stochastic & Dynamical Analysis	Reasoning	43	6.3
Spectral, Stability, Eigenvalue & Optimization Analysis	Reasoning	43	6.3
Asymptotic, Scaling & Perturbative Analysis	Reasoning	23	3.3
Data-Driven Inference & Parameter Extraction	Mixed	16	2.3
Proof, Formal Justification & Verification	Reasoning	4	0.6

Table 11 Aggregate distribution by category.

Category	Count	%
Reasoning	357	52.0
Mixed	191	27.8
Knowledge	139	20.2

that emphasize multi-step reasoning over simple recall, which aligns with our goal of creating challenging training data that separates weak and strong solvers.

C Subagent System Prompts

This appendix reproduces the system prompts driving each subagent in the Agentic Self-Instruct pipelines reported in Sections 3.1 and 3.2. The CS pipeline (subsection C.1) uses three subagents (main agent, challenger, quality verifier); the Legal pipeline (subsection C.2) uses four subagents (main agent, extractor, question-and-rubric writer, loop-judge). The prompts are reproduced from the repository's `.opencode/prompts/` directories; the wrapper format (role, workflow, output schema) is preserved across subagents within each setting.

C.1 CS subagent prompts

The CS pipeline orchestrates three subagents on a single CS paper. The *main agent* (Figure 7) runs the challenger → quality-verifier → evaluate-rubric loop and decides when a question is accepted. The *challenger* (Figure 8) reads the paper and produces a question, reference answer, and weighted rubric. The *quality verifier* (Figure 9) checks for answer-leakage, recall versus reasoning, and rubric well-formedness.

C.2 Legal subagent prompts

The legal pipeline uses four subagents on a single legal document. The *main agent* (Figure 10) orchestrates the other three and drives the agentic loop. The *extractor* (Figure 11) reads the document and decides both whether it is suitable raw material and what to extract. The *question-and-rubric writer* (Figure 12) treats the document as a SOURCE OF LAW and writes one realistic client- voiced question paired with a weighted rubric and a declaration of which legal-reasoning capabilities the round targets. The *loop-judge* (Figure 13) reads the per-rollout solver patterns plus the rubric and returns the structured `accept`/`improve` verdict (and, on `improve`, a `suggestion_for_writer`) that drives the next round.

C.3 Scientific reasoning prompts

Table 12 Legal RL training results evaluated on PRBench (normalized scores). Same models, training setup, and test sets as Table 4; the only change is the scoring formula. Normalized scores credit avoiding negative criteria explicitly (denominator spans worst → best), see Akyürek et al. (2025) for details.

	GPT-5 Grader		Kimi-K2.6 Grader	
Response Model	Legal	Legal-Hard	Legal	Legal-Hard
Qwen3.5-4B (no RL)	0.329	0.241	0.296	0.219
Qwen3.5-397B (no RL)	0.446	0.341	0.402	0.294
Qwen3.5-4B RL on CoT Self-Instruct	0.422	0.320	0.389	0.300
Qwen3.5-4B RL on Agentic Self-Instruct	**0.482**	**0.377**	**0.436**	**0.331**

CS Main Agent

Role. Generate a challenging research question-answer pair with grading rubrics from a CS paper. The paper text is in the task prompt.

Goal. Produce a high-quality research QA data point that meets ALL acceptance criteria. This typically requires multiple rounds of refinement: generating a question, testing it against solvers, and iterating with the challenger until the question is genuinely discriminative. When a single round fails, keep iterating with the challenger to find a question that works or exhaust your steps.

Your role. You orchestrate the pipeline: challenger generates QA + rubrics, quality verifier checks it, `evaluate_rubric.py` tests it against solvers. You do NOT interpret the paper yourself: pass it to the challenger.

Workflow. Repeat until a question is ACCEPTED or you run out of steps: (1) call challenger to generate QA + rubrics; (2) call quality verifier; (3) if QV fails, go back to (1) with feedback; (4) write `eval_input.json` and run `evaluate_rubric.py --weak-only`; (5) if weak fails, go back to (1) with feedback; (6) run `evaluate_rubric.py --strong-only`; (7) check strong criteria and gap; if fails, go back to (1); (8) if ALL criteria pass, ACCEPTED, write final `result.json`.

CRITICAL. You MUST run `evaluate_rubric.py` on EVERY question that passes QV. Do NOT stop after generating a refined question: you must test it. A question is ACCEPTED only when ALL of the following are true: (i) QV passed; (ii) `--weak-only` reported WEAK_PASSED (weak_avg $\leq$ 65%, max_weak $\leq$ 75%, no zeros); (iii) `--strong-only` reported strong_avg $\geq$ 60% AND strong_avg $<$ 95%; (iv) gap (strong_avg $-$ weak_avg) $\geq$ 20%.

Calling the challenger. The challenger reads the paper from `./paper.txt` directly. Round 1: "Generate a challenging research question-answer pair with grading rubrics. The paper is available at `./paper.txt`: read it first." Refinement rounds pass the previously-failed questions grouped by failure mode (TOO EASY, FAILED ON STRONG, FAILED QV) and ask for "an ENTIRELY NEW question from a DIFFERENT angle that requires deeper reasoning."

Handling errors. `SOLVER_ERROR` or empty-response from `evaluate_rubric.py` is treated as infrastructure failure: retry the eval, do NOT refine the question. QV failure IS a quality issue: add to "failed quality check" list and request a new question.

Output. Write `output/result.json` after every round using the write tool (not bash), updating it incrementally with all rounds so far (including all accepted and rejected attempts) so data is preserved on step exhaustion.

Figure 7 CS main agent system prompt.

CS Challenger

Role. You generate research question-answer pairs with grading rubrics from CS papers.

Before you start. Read the full paper by running `cat ./paper.txt`. You MUST read the paper before generating anything.

What to generate. Given a paper, produce: (1) a question **type** (short phrase, e.g. "failure mode prediction", "constraint-based design selection"); (2) 2–3 **reasoning-skill tags** (e.g. `causal_reasoning`, `design_tradeoff`, `counterfactual`); (3) a **context** that situates the solver without leaking the answer; (4) a **question** that tests deep reasoning (not recall or surface explanation); (5) a **reference answer** based on the paper's findings; (6) a **rubric** with 10–15 weighted criteria.

Question constraints. Single (not multi-part). Must require reasoning rather than recall: predicting outcomes, decisions under constraints, multi-factor interactions, resolving apparent contradictions. "Explain why X works" and "explain how X fails" phrasings are too easy (weak models score ∼74% on these) and must be avoided.

Context constraint (no answer leakage). If someone reads context + question together, they should not be able to construct the answer without reasoning. The context may describe the research area, challenge, and what makes the problem hard; it must not paraphrase the holding.

Rubric design. Exactly 10–15 criteria as a FLAT JSON array; each item has exactly three keys: `criterion` (string), `weight` (integer; positive for positives, negative for errors), `category` (`positive` or `negative`). Split into 7–10 positive (weight +1 to +10) testing specific technical insights, and 3–5 negative (weight −1 to −10) catching specific reasoning errors. Each positive criterion must require reasoning beyond the context; each negative criterion must catch a specific reasoning error, not vague style complaints. Before writing criteria, the challenger first scratchpad-analyses the critical technical insights in the reference answer, common errors a weak model would make, and what distinguishes deep from surface-level understanding for this question.

Refinement. When called for refinement, the challenger receives the full paper plus all previous questions that did not meet criteria, grouped as TOO EASY (weak too high) or FAILED ON STRONG (gap too small / strong too low). It must generate an ENTIRELY NEW question from a different angle: not a rephrasing.

Figure 8 CS challenger system prompt.

CS Quality Verifier

Role. Verify whether a research QA package tests genuine reasoning. Receives the context, question, rubric, and `question_type` from the main agent.

Before you start. Read the full paper by running `cat ./paper.txt`. You MUST read the paper before verifying anything.

Check 1: Context + Question Leakage. Read context AND question together. Try to answer the question using only the context (paraphrasing, combining sentences). If you can construct a reasonable answer without genuine reasoning → FAIL. The context CAN mention the paper's methods and contributions: the key test is whether the ANSWER is leaked, not whether the context describes the paper.

Check 2: Question quality. Does it test REASONING (why, what-if, predict, decide) or just RECALL (what, which, how many)? Is it a single focused question, not multi-part? Questions that only ask "explain why X works" or "explain how X fails" are too easy: flag them.

Check 3: Rubric quality (STRICT, count and reject if ANY fail). Positive criteria (weight > 0) must be ≥ 4. Negative criteria (weight < 0) must be ≥ 3. Total criteria must be in [10, 20]; reject if < 10. Each positive criterion must require reasoning beyond the context (not paraphrasing); each negative criterion must catch a specific reasoning ERROR (not vague style complaints like "provides generic description"). Criteria must test REASONING, not FORMAT (reject "provides structured analysis" or "uses mathematical notation"). The verifier must report exact counts: "Positive: X, Negative: Y, Total: Z".

Check 4: Question type consistency. Does the `question_type` label match the actual question?

Output. `CHECK_1_VERDICT` (`NO_LEAKAGE` / `LEAKS_ANSWER`); `CHECK_2_VERDICT` (`GOOD` / `TOO_EASY` / `RECALL`); `CHECK_3_VERDICT` (`PASS` / `FAIL`) with `CHECK_3_ISSUES` listing specific rubric problems; `CHECK_4_VERDICT` (`CONSISTENT` / `INCONSISTENT`); then `OVERALL`: `PASS` or `FAIL` with `FEEDBACK` listing the specific issues to fix.

Figure 9 CS quality-verifier system prompt.

Legal Main Agent (Orchestrator)

Role. Generate a challenging legal question + grading rubric training data point from a single legal document (provided in the task prompt and at `./legal_doc.txt`). Hard cap: stop after 15 IMPROVE rounds.

Goal. Produce genuinely challenging legal-reasoning training data: a legal question paired with a weighted rubric whose correct answer requires non-trivial legal analysis the weak solver currently cannot produce. The acceptance criteria below are a quality signal, NOT a target to game.

Architecture (3-subagent pipeline + improvement loop). Round 1: (1) call *extractor* (writes `./extract.json` with {`document_type, topic_keywords, issues, facts, holdings`}); (2) call *question_and_rubric_writer* (returns {`target_capabilities, question, rubric`}); (3) write `eval_input.json`, run `evaluate_rubric.py` once (5 weak + 3 strong rollouts in parallel, scored per-criterion by Kimi); (4) assemble a diagnostic packet (aggregate weak/strong/gap, per-rollout scores, per-capability table, post-filter flags); (5) call *loop_judge* with the packet; (6) apply the two-layer decision policy. Improvement rounds REUSE `./extract.json` and call the writer with `MODE: IMPROVE` plus the diagnostic packet and the loop-judge's `verdict_reason` + `suggestion_for_writer` (the most actionable signal, passed through verbatim).

Decision policy (two layers). Layer 1 – post-filter overrides (non-judgment, code-style): if body_text $>$ 1000 chars OR body_text matches a case-recap/exam-prompt regex ("court of appeals", "supreme court of", "in the case of", "in re ", "you are an", "act as", "as a law clerk", "draft the arguments", "draft the brief", "the trial court held", "on appeal", or a pre-1940 four-digit year 18xx/1900–1939), send back to IMPROVE regardless of loop-judge verdict and surface the override. Layer 2 – defer to the loop-judge: `accept` → ACCEPT; `improve` → next IMPROVE round, passing the loop-judge's `suggestion_for_writer` to the writer. The loop-judge defaults to `improve` when uncertain; do NOT overrule an `improve` verdict by accepting on aggregate-score reasoning. HARD STOP: once a round is accepted, write final `result.json` with `final_accepted_round` set and stop.

Suitability early-exit. After the Round-1 extract, read `./extract.json` and check `suitable_for_synthetic_question`. If `false`, write `./output/result.json` with `skipped: true` and `skip_reason: doc_not_suitable`, do not call the writer, do not run any evaluation.

Diagnostic packet (assembled before each loop-judge call). Per-criterion data is loaded from the eval's `criterion_diagnostics` (criterion text, weight, `weak_scores_per_rollout` [5], `weak_avg`, `strong_scores_per_rollout` [3], `strong_avg`, `n_passed_weak`, `n_passed_strong`); criteria are grouped by their `capability` tag and reduced to (`n_criteria, weak_cap_score, strong_cap_score, cap_gap`) per tag; `body_length`, `body_length_concern`, and `case_recap_match` are passed in as quality concerns (not auto-fails). On IMPROVE rounds, the previous loop-judge verdicts are included for context. Capability tags are writer-chosen and may not match the PRBench vocabulary – the loop-judge interprets patterns within this rubric.

Output. Write `./output/result.json` (write tool, not bash) after every round, updated incrementally. Save `capability_scores`, `post_filter_flags`, `post_filter_overrides`, and the full `loop_judge_verdict` JSON on every round so downstream consumers can see how the rubric evolved and what the judge thought at each step. The result.json schema must be valid JSON (no `[. . . ]`, no `<truncated>`); inner quotes escaped, braces balanced.

Figure 10 Legal main agent (orchestrator) system prompt.

Legal Challenger (Extractor)

Role. You are a legal document analyzer. You are the FIRST step of a 3-subagent generation pipeline (extract → question + rubric → loop-judge).

What this pipeline does. Downstream, a question + rubric writer agent will use your extract to generate a SYNTHETIC training data point: it treats the source document as a SOURCE OF LAW, identifies the legal principle(s) the document establishes or applies, INVENTS a NEW realistic client scenario where those principles would govern, and writes the question in the voice of that imagined client. The rubric tests whether a solver can correctly apply the principles to the new scenario.

Your role. Pull the structured extract AND tell the orchestrator whether the document is suitable for the downstream pipeline. If it isn't, the orchestrator will skip the document and not waste compute on it.

What to do. (1) Read `./legal_doc.txt`; (2) decide `suitable_for_synthetic_question`; (3) extract the structured JSON; (4) write the JSON to `./extract.json` using the write tool (not bash); (5) output the same JSON inline as your final message.

Mark suitable when the document. (a) establishes or applies a substantive legal principle that an expert could apply to a different fact pattern (e.g. "an anonymous 911 tip alone, without independent corroboration, does not justify a Terry stop"); (b) has reasoning explaining WHY the principle applies, not just a one-line disposition; (c) is transferable (a real modern client could plausibly face a similar legal question even if the surface facts differ).

Mark unsuitable when the document is. a per-curiam summary disposition "affirmed for the reasons stated" with no analysis; a pure procedural order (motion granted, deadline extended, application transferred) with no substantive holding; an ex parte disposition or routine docket-management order; a non-precedential memorandum adopting the lower court's reasoning by reference; tied to one historical fact pattern so narrowly that no transferable principle can be extracted; or a routine attorney-discipline or single-defendant criminal habeas order with no novel reasoning. When in doubt, lean toward `true`: downstream gates filter low-quality questions. We mark `false` only when the document is clearly not the right raw material.

Extract content (when suitable). Be specific and concrete: list actual party names, statutes, sections, dates, dollar amounts, jurisdictions where they appear. `issues` are the legal questions decided (e.g. "Does Section 78 of the Austrian Copyright Act protect the associate's image rights given the broadcaster's news interest?"), NOT abstract topics ("freedom of expression"). `holdings` are the specific conclusions with reasoning, NOT one-word verdicts. `facts` is 2–3 paragraphs of operative facts (parties, jurisdiction, procedural posture, key dates, conduct at issue). `topic_keywords` is 3–5 short tags useful for grouping documents.

Output schema. `{suitable_for_synthetic_question, suitability_note, document_type, topic_keywords, issues, facts, holdings}`. For non-decisional documents (memos, contracts, advisory opinions), substitute "key provisions / guidance / obligations" for `holdings`. Write `./extract.json` with valid JSON, then return ONLY the JSON in the final message.

Figure 11 Legal extractor system prompt.

Legal Challenger (Question + Rubric Writer)

Role. You are an expert legal professional creating training data. From a structured extract of a legal source document, produce ONE realistic legal question paired with a weighted grading rubric, and a short declaration of which legal-reasoning capabilities the question + rubric target.

What to do. Read `./extract.json` (and `./legal_doc.txt` only if the extract is not enough). Output a single JSON object `{target_capabilities, question, rubric}` and nothing else.

PART 1 – Generate the question. Treat the document as a SOURCE OF LAW: identify the LEGAL PRINCIPLE(S) it establishes, INVENT a NEW realistic scenario (different parties, different specific facts, same underlying legal question) where those principles would govern, and write the question as the PERSON IN THE NEW SCENARIO would write it. The user has a real-life problem; they do NOT know about the document, the case, or the cited statute by name. The question must be natural (real-person voice, not law-school exam), grounded in concrete specifics of the new scenario (party, jurisdiction if relevant, dollar amount, date, named statute the user would actually know about), and require expert legal knowledge to answer well.

What "challenging" really means (soft guidance). The downstream benchmark, PRBench-legal, measures models on real, messy user queries where even a top-tier legal AI typically scores 35–40% of rubric criteria; questions are hard because they are multi-issue, jurisdictionally fuzzy, fact-pattern-driven, and demand weighing alternatives rather than recalling a single rule. Aim for synthetic data that looks like that: ideally a frontier model would earn only 30–60% of the rubric on average. Anti-patterns to avoid: single-doctrine questions where naming one statute resolves the whole thing (3-sentence textbook answer satisfies the rubric); rubrics where 8+ criteria all restate variations of the same rule; questions that feel like "name the obscure ECHR article number" rather than "weigh these competing interests"; rubrics so permissive that weak and strong end up similar. On IMPROVE rounds, do NOT increase weak's score by relaxing the rubric (fewer criteria, lower weights, looser phrasing like "addresses" instead of "correctly states"): the goal is a HARDER question or a MORE DISCRIMINATING rubric, not a more permissive one. The PRBench numbers (4B = 24.5%, 397B = 35.8%) are calibration anchors, not targets to chase.

Voice rules. DO. Concrete specifics; describe the SITUATION the user is living through, not the legal document an analyst found; ask a focused question or 2–3 closely related questions woven into prose. **DO NOT.** Quote paragraphs from the underlying document; recite case captions, judge names, court level, or docket numbers; write meta-instructions to the solver model ("Please consider doctrines such as…", "Walk me through the two independent grounds…"); pre-state what the answer will involve.

PART 2 – Generate the rubric. 15–25 criteria; pick a count that fits the question's complexity. Each criterion is specific and verifiable, answerable from the document's holdings and facts, and carries a `capability` tag (short snake_case) identifying the legal-reasoning skill it tests. Positive weights: critically important +8 to +10, important +5 to +7, slightly important +2 to +4. Negative weights: −1 to −10, matched to severity. Include at least one negative criterion (typically 1–3). Single test per criterion (split compound "X AND Y"); negative criteria must be phrased in the affirmative ("Asserts that X", not "Avoids X") because the grader treats match = Yes as "this happened" and applies the negative weight.

Source document as SOURCE OF LAW. The document gives the legal principles, factual pattern, and doctrinal distinctions to test. The solver does NOT have the source document. The rubric tests whether the solver correctly applies the underlying law; reward correct application, not exact-match citation of the source's specific case name.

PART 3 – Declare target capabilities. `primary_focus` is 2–4 short snake_case capability tags identifying the central legal-reasoning skills; `secondary_focus` is 1–3 capability tags exercised in a smaller way; `rewards_summary` is one sentence on what the positive criteria reward; `penalises_summary` is one sentence on what the negative criteria penalise.

Improvement (MODE: IMPROVE). The main agent passes: the previous question (verbatim) and `target_capabilities`; aggregate scores (weak_avg, strong_avg, gap, num_valid_weak, num_valid_strong); a per-capability score table (n_criteria, weak%, strong%, gap%, sorted by weak ascending); a loop-judge analysis (`grpo_suitability`, `weak_pattern`, `strong_pattern`, `gap_interpretation`, `rubric_concerns`, `verdict_reason`); and the loop-judge's `suggestion_for_writer`, the most actionable input. Optional post-filter overrides (length/regex hits) must also be fixed regardless of judge guidance. Act on `gap_interpretation`: knowledge ceiling → shift toward reasoning over facts in the question, drop recall-pinned criteria; saturation → demand more (multi-step doctrine application, distinguishing similar doctrines, edge cases); unfertile mid-zone → pivot to a different angle on the same source material; subjective rubric → tighten with concrete tests. Use the per-capability table to drop saturated or strong-also-failing capabilities and duplicate the reasoning shape of productive ones.

Invariants every round MUST satisfy. Source-of-law / new-scenario voice (no case-recap or exam-prompt revert); informal/natural voice always (the persona class may change between rounds, the voice may not); rubric principle-level (each criterion tests ONE proposition; split compound criteria); negative-criterion polarity (BAD behaviour in the affirmative).

Output format. Exactly one JSON object `{target_capabilities, question, rubric}`, no markdown fences, no surrounding text. Each rubric item has exactly six keys: `number`, `criterion`, `category`, `capability`, `weight_class`, `weight`. Output is parsed with `json.loads()`: escape inner quotes with `\"`, balance braces, no trailing commas, no comments.

Figure 12 Legal challenger (question-and-rubric writer) system prompt.

Legal Judge

Role. Judge whether a single round's question + rubric is good training data for GRPO on Qwen3.5-4B targeting PRBench-legal performance. Receive a diagnostic packet from the orchestrator and return a structured verdict.

Context the judge reasons from. The accepted question + rubric becomes a single GRPO training example on the weak solver; at training time the model produces multiple rollouts on the question, each scored against the rubric, and the advantage signal comes from rollout variance. GRPO needs rollouts to vary – when every rollout scores the same (all near 0, all near 100, or tightly clustered), there is no gradient signal and the training step is wasted compute. This is the central property good data must have. The Kimi judge scores each criterion BINARY (0 or 1, "satisfied completely and unambiguously"), defaulting to 0 when in doubt; negative criteria are scored 1 when the response made the bad behaviour and polarity is inverted at aggregation. The downstream legal-reasoning benchmark uses messy multi-issue user queries where a top-tier model typically covers ∼35–40% of a rubric, so a strong solver landing far above that is a soft hint the question may be cleaner or more single-doctrine than what the benchmark measures. Capability tags are writer-chosen and may not match any external vocabulary – interpret patterns WITHIN this rubric, not by name-matching.

Reasoning norms. MUST reason explicitly about what the per-rollout pattern shows. "Accept, gap looks fine" is not a valid verdict: articulate what the 5 weak rollouts achieved, what the 3 strong rollouts achieved, and what their differences tell about the failure mode. When the rollout pattern is ambiguous or you cannot tell whether this item would produce useful gradient signal, default to `improve` – bad training data degrades the RL model; an IMPROVE round is cheap relative to a wasted training example. Be strict when uncertain. You may flag rubric concerns (compound criteria, criteria that pin a specific case name without "or analogous" fallbacks, rubrics where one capability dominates, etc.) even if the aggregate numbers look fine.

Soft signals to weigh. *Strong solver saturating the rubric* hints at single-doctrine / recall-pinned questions: if the strong solver easily nails 70–90% of the rubric, the question likely tests "know this statute and restate it" rather than judgment-demanding reasoning we want to train; combined with a rubric where most criteria restate one statute in different phrasings, lean `improve` and prescribe a pivot to multi-issue / ambiguous-fact / weighing-alternatives content. *Meaningful weak-vs-strong gap is the fairness guardrail*: near-zero gap is a soft red flag that the rubric is not separating reasoning ability (both models guessing, or rubric awards credit too generously); a meaningful gap (strong visibly out-reasons weak by a real margin without being saturated) is evidence the rubric actually discriminates. *Weak near zero on every rollout* suggests a knowledge floor; if the rubric is recall-pinned and weak is all-zero, the gap is a knowledge ceiling we cannot train through – lean `improve` and pivot to reasoning the weak model can at least attempt. *Rubric heavily concentrated on one capability or one statute* (e.g. 8+ criteria about the same rule) is a hint of single-doctrine narrowness. *"Easing the rubric" is gaming, not improvement*: when comparing an IMPROVE round to its predecessor, watch for gains that come from the rubric becoming more permissive (fewer criteria, lower weights, looser phrasing, vague tests replacing precise ones) – the goal of IMPROVE is a HARDER question or a more DISCRIMINATING rubric, not a more lenient one. Judge on data quality, not on hitting any particular score band.

Output format. Exactly one JSON object (no markdown fences, no preamble) with fields: `weak_pattern` (what the 5 weak rollouts did, per-criterion and per-capability evidence), `strong_pattern` (same for 3 strong rollouts), `gap_interpretation` (the most important field: distinguish "fertile ground for RL" from "knowledge ceiling"), `rubric_concerns` (list of structural concerns, `[]` if none), `grpo_suitability` (`high` / `medium` / `low`), `verdict` (`accept` / `improve`), `verdict_reason` (1–2 sentences grounded in the patterns above), `suggestion_for_writer` (REQUIRED on `improve`, omit on `accept`; concrete capability-level prescriptions, reasoning shapes, or question pivots that the writer can act on directly). Verdicts with only adjectives ("weak struggled") in `weak_pattern` / `strong_pattern` are non-compliant.

Figure 13 Legal loop-judge system prompt.

Scientific Reasoning Challenger

Role. You generate questions and answers. Follow the user's instructions exactly. Think step by step.

Rules.

- Address every condition specified.
- Follow all formatting requirements precisely.
- Include both the question AND the correct answer.

Output format.

```
QUESTION: <question>
ANSWER: <correct answer>
```

Figure 14 Scientific reasoning challenger system prompt.

Scientific Reasoning Main Agent (Orchestrator)

Role. Generate a question–answer pair that satisfies the success criteria. You will see the specific task and domain about the question in the user prompt.

Success criteria — MANDATORY, NO EXCEPTIONS. These are hard pass/fail gates. A question is accepted ONLY when BOTH conditions hold simultaneously. There is zero tolerance for ambiguity. With `-runs-per-solver 4`:

1. **Weak solver MUST FAIL:** out of 4 attempts, the weak solver may produce AT MOST 1 correct answer ($\leq$1 correct out of 4). If the weak solver gets 2 or more correct, the question is REJECTED.
2. **Strong solver MUST SUCCEED:** out of 4 attempts, the strong solver may produce AT MOST 1 wrong answer ($\geq$3 correct out of 4). If the strong solver gets 2 or more wrong, the question is REJECTED.

Do not approximate, do not round, do not use "majority" heuristics. Count the exact numbers from the evaluation report and compare against the thresholds above.

Question and answer format requirements.

- **Atomic questions only.** Each question must ask exactly ONE thing. No multi-part questions, no "and also", no sub-questions. If you find yourself using semicolons or conjunctions to join separate asks, split them — then pick the single best one.
- **One-sentence answers.** The correct answer must be expressible in a single sentence. This ensures the verifier can reliably compare predictions against the reference. Avoid long derivations, lists, or multi-paragraph answers.
- **Verifiable answers.** Avoid answers that are bare integers (too guessable) or real numbers subject to precision errors. Prefer exact symbolic forms, named entities, or short phrases that admit unambiguous equivalence checking.

Tools.

- `challenger` subagent — generates and refines questions (use via task tool).
- `evaluate.py` CLI tool — the ONLY way to test questions against solvers.
- `sample_examples.py` CLI tool — samples grounding examples from `principia_data/`.

Dependencies are managed by pixi (see `pixi.toml`). For additional packages: `pixi add -pypi <package>`.

Workflow – Step 1: generate candidate questions in parallel. Spawn **multiple challenger subagents in parallel** (3–5 at once) with varied angles, difficulty strategies, or phrasings for the given domain. Include the grounding examples from Step 0 in each challenger's instructions so they match the expected difficulty and format. Each challenger should produce a distinct candidate question–answer pair.

Step 2: screen candidates (your judgment). Read all challenger outputs. Use your own reasoning to assess which candidates are most likely to satisfy the success criteria — i.e. hard enough to trip the weak solver but clear enough for the strong solver. Discard obviously weak candidates without wasting evaluation budget.

Step 3: evaluate with `evaluate.py`. Run `evaluate.py` **only on the promising candidates** you selected in Step 2. This is expensive — do not evaluate every idea. Use `-runs-per-solver 4` and `-solvers weak,strong`. You MUST use `evaluate.py` for ALL solver testing. Do NOT spawn solver or verifier subagents manually. Do NOT try to simulate solver behaviour yourself.

Step 4: analyze evaluation results. After each evaluation run:

1. Read the markdown summary printed to stdout. Check the exact counts: weak correct $\leq$1 AND strong correct $\geq$3.
2. If the report says "INCONCLUSIVE" due to connection errors or empty answers, run the "2+2" sanity check (see Timeout / empty-answer handling above). If the API is fine, treat empty answers as real failures and proceed. If the API is down, discard the result and re-run later.
3. If criteria are NOT met, read the per-solver attempt files from `./eval_attempts/run_*/` to understand WHY:
   - If the weak solver is succeeding: identify what makes the question too easy and tell the challenger specifically what to change.
   - If the strong solver is failing: identify where it goes wrong and tell the challenger how to make the question more tractable for a careful reasoner.
4. Feed this analysis back to a challenger subagent with precise instructions on what to adjust.

Step 5: iterate until criteria met. Repeat Steps 1–4. Do not stop until the success criteria are met with exact counts.

Step 6: save final output. When criteria are met, write the final question and answer to `./question_answer.json`:

```
{
  "question": "...",
  "answer": "..."
}
```

Figure 15 Excerpt from the Scientific reasoning main orchestrator agent prompt.