WORKFLOW-STYLE INSTRUCTION FOLLOWING WITH ATOMIC SYNTHESIS AND ADAPTIVE RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Complex instruction following is almost always scaled along one axis—more constraints on a single task—yet many instructions are workflows: interdependent sub-tasks under conditional branches, the hardest of which turn on the model's own intermediate output, so the correct route cannot be known before the response is written. Such execution flow is largely absent from existing data, and checklist-style verification does not match it: a checklist is fixed before the response, whereas the path to be checked is not. We propose CASCADE, an iterative synthesis framework that expands a seed instruction into a **workflow-style complex instruction** by adding exactly one atomic unit per round—a constraint, a sub-task, or a conditional branch. Verification follows the same structure: a check graph models an instruction's execution as a directed acyclic graph, and each response is judged node by node along the path it ought to follow. The same pipeline on held-out seeds yields CASCADEEVAL, a benchmark of such instructions. Because expansion is atomic, synthesis also yields, at no extra cost, a difficulty spectrum for every instruction: the ordered sequence of versions from the seed to the hardest one. We exploit it with SPIRAL, a reinforcement learning (RL) data-selection algorithm that chooses, at each step, which version of each instruction to rollout, keeping training matched to the model's current ability. On Qwen3.5-9B, RL on the hardest CASCADE instructions raises the instruction-following average by **+11.00** points over the strongest existing corpus; SPIRAL adds a further **+1.88** points over that setting while cutting wall-clock training time by **40%**. Our data and code will be made publicly available after peer review.

1 INTRODUCTION

Post-training has substantially improved how reliably large language models (LLMs) follow instructions, most recently by optimizing verifiable rewards with reinforcement learning (RL) (Shao et al., 2024; DeepSeek-AI et al., 2025; Lambert et al., 2024). What counts as a *complex* instruction, however, has stayed narrow: difficulty is almost always scaled by attaching more constraints to a single task, so benchmarks score a response by the atomic constraints it satisfies (Zhou et al., 2023; Jiang et al., 2023; Pyatkin et al., 2025) and synthesis pipelines produce harder data by stacking further constraints onto a seed (Xu et al., 2023; Sun et al., 2024; Qi et al., 2024; An et al., 2025). Complexity grows along one axis; the task underneath stays one task.

Many requests are not of this shape. The instruction in Figure 1 (left) chains several sub-tasks in one turn, each consuming the output of the previous one, under constraints that hold across the response and two conditional branches that decide what is required. Succeeding here means running a small workflow in the right order, not satisfying more constraints. We call such requests **workflow-style complex instructions**, in contrast to the constraint-style complex instructions—one task with ever more constraints—that existing work targets. Workflow-style difficulty grows along three orthogonal axes: the constraints on the response, the number and dependency depth of the sub-tasks, and the branch structure of the execution flow. Branches come in two kinds. A **deterministic branch** takes its condition from information already fixed before the response exists, so the correct path can be worked out in advance; a **non-deterministic branch** takes its condition from the model's own intermediate output, so the correct path emerges only during execution. The latter is the defining structure of this regime: it obliges a model to act, read what it produced, and decide where to go from there.

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

Instruction

Review this spec sheet: `[...] RAM: 8192 bytes | Storage: Audio Cassette`. Write a detailed description of an image of a late-1970s home computer setup. First convert the RAM to kilobytes. If even, use only visual and auditory detail, with no smell or touch; if odd, include at least one simile. Then check whether your description mentions a tape drive, and declare it with '[FLAG: YES]' or '[FLAG: NO]' on its own line. If YES, explain how to load a game from it; if NO, describe the loud sounds the thick keys make. From those steps or sounds, invent a short magazine-article title, then describe the readers it would draw. Throughout: no word over 10 characters, no word more than 3 times, and no commas.

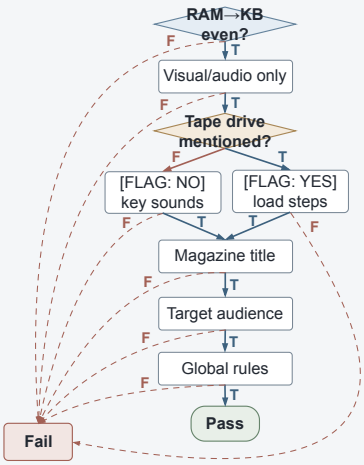


Figure 1: A workflow-style complex instruction (left) and its simplified check graph (right). Rectangles are instruction-constraint nodes, diamonds are route nodes, and dashed red edges lead to failure. The first route node is deterministic, so its untaken false branch is drawn collapsed into failure.

This structure breaks the post-training stack at four points, in a chain. It begins with **data**: instruction-following corpora grow along the constraint axis, so dependent sub-tasks are rare and non-deterministic branches essentially absent. Because such data does not exist, **verification** never had to represent it; complex instructions are checked against a flat list of criteria (Qin et al., 2024; Zhang et al., 2024; Huang et al., 2025), which is fixed before the response exists and therefore either enumerates every branch—failing a response that correctly took one of them—or commits to one branch in advance, downgrading a non-deterministic branch to a deterministic one. **Training** then has nothing to optimize; and once such instructions are synthesized, their difficulty raises a further obstacle: under Group Relative Policy Optimization (GRPO) (Shao et al., 2024), the mainstream RL algorithm for LLM post-training, an instruction whose rollouts all fail contributes no gradient (Yu et al., 2025; Bae et al., 2025)—in our controlled comparison such instructions make up 43.4% of a batch on average. **Evaluation** inherits the gap: the benchmarks that do compose tasks and branches settle every branch condition before the response, so none contains a non-deterministic branch.

We break the chain at its source with **Cascade** (Check-graph Augmented Seed-driven Compound instruction Assembly via Delta Expansion), which synthesizes workflow-style complex instructions by expanding a seed one atomic unit at a time—one constraint, one sub-task, or one conditional branch—so that complexity rises along all three axes together. Verification is built in the same motion: every instruction carries a check graph (Figure 1, right) that models its execution as a directed acyclic graph (DAG) whose route nodes can decide, from the response's own intermediate output, which path it ought to follow, so that verifying the instruction reduces to verifying it node by node along that path. We call this graph-decomposed verification (GDV); each node is decided by an Agentic Verifier (Zhuge et al., 2024) that needs no per-instruction rubric or program. Atomic expansion also settles the training obstacle: the synthesis history is itself a monotone sequence of versions of the same instruction—a **difficulty spectrum**, obtained with no difficulty annotation—from which **Spiral** (difficulty-**S**pectrum-guided **P**rogressive **I**nstruction **R**outing with **A**daptive **L**evels) picks, at every training step, the version of each instruction to train on. Unlike difficulty-aware methods that schedule *across* samples (Shi et al., 2025; Wang et al., 2025), Spiral keeps the question and changes only its difficulty level. Running the same pipeline over held-out seed tasks yields **CascadeEval**, a benchmark for this regime.

Across three base models and seven benchmarks, Cascade data improves instruction following in both supervised fine-tuning (SFT) (Wei et al., 2022; Ouyang et al., 2022) and RL, and Spiral gives the best IF-Avg on all three models (§3). The effect is most pronounced on Qwen3.5-9B, where the hardest Cascade instructions raise IF-Avg, the average over five instruction-following benchmarks, by **+11.00** points over the strongest existing corpus, while Spiral adds a further **+1.88** points over that setting, generates **21%** shorter responses than it, gains **+3.70** points out of distribution, and cuts wall-clock training time by **40%**; on Qwen3.5-4B the share of rollout groups whose rollouts all fail,

and which therefore yield no gradient at all, drops from **43.4%** to **21.5%**, confirming the intended mechanism. On CascadeEval, the best of 48 frontier models solves **92.0%** of the hardest-version instructions and the weakest **8.8%**, and node-level attribution traces the dominant failure of most leading models to a single capability, counting.

Our contributions are threefold.

- **Cascade**, an iterative framework that synthesizes workflow-style complex instructions by adding one atomic unit per round, raising complexity along the constraint, sub-task and execution-flow axes at once, and the check graph it builds in the same motion, which is what makes verification GDV: node-level decisions along the path a response ought to follow.
- **Spiral**, difficulty-adaptive data selection at the granularity of versions within one instruction; to the best of our knowledge, it is the first to select among the versions of a difficulty spectrum built in advance for each instruction.
- **CascadeEval**, a benchmark for this regime; to the best of our knowledge, it is the first instruction-following benchmark to contain non-deterministic branches.

2 Method

Cascade turns one seed instruction into a family of workflow-style complex instructions and, in the same motion, the structure that verifies them (§2.1). Spiral then consumes the by-product of that synthesis—a difficulty spectrum per instruction—during RL (§2.2).

2.1 Data Synthesis

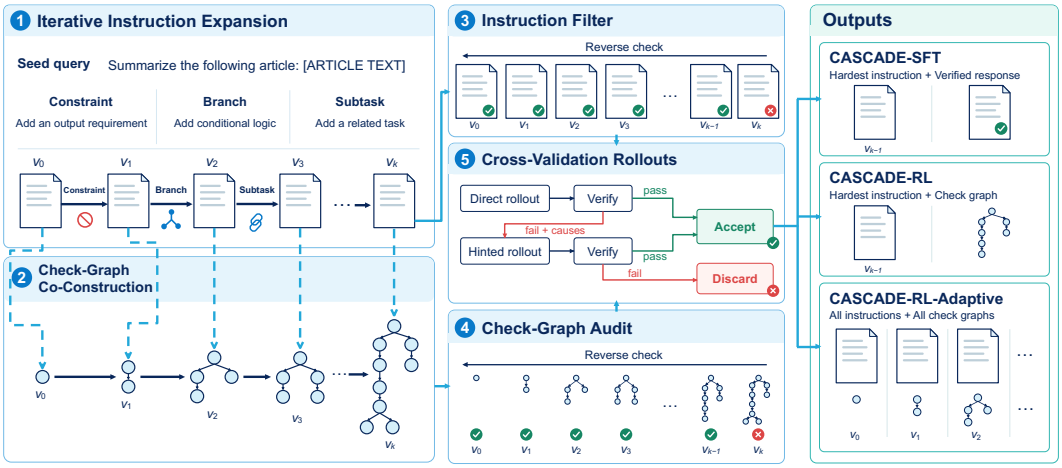


Figure 2: Cascade pipeline. A seed instruction is expanded one atomic unit per round into a difficulty spectrum, with a check graph grown alongside every version; the spectrum is then filtered, audited, and cross-validated against sampled responses, each stage reverting to a shorter prefix on failure.

Atomic expansion. Starting from a seed instruction that is self-contained and objectively verifiable (§B.1 gives the seed sources and the filter that enforces both properties), the pipeline applies exactly one **atomic expansion operator** per round, adding a single minimal logical unit, an **atomic unit**: one constraint, one sub-task (independent of, or dependent on, an existing sub-task), or one conditional branch (deterministic or non-deterministic). Writing $\mathcal{O}$ for that set of five operators (§B.2 defines each one, and the constraint categories it may draw on) and $q^{(k)}$ for the round-k instruction,

$$q^{(k)} = o_k\big(q^{(k-1)}\big), \quad o_k \in \mathcal{O}, \qquad \mathcal{S}(q) = \big(q^{(0)}, q^{(1)}, \ldots, q^{(T_q)}\big), \quad T_q \le K, \tag{1}$$

so that over at most K rounds ($K = 8$ here) a seed becomes a difficulty spectrum whose adjacent versions differ by exactly one unit and whose complexity rises along the constraint, sub-task and execution-flow axes together (Figure 2); scheduling is in §B.3. Atomicity buys two things. First, it makes each round adjudicable: every operator is required to add exactly one unit, and every round is adjudicated against a list of defect categories before it is accepted. Second, when a downstream

check fails—there are three: a filter on the instructions, an audit of the graphs (§B.5), and a cross-validation of both against sampled responses (§B.6)—the spectrum reverts to its last passing prefix, **reverse truncation**, which is what sets T_q; §B.4 gives the three triggers. Since reverting one step is exactly undoing one atomic edit, $q^{(T_q)}$ is the last version on that chain that is still well-formed and solvable, and truncation points differ across instructions, so spectra vary in length. The three corpora this yields are listed in Table 5 of §B.6; §E profiles them—how complexity grows along a spectrum, how execution flow enters the instructions, and what the checks actually test—and §G follows two instructions from seed to hardest version in full.

Check graphs. Verification is constructed in the same motion. Every version carries a check graph G, a DAG whose nodes are of two kinds, $V = V_{\text{chk}} \cup V_{\text{rt}}$. An **instruction-constraint node** $u \in V_{\text{chk}}$ holds one binary check ϕ_u of a constraint or a sub-task; failing it fails the response. A **route node** $u \in V_{\text{rt}}$ realizes one of the instruction's branches: it picks between two onward paths by a predicate β_u and never judges success itself. Route nodes come in the same two kinds as the branches—a deterministic one carries a truth value fixed at construction time and costs no model call, a non-deterministic one is decided at verification time from the response's own intermediate output. A response y is judged along the single path $\mathcal{P}(y, G)$ that the route predicates select:

$$\text{pass}(y, G) = \bigwedge_{u \in \mathcal{P}(y,G) \cap V_{\text{chk}}} \phi_u(y). \tag{2}$$

The entire difference from a checklist sits in the index set of that conjunction: a checklist fixes it before y exists, whereas $\mathcal{P}(y, G)$ is a function of y. Crucially, a route node decides the path the response *ought* to take, not the one it actually took—taking the wrong branch is precisely the failure this regime must expose, and a route node following the model's own choice would hide it. A non-deterministic β_u therefore reads the content the response produced, not any route the response declares. Verification walks $\mathcal{P}(y, G)$ alone, so its cost scales with path length rather than graph size (Figure 1). Each ϕ_u and each non-deterministic β_u is decided by an Agentic Verifier (Zhuge et al., 2024) in the agent-as-a-judge paradigm: it reasons over multiple turns and calls tools, settling checks a single forward pass could not, and needs neither a rubric nor a verification program (§B.5 specifies what it is given at each node, and the audit every graph passes before it is accepted).

CascadeEval. Running the same pipeline over held-out seed task types disjoint from the training ones yields CascadeEval, a benchmark of **125** workflow-style complex instructions spanning **1,125** versions and **6,421** check-graph nodes. Unlike the training corpus, every instruction is run to the full $K + 1$ versions and human-inspected, so that difficulty at a given round is comparable across instructions; §B.7 details these differences and the 13 capability dimensions labeled on every node. We report three metrics (§B.8 defines them formally, together with the node-level attribution built on them): the **instruction-level pass rate** (ILPR), the fraction of instructions for which equation 2 holds at version k, written $\text{ILPR}(k)$; the **completion ratio** (CR), how far along $\mathcal{P}(y, G)$ a model gets before failing; and the **complexity-resilience index** (CRI), the ratio $\text{ILPR}(K)/\text{ILPR}(0)$. The curve $k \mapsto \text{ILPR}(k)$ is the *complexity decay curve*; §F scores 48 frontier models at its two ends.

2.2 Spiral: Difficulty-Adaptive Data Selection

Atomic expansion makes the training data *two-dimensional*, which is what Spiral requires (§C.7 asks what it would take to run it on ordinary one-dimensional data). An ordinary RL sample comes at a single difficulty, so scheduling acts *across* samples; here each instruction is itself the spectrum $\mathcal{S}(q)$ of equation 1, and each step uses one of its versions. The obvious choices are a random version or always the hardest; Spiral instead picks the version whose difficulty best matches the model as it currently stands. It is a data-selection rule layered on GRPO, not a change to the objective—rollout, reward, and policy update are unchanged, and the selected version brings its own check graph, so switching versions does not change how the reward is computed. Its scope is *within* an instruction: it never swaps the question, only the difficulty at which that question is posed (Figure 3).

The pass rate is a free probe. Under GRPO a group of m rollouts contributes a gradient only when its rewards differ. With a binary verdict and group pass rate ρ_q, the mean absolute advantage is $\frac{1}{m}\sum_{i=1}^{m} |A_i| = 2\sqrt{\rho_q(1-\rho_q)}$, which peaks at $\rho_q = 1/2$ and vanishes at the all-pass and all-fail extremes (derived from the GRPO advantage in §C.1); under a binary reward those groups are the

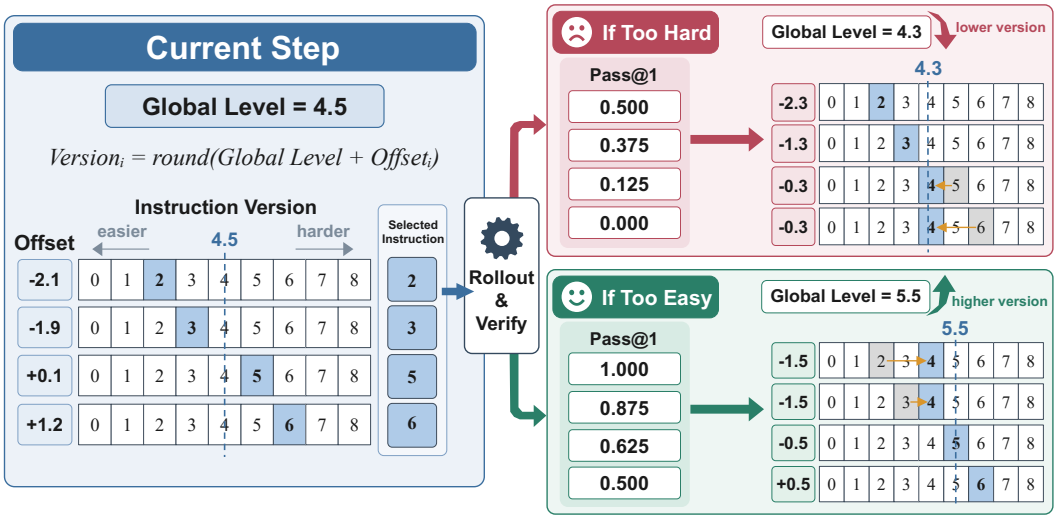


Figure 3: Overview of SPIRAL. **Left:** each row is one instruction's difficulty spectrum; the shaded cell is the version posed this step, chosen by one level shared across the batch plus that instruction's own offset. **Right:** the pass rates that come back move the level—either down (too hard) or up (too easy)—and re-anchor the offsets; the arrows mark which versions then shift.

zero-gradient samples. We call $[\tau^-, \tau^+] = [0.3, 0.7]$ the **golden zone**. The pass rate is thus a probe—free every step, from training itself rather than from any difficulty annotation—of whether the version currently posed matches the model, and SPIRAL simply keeps as many instructions inside the zone as it can.

Two cursors, moved by the probe. The first is global: a cursor shared by every instruction, on the same scale as version indices and estimated online, tracking the model's overall command of this data. We call it the **model ability level** L; it is the *Global Level* of Figure 3. The second is per-instruction: an **offset** δ_q, that instruction's bias relative to L. Together they select the version posed this step; the batch mean of a signed squared deviation d about a center c then moves both:

$$v_q = \mathrm{clip}\big(\mathrm{round}(L + \delta_q),\, 0,\, T_q\big), \quad d(\rho) = (\rho - c)^2\, \mathbb{1}[\rho > \tau^+] - (c - \rho)^2\, \mathbb{1}[\rho < \tau^-], \tag{3}$$

$$e_t = \frac{1}{|\mathcal{B}_t|} \sum_{q \in \mathcal{B}_t} d(\rho_q), \qquad L \leftarrow \max\big(0,\, L + \eta\, e_t\big), \qquad \delta_q \leftarrow v_q^* - L, \tag{4}$$

$$v_q^* = \mathrm{clip}\big(v_q + s(\rho_q),\, 0,\, T_q\big), \qquad s(\rho) = \mathbb{1}[\rho > \tau^+] + \mathbb{1}[\rho = 1] - \mathbb{1}[\rho < \tau^-] - \mathbb{1}[\rho = 0], \tag{5}$$

with $\mathcal{B}_t$ denoting the step-t batch, the target level v_q^* of equation 5 is where the instruction *should* have been posed, judged after the fact: s moves it up one level if the group came out too easy and down one if it came out too hard, leaves it where it is inside the golden zone, and applies an additional level of adjustment at $\rho_q \in \{0, 1\}$, where all m rollouts agreeing is the clearest signal the version is mismatched. Because δ_q is re-anchored against the new L each time, repeated visits do not compound it. §C.2 writes s out band by band and bounds the extreme steps; §C.3 gives the full step as pseudocode, fixing the order in which the version is selected and L and δ_q are updated. The two cursors track different things: L the global position along the spectra, δ_q the difficulty differences *between* instructions.

An asymmetric center biases training slightly hard. Training is stationary when $\mathbb{E}[d(\rho)] = 0$. Since d vanishes inside the golden zone this does not pin the steady state to one pass rate: it only requires the outward pulls on the two sides to cancel, and c sets their relative magnitude—at the extremes the ratio is $(1-c)^2/c^2$ (§C.4 derives this ratio and the steady state it implies). The midpoint $c = 0.5$ makes the two sides symmetric and settles training in the middle of the zone. We deliberately set $c = 0.3$, the zone's *lower* edge: the too-easy side is amplified and the too-hard side damped, so L carries a persistent upward pull that only a far larger share of too-hard groups can offset, and the steady state is pressed toward the harder end of the zone. The model therefore trains on data just beyond its current ability rather than at medium difficulty. The rule is self-limiting—d vanishes inside the zone, so the pull switches off there—and near-free: v_q is chosen *before* rollout so the m

rollouts are reused rather than added, ρ_q is already a by-product of the reward, and equation 4 costs $O(|\mathcal{B}_t|)$ scalar operations with no extra generation or model call. §C.6 lists what we track instead of reward, which no longer measures progress here.

3 Experiment

3.1 Setup

We run RL on three bases—Qwen3-4B (Yang et al., 2025), Qwen3.5-4B and Qwen3.5-9B. Baselines are four corpora differing in how they verify—IFEval (Zhou et al., 2023) and IFBench (Pyatkin et al., 2025) purely by rule, VerInstruct (Peng et al., 2025) and RAIF (Qin et al., 2025) mixing rules with LLM judging. Against them we place three uses of *the same* 3,000 Cascade instructions: **Cascade** flattens all versions into independent samples, **-hard** keeps only the hardest, and **-adaptive** lets Spiral pick a suitable version at each step. We evaluate on five instruction-following benchmarks—IFEval, IFBench, ComplexBench (Wen et al., 2024), AdvancedIF (He et al., 2025) and CascadeEval, averaged as IF-Avg—and, as a check that instruction tuning does not cost general reasoning, two out-of-distribution (OOD) benchmarks, GPQA-Diamond (Rein et al., 2023) and AIME26, averaged as OOD-Avg; IF-Len is mean response length. Base models, the cold-start checkpoint RL starts from, all training and evaluation hyperparameters, the reward and evaluation judges, and what each benchmark measures and why we chose it are in §D.1. We also run an SFT comparison of four data recipes on the same three bases; its configuration is in §D.1 and its results in §D.2.

3.2 Main Results

Table 1: **Main RL results.** Every run starts from that base's cold-start model, and Δ is against it. The last three rows of each block pose the same instructions at a different version: Cascade a random one, **-hard** always the hardest, **-adaptive** the one Spiral picks for the model's current ability. Within each block **bold** marks the best value in a column and underline the second best, over the trained rows.

		In-distribution								Out-of-distribution		
Base	Data	IFEval	IFBench	Complex	AdvIF	CasEval	IF-Avg	Δ	IF-Len	GPQA	AIME26	OOD-Avg
Qwen3-4B	cold-start	69.78	29.67	47.22	11.29	6.80	32.95	–	1703	31.57	13.75	22.66
	IFEval	61.18	**48.33**	14.83	4.48	0.00	25.76	−7.19	224	31.06	0.00	15.53
	IFBench	76.16	47.67	30.09	9.01	2.40	33.07	+0.11	725	35.86	1.88	18.87
	VerInstruct	**76.71**	36.83	45.48	12.92	4.80	35.35	+2.40	1420	36.87	13.75	25.31
	RAIF	76.25	34.00	44.48	12.48	5.60	34.56	+1.61	1278	36.11	14.17	25.14
	Cascade	73.75	33.50	43.39	13.18	6.80	34.12	+1.17	802	35.86	11.88	23.87
	-hard	**76.71**	35.33	**46.13**	**14.54**	5.60	35.66	+2.71	1356	**38.89**	**15.21**	**27.05**
	-adaptive	**76.71**	35.17	44.70	13.62	**9.20**	**35.88**	**+2.93**	1254	37.63	14.37	26.00
Qwen3.5-4B	cold-start	64.33	28.67	33.74	15.91	3.60	29.25	–	2700	50.51	41.67	46.09
	IFEval	53.51	47.00	11.26	3.43	0.80	23.20	−6.05	109	36.62	24.58	30.60
	IFBench	76.25	**54.67**	24.13	8.22	1.20	32.89	+3.64	208	33.84	18.96	26.40
	VerInstruct	80.50	35.83	47.48	18.85	6.40	37.81	+8.56	1021	54.04	31.87	42.96
	RAIF	**83.46**	40.83	43.52	19.77	8.00	39.12	+9.87	308	54.55	36.25	45.40
	Cascade	77.73	38.67	41.17	16.43	13.20	37.44	+8.19	550	61.11	33.96	47.54
	-hard	82.44	43.33	**61.83**	25.53	19.20	46.47	+17.22	2814	60.61	37.50	49.06
	-adaptive	82.44	43.50	61.57	**25.92**	**22.00**	**47.09**	**+17.84**	2274	**62.12**	**42.71**	**52.42**
Qwen3.5-9B	cold-start	65.62	35.33	40.65	25.92	5.60	34.62	–	2757	63.13	41.04	52.09
	IFEval	56.93	**52.67**	21.83	8.92	0.80	28.23	−6.39	364	40.15	18.33	29.24
	IFBench	83.27	52.17	43.22	15.60	7.60	40.37	+5.75	387	39.90	25.21	32.56
	VerInstruct	81.33	39.33	52.30	25.53	7.60	41.22	+6.59	477	55.30	36.46	45.88
	RAIF	84.66	42.33	45.78	26.54	11.20	42.10	+7.48	350	63.89	39.17	51.53
	Cascade	80.04	41.00	47.43	27.42	26.40	44.46	+9.83	622	69.95	49.17	59.56
	-hard	84.94	51.50	**67.96**	33.92	27.20	53.10	+18.48	2647	67.93	48.12	58.03
	-adaptive	**85.67**	**52.67**	67.78	**36.78**	**32.00**	**54.98**	**+20.36**	2078	**70.96**	**52.50**	**61.73**

Table 1 reports RL. Cascade-adaptive takes the best IF-Avg on all three bases and the best OOD-Avg on both Qwen3.5 bases. Three readings follow. *Rule-only verification buys compliance at a steep out-of-distribution cost.* IFEval- and IFBench-trained models top IFBench—the best or joint-best score in every block—and collapse on everything structurally complex, losing up to 22.85 OOD-Avg points while IF-Len falls to a few hundred tokens against a cold start of 1,703–2,757. They overfit the verifier, not the task. *The three uses of one instruction set form a clean order*, Cascade <

CASCADE-hard < CASCADE-adaptive, at identical instructions and identical rollout counts. Flattening all versions is the worst of the three because a large share of the budget lands on versions the model has already mastered, which §3.4 measures directly. It separates far less on Qwen3-4B, whose unstable CoT format leaves every arm on that base a noisy reward (§D.1). *The gain is not bought with length*: against CASCADE-hard, SPIRAL cuts IF-Len by 8%, 19% and **21%** while scoring higher on all three bases; its *accuracy* lead scales with the base—four of five in-distribution benchmarks on Qwen3.5-9B, CASCADEEVAL alone on Qwen3-4B.

3.3 ABLATION ON SPIRAL

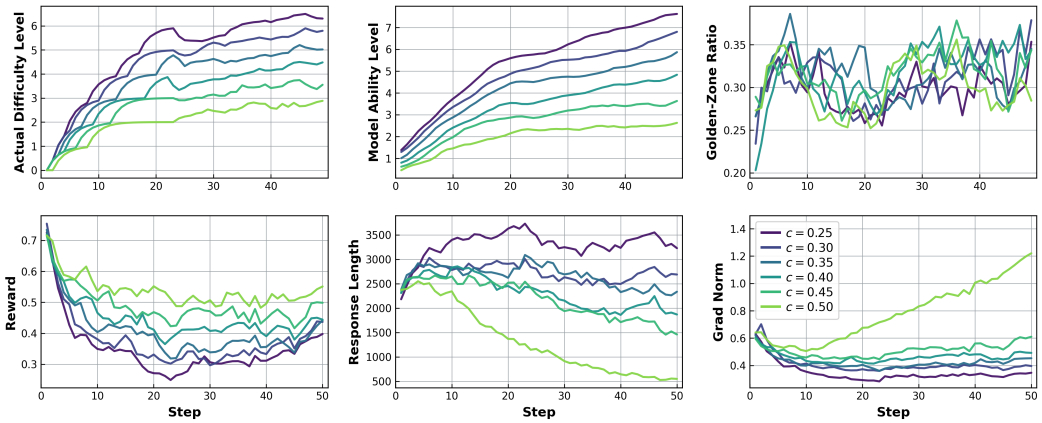
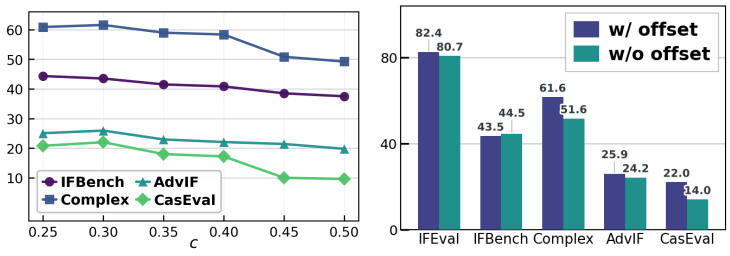


Figure 4: Six observables across the c sweep, one curve per c.



(a) Benchmarks across the c sweep. (b) Removing the offset.

Figure 5: Ablation of SPIRAL's two components.

The asymmetric center is load-bearing. Sweeping c from 0.50 to 0.25 moves IF-Avg from 39.78 to **47.09** at $c = 0.30$ and back down at 0.25. The optimum is therefore interior—"bias training slightly hard" is a design point, not a monotone push toward difficulty—and the symmetric choice $c = 0.50$ is *worse than not adapting at all* (CASCADE-hard, 46.47), so the asymmetry is a precondition rather than a tuning detail. Figure 4 explains both ends: in the actual difficulty level and the model ability level alike the six curves never cross, separating by c after the first few steps, which makes c a monotone knob on curriculum aggressiveness, while large c stalls with collapsing response length and a runaway gradient norm, and small c overloads. The golden-zone panel is the one place no ordering in c appears, so that statistic cannot be used alone to choose c.

Figure 5a adds what the average hides. Three of the four benchmarks plotted climb together from $c = 0.50$ to $c = 0.30$ and turn over at 0.25, CASCADEEVAL most steeply (9.60 to 22.00); IFBench alone keeps rising, and IFEval, flat across the whole sweep, is left out. The out-of-distribution average stays within 51.19–52.42 across the whole sweep and beats CASCADE-hard at every setting (§D.4): across this sweep the OOD benefit comes from selecting along the spectrum *at all*, while only the in-distribution scores depend on *where* one selects.

The per-instruction offset is what makes the curriculum per-instruction. Removing it costs −4.09 IF-Avg and −4.07 OOD-Avg, and the loss is concentrated: ComplexBench falls 9.96 and CASCADEEVAL 8.00, the two most complex benchmarks, while IFBench actually rises (Figure 5b). The reason is not that difficulty drops. Figure 6 shows the opposite—without the offset the selected version ends *higher* (about 6.8 against 5.8) at a model ability level that is nearly identical, so the

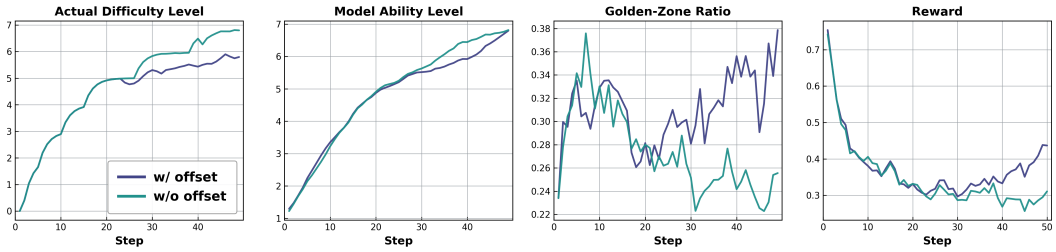
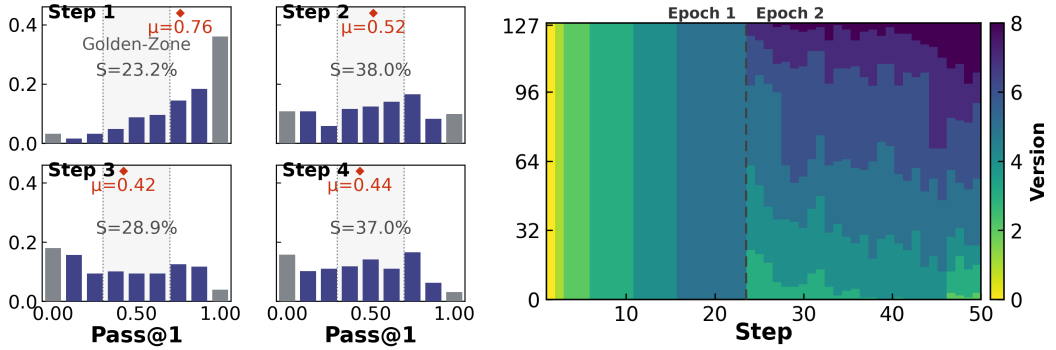


Figure 6: Training with and without the per-instruction offset.

whole difference is per-instruction spread—while the golden-zone share and the reward both end lower. One global level must be at once too easy for the easier instructions and too hard for the harder ones. What the offset learns, and how quickly, is in §D.4.

3.4 How Spiral Changes What Training Sees



(a) Pass-rate distribution over the first four steps; μ is the mean pass rate and S the golden-zone share. (b) Version posed to each of the 128 instructions sampled per RL step.

Figure 7: Spiral at work: the pass rate is pulled into the golden zone within the first few steps, and in the second epoch the offset pushes different instructions to different versions.

What decides whether a rollout group teaches anything is its pass rate, so that is what we measure (Table 2). Training on the hardest versions alone, 43.4% of groups are all-fail: close to half the budget on the most expensive data returns nothing. Flattening the versions does not fix this—it moves the waste to the other end, 15.4% all-pass—and Spiral is the only configuration below flattening at *both* ends (4.4% and 21.5%), with the highest golden-zone share, 31.6% against Cascade-hard's 19.1%. Note that minimizing the extremes is not itself the objective: Cascade-hard has the fewest all-pass groups of any configuration yet still scores below Spiral, and flattening—which wastes fewer groups in total—is nine points worse than Cascade-hard.

Table 2: Rollout-group shares during RL on Qwen3.5-4B.

Data	All-pass	All-fail	Golden
Cascade	15.4%	24.9%	22.1%
-hard	**1.7%**	43.4%	19.1%
-adaptive	4.4%	**21.5%**	**31.6%**
w/o offset	4.5%	27.0%	27.6%

Figure 7a shows the correction happening, and how fast. At step 1 the distribution is badly skewed toward high pass rates—every instruction is posed at its seed version, and the all-pass bin alone holds 36%. By step 2 it is nearly flat, by step 3 its center of mass has crossed the midpoint, and by step 4 it has settled inside the golden zone below its midpoint, the direction $c = 0.30$ predicts. Four steps, from an untuned start, with no difficulty annotation anywhere in the loop—the correction overshoots slightly on the way, since L is traveling from far too easy toward slightly hard and does not stop precisely, and §D.5 gives the per-step numbers.

Figure 7b then separates the two cursors by eye. Through the first epoch every column is a single color: no offset has taken effect yet, so the version is set by the model ability level alone, which lifts it steadily from 0 to about 5. From the epoch boundary the columns stratify across 5–6 levels and stay stratified rather than being smoothed away, while the whole band keeps drifting upward. That is

the offsets coming into effect, and the arithmetic agrees—3,000 instructions at batch size 128 is 23.4 steps per epoch, exactly where the transition falls. The two cursors are visually separable because they act on different timescales: the level as vertical drift, the offsets as within-column spread.

3.5 Further Experiments

Four further experiments are reported in the appendix for reasons of space. **Verification ablations** (§D.3) replace each baseline corpus's native verifier with the Agentic Verifier, and replace the check graph with a single end-to-end judgment, to separate what verification buys in accuracy from what it buys in unhackability. **Training efficiency** (§D.6) splits each RL step into rollout, verification and update, and shows that Spiral is also the cheapest of the three configurations at equal rollout count. **Judge robustness** (§D.7) re-scores every run of Table 1 with a second judge, from a different vendor and roughly an eighth the size, and finds no ordering changed. **CascadeEval as a benchmark** (§F) scores 48 frontier models at the seed and hardest versions and attributes their failures to node-level capability dimensions.

4 Related Work

Complex instruction following grows along a single axis: synthesis makes an instruction harder in the constraint-style sense—more constraints on one task—and verification follows suit, decomposing the response against a checklist fixed before that response exists (full citation trail in §H.1). Underneath both lies one assumption: everything the response must do can be settled before the model starts. Work on dependent sub-tasks does not disturb it—Chain-of-Instructions (Hayati et al., 2024), the closest prior work on dependent sub-tasks, feeds one instruction's output to the next, but chains existing tasks with no constraints and no branches, so the path through the chain is the same for every input—and where conditional execution does appear, the condition is settled before the model speaks: the model emits a routing DAG rather than executing one (Iyer et al., 2026), or the condition is read off the dialogue state, the environment, or the image. Cascade instead synthesizes *workflow-style complex instructions*, and in the hardest of them a branch condition is a predicate over the model's own intermediate output, so the path is a property of the rollout rather than of the prompt, and the GDV check graph makes a later check depend on that same output—the node-level judge is itself an agent-as-a-judge instance (Zhuge et al., 2024), so what is new is the graph, not the agent. Even prior work that pairs DAGs with judges (Verma et al., 2026) varies the workflow structure as a controlled variable, with no route decided at verification time; the difficulty-aware RLVR literature surrounding Spiral is deferred to §H.2.

5 Conclusion

Complex instruction following has been scaled along one axis, more constraints on one task, yet many instructions people write are workflows: interdependent sub-tasks under conditional branches, the hardest of which settle their route only as the response is produced. That structure breaks the post-training stack as a chain—missing from data, inexpressible for a checklist fixed before the response, unlearnable wherever every rollout of the hardest version fails, unmeasured by benchmarks—and one commitment breaks it at its source: expanding a seed by exactly one atomic unit per round yields the instruction, the check graph that verifies it, and the difficulty spectrum that schedules it as one artifact rather than three. Training on this data makes models follow complex instructions better than the strongest existing corpus does, and selecting along the spectrum improves them further while making their responses shorter and training cheaper. What we instantiate is one turn wide; the external non-determinism of multi-turn and agentic settings is a separate axis, left to future work.

AI Use Statement

In this work, we used generative AI tools to generate synthetic data sets and to clean and reformat datasets. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, support qualitative and thematic data analysis, or interpret results, and providing critical ingredients for

proving mathematical claims and assisting in the writing of proofs are not applicable to this work. Additionally, we used generative AI tools to edit this paper to improve readability, to search for information, and to identify and summarize existing literature. We have reviewed all AI-assisted work: synthesized data was kept only after passing the automated audits of §B, and AI-assisted text was checked by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Ethics Statement

We have read the ICLR Code of Ethics and this work adheres to it. All seed instructions come from publicly released research corpora, used under their own terms and screened for unsafe content by the pipeline of §B; we collected no new user data, and the 100-item gold set of §D.3 was labeled by the authors rather than by commissioned annotators. The models of §F were accessed through vendor APIs or public weights under the applicable terms of use. Cascade optimizes whether a response satisfies the requirements it was given and not the content of those requirements, so the content-safety mechanisms of a deployed system remain necessary; the authors declare no conflicts of interest.

Reproducibility Statement

The appendix documents both halves of the contribution at the level needed to rebuild them. §B gives atomic expansion end to end—seed selection, the operator inventory, check-graph construction and cross-validation, and the definition of every metric we report—while §E profiles what the pipeline produced and §G follows two instructions from seed to hardest version. §C states Spiral as pseudocode with every hyperparameter and its default, and §D.1 fixes the bases, the cold-start checkpoint, the random seed, the judges, the training framework and the protocol for each benchmark. After peer review we will release the synthesis pipeline, the Cascade corpus, CascadeEval with its check graphs, and the Spiral implementation.

References

Taha Aksu, Devamanyu Hazarika, Shikib Mehri, Seokhwan Kim, Dilek Hakkani-Tür, Yang Liu, and Mahdi Namazifar. CESAR: automatic induction of compositional instructions for multi-turn dialogs. *arXiv preprint arXiv:2311.17376*, 2023.

Kaikai An, Li Sheng, Ganqu Cui, Shuzheng Si, Ning Ding, Yu Cheng, and Baobao Chang. UltraIF: advancing instruction following from the wild. *arXiv preprint arXiv:2502.04153*, 2025.

Sanghwan Bae, Jiwoo Hong, Min Young Lee, Hanbyul Kim, JeongYeon Nam, and Donghyun Kwak. Online difficulty filtering for reasoning oriented reinforcement learning. *arXiv preprint arXiv:2504.03380*, 2025.

Yuelin Bai, Xinrun Du, Yiming Liang, Yonggang Jin, Junting Zhou, Ziqiang Liu, Feiteng Fang, Mingshan Chang, Tianyu Zheng, Xincheng Zhang, et al. COIG-CQIA: Quality is all you need for chinese instruction fine-tuning. *arXiv preprint arXiv:2403.18058*, 2024.

Andrei Baroian and Rutger Berger. Prompt replay: Speeding up GRPO with on-policy reuse of high-signal prompts. *arXiv preprint arXiv:2603.21177*, 2026.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, Gerald Shen, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Yoshi Suhara, Olivier Delalleau, Zijia Chen, Zhilin Wang, David Mosallanezhad, Adi Renduchintala, Haifeng Qian, Dima Rekesh, Fei Jia, Somshubra Majumdar, Vahid Noroozi, Wasi Uddin Ahmad, Sean Narenthiran, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Siddhartha Jain, Igor Gitman, Ivan Moshkov, Wei Du, Shubham Toshniwal, George Armstrong, Branislav Kisacanin, Matvei Novikov, Daria Gitman, Evelina Bakhturina, Prasoon Varshney, Makesh Narsimhan, Jane Polak Scowcroft, John Kamalu, Dan Su, Kezhi Kong, Markus Kliegl, Rabeeh Karimi Mahabadi, Ying Lin, Sanjeev Satheesh, Jupinder Parmar, Pritam Gundecha, Brandon Norick, Joseph Jennings, Shrimai Prabhumoye, Syeda Nahida

Akter, Mostofa Patwary, Abhinav Khattar, Deepak Narayanan, Roger Waleffe, Jimmy Zhang, Bor-Yiing Su, Guyue Huang, Terry Kong, Parth Chadha, Sahil Jain, Christine Harvey, Elad Segal, Jining Huang, Sergey Kashirsky, Robert McQueen, Izzy Putterman, George Lam, Arun Venkatesan, Sherry Wu, Vinh Nguyen, Manoj Kilaru, Andrew Wang, Anna Warno, Abhilash Somasamudramath, Sandip Bhaskar, Maka Dong, Nave Assaf, Shahar Mor, Omer Ullman Argov, Scot Junkin, Oleksandr Romanenko, Pedro Larroy, Monika Katariya, Marco Rovinelli, Viji Balas, Nicholas Edelman, Anahita Bhiwandiwalla, Muthu Subramaniam, Smita Ithape, Karthik Ramamoorthy, Yuting Wu, Suguna Varshini Velury, Omri Almog, Joyjit Daw, Denys Fridman, Erick Galinkin, Michael Evans, Shaona Ghosh, Katherine Luna, Leon Derczynski, Nikki Pope, Eileen Long, Seth Schneider, Guillermo Siman, Tomasz Grzegorzek, Pablo Ribalta, Monika Katariya, Chris Alexiuk, Joey Conway, Trisha Saar, Ann Guan, Krzysztof Pawelec, Shyamala Prayaga, Oleksii Kuchaiev, Boris Ginsburg, Oluwatobi Olabiyi, Kari Briski, Jonathan Cohen, Bryan Catanzaro, Jonah Alben, Yonatan Geifman, and Eric Chung. Llama-nemotron: Efficient reasoning models. *arXiv preprint arXiv:2505.00949*, 2025.

Justin Chih-Yao Chen, Archiki Prasad, Zaid Khan, Joykirat Singh, Runchu Tian, Elias Stengel-Eskin, and Mohit Bansal. Cog-DRIFT: exploration on adaptively reformulated instances enables learning from hard reasoning problems. *arXiv preprint arXiv:2604.04767*, 2026.

Peng Cui, Boyao Yang, and Jun Zhu. Learning-zone energy: Online data selection for efficient RL post-training. *arXiv preprint arXiv:2605.17003*, 2026.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1: incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

DeepSeek-AI et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

DeepSeek-AI et al. DeepSeek-V3.2: pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

DeepSeek-AI et al. DeepSeek-V4: towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. Self-play with execution feedback: Improving instruction-following capabilities of large language models. *arXiv preprint arXiv:2406.13542*, 2024.

Sarik Ghazarian, Abhinav Gullapalli, Swair Shah, Anurag Beniwal, Nanyun Peng, Narayanan Sadagopan, and Zhou Yu. TOD-ProcBench: benchmarking complex instruction-following in task-oriented dialogues. *arXiv preprint arXiv:2511.15976*, 2025.

GLM-5 Team. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Zhengkang Guo, Wenhao Liu, Mingchen Xie, Jingwen Xu, Zisu Huang, Muzhao Tian, Jianhan Xu, Yuanzhe Shen, Qi Qian, Muling Wu, Xiaohua Wang, Changze Lv, He-Da Wang, Hu Yao, Xiaoqing Zheng, and Xuanjing Huang. RECAST: expanding the boundaries of LLMs' complex instruction following with multi-constraint data. *arXiv preprint arXiv:2505.19030*, 2025.

Shirley Anugrah Hayati, Taehee Jung, Tristan Bodding-Long, Sudipta Kar, Abhinav Sethy, Joo-Kyung Kim, and Dongyeop Kang. Chain-of-Instructions: compositional instruction tuning on large language models. *arXiv preprint arXiv:2402.11532*, 2024.

Yun He, Wenzhe Li, Hejia Zhang, Songlin Li, Karishma Mandyam, Sopan Khosla, Yuanhao Xiong, Nanshu Wang, Xiaoliang Peng, Beibin Li, Shengjie Bi, Shishir G. Patil, Qi Qi, Shengyu Feng, Julian Katz-Samuels, Richard Yuanzhe Pang, Sujan Gonugondla, Hunter Lang, Yue Yu, Yundi Qian, Maryam Fazel-Zarandi, Licheng Yu, Amine Benhalloum, Hany Awadalla, and Manaal Faruqui. AdvancedIF: rubric-based benchmarking and reinforcement learning for advancing LLM instruction following. *arXiv preprint arXiv:2511.10507*, 2025.

Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, Xijun Gu, Peiyi Tu, Jiaxin Liu, Wenyu Chen, Yuzhuo Fu, Zhiting Fan, Yanmei Gu, Yuanyuan Wang, Zhengkai Yang, Jianguo Li, and Junbo Zhao. Reinforcement learning with rubric anchors. *arXiv preprint arXiv:2508.12790*, 2025.

Anand Iyer, Bhanu Khetharpal, Srinivas Upadhya, and Ramkumar Rajagopal. Generating workflow DAGs from natural language with non-reasoning LLMs. *arXiv preprint arXiv:2608.30250*, 2026.

Guochao Jiang, Wenfeng Feng, Guofeng Quan, Chuzhan Hao, Yuewei Zhang, Guohua Liu, and Hao Wang. VCRL: variance-based curriculum reinforcement learning for large language models. *arXiv preprint arXiv:2509.19803*, 2025.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. FollowBench: a multi-level fine-grained constraints following benchmark for large language models. *arXiv preprint arXiv:2310.20410*, 2023.

Kimi Team. Kimi K2: open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Kimi Team. Kimi K2.5: visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026a.

Kimi Team. Kimi K3: open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026b.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. *arXiv preprint arXiv:2309.06180*, 2023.

Yongchan Kwon, Shang Zhu, Federico Bianchi, Kaitlyn Zhou, and James Zou. ReasonIF: large reasoning models fail to follow instructions during reasoning. *arXiv preprint arXiv:2510.15211*, 2025.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Ming Li, Pei Chen, Chenguang Wang, Hongyu Zhao, Yijun Liang, Yupeng Hou, Fuxiao Liu, and Tianyi Zhou. Mosaic-IT: cost-free compositional data synthesis for instruction tuning. *arXiv preprint arXiv:2405.13326*, 2024.

Xiaomin Li, Zhou Yu, Zhiwei Zhang, Xupeng Chen, Ziji Zhang, Yingying Zhuang, Narayanan Sadagopan, and Anurag Beniwal. When thinking fails: The pitfalls of reasoning for instruction-following in LLMs. *arXiv preprint arXiv:2505.11423*, 2025a.

Ziheng Li, Zexu Sun, Jinman Zhao, Erxue Min, Yongcheng Zeng, Hui Wu, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Xu Chen, and Zhi-Hong Deng. Staying in the sweet spot: Responsive reasoning evolution via capability-adaptive hint scaffolding. *arXiv preprint arXiv:2509.06923*, 2025b.

Baohao Liao, Hanze Dong, Xinxing Xu, Christof Monz, and Jiang Bian. Self-hinting language models enhance reinforcement learning. *arXiv preprint arXiv:2602.03143*, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Zehao Liu, Yuanpu Cao, Jinghui Chen, and Vasant G. Honavar. Restoring the sweet spot: Pass-rate weighted self-distillation for LLM reasoning. *arXiv preprint arXiv:2605.27765*, 2026.

Sushant Mehta, Liudas Panavas, Suhaas Garre, and Edwin Chen. ComplexConstraints and beyond: Expert rubrics for RLVR. *arXiv preprint arXiv:2606.09118*, 2026.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActScore: fine-grained atomic evaluation of factual precision in long form text generation. *arXiv preprint arXiv:2305.14251*, 2023.

Maximilian Mordig, Andreas Opedal, Weiyang Liu, and Bernhard Schölkopf. Rethinking Easy-to-Hard: limits of curriculum learning in post-training for deductive reasoning. *arXiv preprint arXiv:2603.27226*, 2026.

Pixel Nomand, Elena Voss, Marcus Hale, and Sofia Reyes. Early verdicts, better budgets: Sequential adaptive rollout allocation for compute-efficient RLVR. *arXiv preprint arXiv:2607.26253*, 2026.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Shrey Pandit, Xuan-Phi Nguyen, Yifei Ming, Austin Xu, Jiayu Wang, Caiming Xiong, and Shafiq Joty. Synthesizing agentic data for web agents with progressive difficulty enhancement mechanisms. *arXiv preprint arXiv:2510.13913*, 2025.

Hao Peng, Yunjia Qi, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. VerIF: verification engineering for reinforcement learning in instruction following. *arXiv preprint arXiv:2506.09942*, 2025.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. *arXiv preprint arXiv:2507.02833*, 2025.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. Constraint back-translation improves complex instruction following of large language models. *arXiv preprint arXiv:2410.24175*, 2024.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. InFoBench: evaluating instruction following ability in large language models. *arXiv preprint arXiv:2401.03601*, 2024.

Yulei Qin, Gang Li, Zongyi Li, Zihan Xu, Yuchen Shi, Zhekai Lin, Xiao Cui, Ke Li, and Xing Sun. Incentivizing reasoning for advanced instruction-following of large language models. *arXiv preprint arXiv:2506.01413*, 2025.

Qwen et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: a graduate-level google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Tommy Sha, Skylar Zhai, and Siqi Zhao. ThinkPrior: zero-rollout difficulty priors for cold-start prompt selection in RLVR. *arXiv preprint arXiv:2609.09075*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Haozhan Shen, Shilin Yan, Hongwei Xue, Shuaiqi Lu, Xiaojun Tang, Guannan Zhang, Tiancheng Zhao, and Jianwei Yin. MM-CondChain: a programmatically verified benchmark for visually grounded deep compositional reasoning. *arXiv preprint arXiv:2603.12266*, 2026.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: a flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*, 2024.

Taiwei Shi, Yiyang Wu, Linxin Song, Tianyi Zhou, and Jieyu Zhao. Efficient reinforcement finetuning via adaptive curriculum learning. *arXiv preprint arXiv:2504.05520*, 2025.

Haoran Sun, Lixin Liu, Junjie Li, Fengyu Wang, Baohua Dong, Ran Lin, and Ruohui Huang. Conifer: Improving complex constrained instruction-following ability of large language models. *arXiv preprint arXiv:2404.02823*, 2024.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Abhigya Verma, Amit Kumar Saha, Seganrasan Subramanian, and Sai Harshitha Aluru. AgentJudgeBench: a multi-difficulty benchmark for evaluating LLM judges on agentic tool-calling. *arXiv preprint arXiv:2608.26623*, 2026.

Tao Wang, Shuo Li, Yan Sun, Dongsheng Ding, and Edgar Dobriban. Where to spend rollouts: Hit-utility optimal rollout allocation for group-based RLVR. *arXiv preprint arXiv:2605.07114*, 2026.

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. *arXiv preprint arXiv:2212.10560*, 2022.

Zhenting Wang, Guofeng Cui, Yu-Jhe Li, Kun Wan, and Wentian Zhao. DUMP: automated distribution-level curriculum learning for RL-based LLM post-training. *arXiv preprint arXiv:2504.09710*, 2025.

Sachin Gopal Wani, Ajay Dholakia, and David Ellison. The reasoning tax: Token economics of LLM reasoning across task types and deployment contexts. *arXiv preprint arXiv:2608.26235*, 2026.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2022.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, Yiming Liu, Jie Tang, Hongning Wang, and Minlie Huang. Benchmarking complex instruction-following with multiple constraints composition. *arXiv preprint arXiv:2407.03978*, 2024.

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. *arXiv preprint arXiv:2304.12244*, 2023.

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. *arXiv preprint arXiv:2406.08464*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Minglai Yang, Xinyu Guo, Utkarsh Tyagi, Mian Zhang, Razvan Dumitru, Sunjie Hou, Yunzhong He, Daniel Yue Zhang, and Ying Liu. Rubric dropout: A simple way to mitigate reward hacking in Rubric-as-Reward RL. *arXiv preprint arXiv:2608.11669*, 2026.

Junjie Ye, Caishuang Huang, Zhuohan Chen, Wenjie Fu, Chenyuan Yang, Leyi Yang, Yilong Wu, Peng Wang, Meng Zhou, Xiaolong Yang, Tao Gui, Qi Zhang, Zhongchao Shi, Jianping Fan, and Xuanjing Huang. MulDimIF: a multi-dimensional constraint framework for evaluating and improving instruction following in large language models. *arXiv preprint arXiv:2505.07591*, 2025.

Junxi Yin, Haisen Luo, Zhenyu Li, Yihua Liu, Dan Liu, Zequn Li, and Xiaohang Xu. Pinpointing crucial steps: Attribution-based credit assignment for verifiable reinforcement learning. *arXiv preprint arXiv:2510.08899*, 2025.

Runyang You, Hongru Cai, Caiqi Zhang, Qiancheng Xu, Meng Liu, Tiezheng Yu, Yongqi Li, and Wenjie Li. Agent-as-a-Judge. *arXiv preprint arXiv:2601.05111*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: an open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Zhiqi Yu, Zhangquan Chen, Mengting Liu, Heye Zhang, and Liangqiong Qu. Unveiling implicit advantage symmetry: Why GRPO struggles with exploration and difficulty adaptation. *arXiv preprint arXiv:2602.05548*, 2026.

Shijie Zhang, Zheng Xiao, Shiyu Liu, Guohao Sun, Kevin Zhang, Xiang Guo, Rujun Guo, Shaoyu Liu, Wangxiao Zhao, and Guanjun Jiang. CLPO: curriculum learning meets policy optimization for LLM reasoning. *arXiv preprint arXiv:2509.25004*, 2025.

Tao Zhang, Chenglin Zhu, Yanjun Shen, Wenjing Luo, Yan Zhang, Hao Liang, Tao Zhang, Fan Yang, Mingan Lin, Yujing Qiao, Weipeng Chen, Bin Cui, Wentao Zhang, and Zenan Zhou. CFBench: a comprehensive constraints-following benchmark for LLMs. *arXiv preprint arXiv:2408.01122*, 2024.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. WildChat: 1M ChatGPT interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Zhanke Zhou, Xiangyu Lu, Chentao Cao, Brando Miranda, Tongliang Liu, Bo Han, and Sanmi Koyejo. The easy, the hard, and the learnable: Confidence and difficulty-adaptive policy optimization for LLM reasoning. *arXiv preprint arXiv:2606.07950*, 2026.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-Judge: evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.

Appendix Contents

A Limitations **17**

B Data Synthesis and Benchmark Construction in Detail **17**

B.1 Seed Instructions 17

B.2 The Five Expansion Operators 18

B.3 Operator Scheduling and Per-Round Adjudication 19

B.4 Reverse Truncation 19

B.5 Check-Graph Construction and Auditing 19

B.6 Cross-Validation, Model Decoupling, and Outputs 20

B.7 From Pipeline to Benchmark: CascadeEval 21

B.8 Benchmark Metrics 22

C Spiral in Detail. **23**

C.1 Gradient Activity and the Golden Zone 23

C.2 Target Version Bands 24

C.3 Algorithm 24

C.4 The Asymmetric Center 25

C.5 Hyperparameters 26

C.6 What to Watch During Training 26

C.7 Applicability 26

D Further Experimental Results **27**

D.1 Training and Evaluation Configuration 27

D.2 Supervised Fine-Tuning Results 30

D.3 Ablation on Verification: Agentic Verifier and Check Graph 30

D.4 Ablation on Spiral's Two Components 33

D.5 Mechanism Analysis: How Spiral Changes What Training Sees 35

D.6 Training Efficiency: Spiral Shrinks the Verification Bottleneck 36

D.7 The Same Runs, Scored by a Second Judge 37

E Profile of the Synthesized Data **37**

E.1 Complexity Along the Spectrum 37

E.2 How Execution Flow Enters the Instructions 38

E.3 What the Checks Actually Test 39

E.4 The Hardest Versions, and What They Cover 40

F CascadeEval Benchmark Results **43**

F.1 Complexity Decay 44

F.2 Full Leaderboard 45

F.3 Failure Attribution 47

G Case Study **55**

G.1 Case EN-1: English, Scenario Simulation 55

G.2 Case ZH-1: Chinese, Poetry and Literary Writing 57

H Related Work in Detail **58**

H.1 Complex Instructions 58

H.2 Difficulty-Aware RL with Verifiable Rewards (RLVR) 59

A Limitations

Our study instantiates atomic expansion on single-turn, text-to-text instructions, but neither Cascade nor Spiral is inherently tied to that setting. Spiral in particular asks only that each sample carry an ordered sequence of versions—a structure that any corpus built by iterative augmentation provides—so it is not specific to instruction-following data; we validate it there because that is where a check graph supplies a reliable pass rate at every version. Extending both to multi-turn and tool-using settings, where part of the non-determinism comes from outside the response itself, is a natural next step and a promising direction for future work.

B Data Synthesis and Benchmark Construction in Detail

This appendix gives the procedures summarized in §2.1, in pipeline order: seed selection (§B.1), the five expansion operators (§B.2), how they are scheduled and adjudicated each round (§B.3), the three points at which a spectrum is truncated (§B.4), how check graphs are built and audited (§B.5), how responses are sampled and cross-validated (§B.6), the extra steps that turn the same pipeline into CascadeEval (§B.7), and the metrics the benchmark reports (§B.8). Appendix C then does the same for Spiral; Appendix D collects the experimental results the main text has no room for; Appendix E profiles the corpora that came out—keeping the training and evaluation splits separate throughout; Appendix F reports what 48 frontier models score on CascadeEval; and Appendix G walks two complete instructions from seed to hardest version.

B.1 Seed Instructions

Seeds are drawn from public user-conversation logs and human-written instruction data: WildChat (Zhao et al., 2024), COIG-CQIA (Bai et al., 2024) and Magpie (Xu et al., 2024) for Chinese, and WildChat, Magpie and No Robots for English. Selection proceeds in five steps.

Table 3: The 35 seed task types. The two groups are disjoint: training instructions are grown from the left column, CascadeEval instructions from the right.

Training seed task types (25)	**Evaluation seed task types (10)**
explanation_and_tutorial	poetry_and_literary_writing
advice_and_consultation	social_media_and_casual_writing
professional_document_writing	sentiment_and_opinion_analysis
planning_and_scheduling	reading_comprehension
story_and_fiction_writing	style_and_tone_transfer
open_ended_discussion	factual_question_answering
correspondence_and_speech_writing	commonsense_reasoning
copywriting_and_marketing	code_understanding_and_debugging
translation	brainstorming_and_ideation
text_summarization	scenario_simulation
paraphrasing_and_rewriting	
data_analysis_and_visualization	
causal_and_critical_analysis	
information_extraction	
roleplay_and_persona	
text_classification_and_labeling	
code_generation	
logical_reasoning	
format_conversion	
grammar_and_proofreading	
fact_verification	
text_completion_and_continuation	
humor_and_entertainment	
math_and_computation	
algorithm_and_system_design	

1. **Rule-based pre-filtering**: deduplication, length truncation, and removal of greetings, short acknowledgments, unsafe content, and garbled text.

2. **Task-type labeling**: a model assigns each instruction to one of 35 predefined task types (Table 3).
3. **Quality scoring** on five dimensions—informativeness and context self-sufficiency, expansion potential, task clarity, language quality, and topic safety. Failing any one dimension eliminates the instruction.
4. **Diversity sampling** under per-task-type quotas and source proportions.
5. **Two-stage post-filtering**: two different models review independently in sequence. The first stage permits one round of correction followed by re-inspection; the second accepts only instructions that pass outright.

The fifth step is concerned above all with *verifiability*: an instruction whose core task is too subjective, or on which human judges would not themselves agree, is discarded. One subjective flaw in a seed makes the entire complex instruction built on top of it unverifiable after K rounds of expansion, so the cost of admitting one is paid across all $K + 1$ versions.

The 35 task types are split into two disjoint groups, one for the training corpus and one for the benchmark, so that evaluation never falls on a task distribution seen in training.

B.2 The Five Expansion Operators

Each round selects one of five atomic expansion operators and applies it to the current version. An operator may act *anywhere* in the instruction—the new constraint, sub-task, or branch may be inserted at any position, not merely appended—and the three classes below grow the instruction along the three complexity axes.

Table 4: The nine constraint categories. Each category contains several atomic constraints; only examples are listed.

Category	What it controls	Examples
content	which elements must or must not appear	required keyword, forbidden topic
format	the data format or visual layout used	JSON output, Markdown table
template	fixed fragments or a structural skeleton	given opening, placeholders
counting	exact numeric control of a unit, rule-checkable	sentence count, paragraph count
language	the natural language and writing system	target language, letter case
syntactic	how words form sentences and sentences pattern	rhyme, acrostic
style	manner of expression and tone	formality, grammatical person
identity	who is speaking and who is addressed	persona, audience fit
ordering	arrangement of elements and non-duplication	chronological order, no repetition

Constraint axis: constraint expansion. One new constraint, compatible with those already present, is added per application. To keep the constraint pool diverse, we fix a categorized inventory in advance (Table 4); at each expansion the model first proposes the candidates compatible with the current instruction, and one is then sampled from among them. The pipeline maintains global usage counts per category and down-weights frequent ones, so that the corpus does not settle on the handful of categories that are easiest to generate. The inventory holds 46 constraints in 9 categories, each an atomic requirement that can be imposed on a response independently of the others, with no overlap or containment between them. Models may also invent constraints outside the inventory; these are recorded separately as a tenth, "custom" category in the data profile.

Sub-task axis: independent and dependent sub-task expansion. Both add one sub-task; they differ in whether the new sub-task takes an existing sub-task's output as its input. Independent sub-task expansion only increases how many things the response must do, making the instruction *wider*. Dependent sub-task expansion deepens the execution hierarchy, making it *taller*: the new sub-task consumes the output of an existing one, the dependency may target *any* existing sub-task rather than only the most recent, and it is expressed by referring to the object itself ("give this poem a title") rather than by meta-level anaphora ("based on what you just generated").

Execution-flow axis: deterministic and non-deterministic branch expansion. Both add one conditional branch; they differ in where the branch condition draws its information from. A deterministic branch depends only on information already given in the instruction, so the correct path can be determined before execution. A non-deterministic branch depends on the model's own output from a preceding sub-task; its generation pattern is fixed—carry out the preceding sub-task, perform one self-inspection of what was just produced, and route on the result—so it requires that the instruction already contain a preceding task to inspect. In both cases the two paths must not be mechanically symmetric, and the branch condition must be causally plausible in the scenario.

B.3 Operator Scheduling and Per-Round Adjudication

Operators are not drawn uniformly. The base sampler assigns preset weights to the three classes, subject to two scheduling limits: no operator may be used for more than three consecutive rounds, and no operator may be used more than $\lfloor K/2 \rfloor$ times on a chain. Together these prevent complexity from accumulating on one axis alone.

Each round is adjudicated immediately after it is produced, with four outcomes: *accept*, *retry*, *reject*, and *terminate*. A retry feeds the stated defect back to the same operator for regeneration; a reject discards the round's output and switches operator for the next round; repeated consecutive rejections terminate the chain. Defects that typically trigger a retry include: a branch condition that is trivial or always true; confusion between the two branch types or between the two sub-task types; a new constraint that conflicts with or duplicates an existing one; introduction of a requirement that cannot be objectively verified; and failure to smooth the connecting prose after new content is inserted.

This per-round check is what keeps atomic expansion on track in practice. "Exactly one minimal logical unit per round" is stated as a hard requirement in every operator's prompt, and the adjudication above is what stops a round that departs from it from entering the spectrum.

B.4 Reverse Truncation

Every quality check downstream of expansion acts on the spectrum as a whole, and all of them fail the same way: only the prefix that passes is kept, every version above it is discarded, and if the truncation point falls below a preset minimum version index the instruction is dropped entirely.

Reverse truncation occurs at three points, with slightly different triggers.

- **Instruction filtering** and **cross-validation** both roll back after the fact: starting from the hardest version, each version is examined in turn and the first one to pass becomes the final version. Instruction filtering judges each version along nine dimensions—executability, task-intent clarity, information completeness, logical self-consistency, constraint compatibility, constraint non-redundancy, language consistency, verifiability, and naturalness. Cross-validation judges a version by whether *any* sampled response passes its check graph (§B.6).
- **Check-graph auditing** truncates in place instead. Graphs are built in increasing version order; if the graph for some version still fails its audit after the permitted retries, that version has no graph and the spectrum ends at the preceding version (§B.5).

Atomicity is what makes truncation a precise operation. Adjacent versions differ by exactly one atomic unit, so rolling back one step is exactly undoing the last atomic edit, and the truncation point is the largest complexity at which the instruction was verified to be both well-formed and solvable. Because truncation points differ across instructions, spectra have unequal lengths, which reappears in Spiral as the per-instruction version bound T_q.

B.5 Check-Graph Construction and Auditing

The object being built. A check graph is a DAG $G = (V \cup \{\top, \bot\}, E)$ with a single source u_0 and two terminals $\top, \bot \notin V$, where $\top$ means the response passes and $\bot$ that it fails. Its nodes split disjointly into the two kinds of §2.1, $V = V_{\text{chk}} \cup V_{\text{rt}}$ with $V_{\text{chk}} \cap V_{\text{rt}} = \emptyset$, and every $u \in V$ carries exactly two out-edges, written u^+ and u^-, each pointing to another node or to a terminal. The two

kinds differ in what decides which edge is taken and in where the edges may lead:

$$u \in V_{\text{chk}}: \ \ \phi_u : \mathcal{Y} \to \{0,1\}, \ \ u^- = \bot; \qquad u \in V_{\text{rt}}: \ \ \beta_u : \mathcal{Y} \to \{0,1\}, \ \ u^+, u^- \neq \bot, \tag{6}$$

where $\mathcal{Y}$ is the space of responses. The asymmetry in equation 6 is the formal content of "an instruction-constraint node judges, a route node only routes": failing an instruction-constraint node sends the response straight to $\bot$, whereas a route node has no edge to $\bot$ at all and so can never by itself fail a response.

The **target path** of a response y is the node sequence $\mathcal{P}(y, G) = (u_0, u_1, \ldots, u_\ell)$ generated by

$$u_{j+1} = \begin{cases} u_j^+, & u_j \in V_{\text{chk}} \ \text{ or } \ \big(u_j \in V_{\text{rt}} \text{ and } \beta_{u_j}(y) = 1\big), \\ u_j^-, & u_j \in V_{\text{rt}} \ \text{ and } \ \beta_{u_j}(y) = 0, \end{cases} \qquad \text{stopping when } u_{j+1} = \top. \tag{7}$$

Note that equation 7 takes u_j^+ at an instruction-constraint node *whatever* $\phi_{u_j}(y)$ returns. The path is therefore a function of the route predicates alone—it is the route the response *ought* to follow—and it does not stop at the first failure. Because instruction-constraint nodes never take their $\bot$ edge here and route nodes have none, the sequence always terminates at $\top$, so $\mathcal{P}(y, G)$ is well defined for every y. We write $\mathcal{P}(y, G)$ for this sequence and, where only membership matters, for the set of nodes appearing in it. This is what makes it legitimate to score partial progress along the path, and what makes the verdict $\text{pass}(y, G) = \bigwedge_{u \in \mathcal{P}(y,G) \cap V_{\text{chk}}} \phi_u(y)$ of equation 2 a conjunction over an index set that depends on y but not on the model's success or failure. A static checklist is the degenerate case $V_{\text{rt}} = \emptyset$, in which $\mathcal{P}(y, G)$ is the same for every y.

Construction. Construction mirrors instruction expansion. A small graph, usually one to three nodes, is built for the seed. The round-k graph is then produced incrementally from the round-$(k-1)$ graph, the round-k instruction, the edit made this round, and the operator used—each operator class having its own update rule. Updates follow a *merge-first* rule: if a new check overlaps an existing node's subject, that node's problem is rewritten rather than a node added. This is why nodes do not correspond one-to-one to individual constraints or sub-tasks.

Each graph then passes two checks. A **structural check**, done by rules, verifies node numbering, pointer targets, reachability of the terminals, and acyclicity. A **semantic audit**, done by a model, covers four dimensions:

- **coverage**: every requirement in the instruction is covered by some node;
- **correctness**: each node's subject is unambiguous and independent of other nodes, and its problem faithfully matches the intent of the instruction;
- **verifiability**: each problem decides "correct or not" rather than "good or not";
- **topological faithfulness**: the branch structure of the graph matches the conditional structure of the instruction.

A graph that fails the audit is discarded and rebuilt with the failure reason injected as an extra hint, up to a retry cap; if it still fails, that round has no graph and the spectrum is truncated there.

Node decisions. Every node that still needs a decision at verification time is decided by the Agentic Verifier, an agent-as-a-judge with multi-turn reasoning and tool use; deterministic route nodes are not among them, their truth value having been settled when the graph was built. Rule-like problems obtain objective evidence through atomic tools—counting, keyword matching, JSON validation, and sandboxed code execution—while non-rule problems are decided by structured reasoning. No per-instruction rubric or verification program is written, so any requirement expressible in natural language can in principle be checked, and the decomposition granularity of the graph is set by what needs checking rather than by what a judge can be programmed to do.

B.6 Cross-Validation, Model Decoupling, and Outputs

The final check verifies two things at once. For each version we sample responses and judge each one node by node against the check graph: the graph judges whether a response passes, and a passing response in turn witnesses that the graph is satisfiable. If every response for a version fails, we have no witness that it is solvable at all, and reverse truncation is triggered.

Sampling has two stages. The first, **prompt-free sampling**, is the normal source of responses. The second, **hinted sampling**, injects the verbatim problem of a failed node as a hint to elicit responses beyond the model's own ceiling; its outputs pass an additional **leakage check**—a response may not quote or paraphrase the hint or refer to itself unnaturally—and are rejected as leakage otherwise. Once an instruction has one passing, non-leaking response, this stage is complete; several other models then contribute further prompt-free responses, so an instruction usually holds several passing candidates. Passing responses additionally undergo a **seven-dimension quality check** (safety, privacy and toxicity, branch consistency, factual accuracy, format completeness, clarity, and prompt leakage); those failing it are removed from the SFT pool.

Throughout, **judging is decoupled from generation**: the verifier model is always different from the model that produced the response under test, and the two seed post-filtering stages likewise use two different models.

Table 5: The three synthesis outputs, differing only in which versions they keep.

Output	Content	Use
SFT corpus	hardest, seed and one middle version + a passing response	SFT
RL corpus	hardest-version instruction + its check graph	standard RL
Adaptive corpus	full spectrum + every version's check graph	SPIRAL version selection

The pipeline emits three corpora from the same instructions, differing only in which versions they keep, as summarized in Table 5.

Why the SFT corpus is not hardest-only. The hardest version alone is the natural choice, and it is what the RL corpus takes, but two things argue against it for SFT. The first is volume: the hardest versions alone come to 8,199 pairs, between a quarter and a third of the 24,000–30,000 of the SFT baselines we compare against (Table 9), so a hardest-only corpus would confound the data recipe with data scale. The second is distribution: the hardest versions sit far from ordinary instructions, and a model trained on them alone does not pick up basic instruction following. We therefore keep the hardest version of every spectrum and add, per instruction, the seed version and one middle version drawn at random, which also gives the corpus a difficulty gradient rather than a single extreme; together these come to 24,864 pairs (Table 9). A response enters the corpus as soon as it passes its version's check graph and the seven-dimension quality check above, so an instruction carries one response per generator that produced a passing one.

B.7 FROM PIPELINE TO BENCHMARK: CASCADEEVAL

CASCADEEVAL is produced by the same pipeline, from seed filtering through cross-validation. It differs in four places, each meant to give the benchmark a property the training corpus need not have.

1. **Disjoint seed task types** (Table 3), so evaluation does not fall on a task distribution seen in training.
2. **Uniform spectrum length.** The training corpus allows reverse truncation to leave spectra of unequal length; the benchmark requires every instruction to complete all K rounds and discards those that do not, so each benchmark instruction has exactly $K + 1$ versions. This is the precondition for comparing complexity decay round by round: if instructions ended at different rounds, "average performance at round K" would not be a single quantity. The benchmark additionally keeps only graphs whose successful paths are all of equal length, so that pass difficulty is comparable across branch paths.
3. **Graph–instruction equivalence check.** Each check graph is verified to express exactly the requirements of its instruction—neither dropping a requirement nor adding one.
4. **Human inspection.** After synthesis and labeling, every instruction is checked by hand for language, scenario plausibility, and agreement between the check problems and the instruction text; failures are reworked or discarded.

The benchmark contains **125** instructions, **1,125** versions, and **6,421** check-graph nodes.

Table 6: The 13 capability dimensions annotated on CascadeEval check-graph nodes; the names are those of the released taxonomy, and the figures use them verbatim.

Dimension	What it decides
coverage	all required content elements are present
positioning	content is in the right place and in the right order
exact text	specified text appears verbatim
counting	discrete quantities meet the requirement
faithfulness	consistent with source material or facts
creation	required creative or structured artifact is produced
branch routing	the condition-appropriate branch is taken
formatting	markup format or structured presentation is correct
style	style, tone, person, and language match
exclusion	prohibited content is indeed absent
arithmetic	mathematical results are correct
reasoning	the required analysis and argument are done
rhetoric	required figures of speech or metrical forms are used

Capability dimensions. Every benchmark check-graph node is additionally labeled with one or more of 13 capability dimensions (Table 6), which the training corpus does not carry. The labeling covers all nodes of all rounds, is produced node by node by a model, has the branch-routing dimension filled in by rule for route nodes, and is verified by hand. These labels let the benchmark report not just pass rates but **failure attribution**: after locating the first failing node, the specific cause is diagnosed on the capability dimensions that node carries, and if the failure is a wrong branch, a second attribution is made at the nearest route node.

B.8 Benchmark Metrics

At evaluation each instruction is judged node by node along its check graph, by an Agentic Verifier kept separate from the model under test. Let $\mathcal{Q}$ be the benchmark instruction set ($|\mathcal{Q}| = 125$), and for instruction q at version $k \in \{0, \dots, K\}$ let $y_q^{(k)}$ be the response under test and $G_q^{(k)}$ the corresponding check graph. Write the instruction-constraint nodes lying on the target path of equation 7, in traversal order, as

$$\mathcal{P}\big(y_q^{(k)}, G_q^{(k)}\big) \cap V_{\text{chk}} = \big(u_{q,1}^{(k)},\, u_{q,2}^{(k)},\, \dots,\, u_{q,n_q^{(k)}}^{(k)}\big), \qquad n_q^{(k)} = \big|\mathcal{P}\big(y_q^{(k)}, G_q^{(k)}\big) \cap V_{\text{chk}}\big|, \tag{8}$$

and let $f_q^{(k)}$ be the position of the first one that fails, with the convention that an all-passing path gets $n_q^{(k)} + 1$:

$$f_q^{(k)} = \min\Big(\big\{\, j \le n_q^{(k)} \;:\; \phi_{u_{q,j}^{(k)}}\big(y_q^{(k)}\big) = 0 \,\big\} \cup \big\{\, n_q^{(k)} + 1 \,\big\}\Big). \tag{9}$$

The three reported metrics are then

$$\text{ILPR}(k) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathbb{1}\big[\text{pass}\big(y_q^{(k)}, G_q^{(k)}\big)\big] = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathbb{1}\big[f_q^{(k)} = n_q^{(k)} + 1\big], \tag{10}$$

$$\text{CR}(k) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \frac{f_q^{(k)} - 1}{n_q^{(k)}}, \qquad \text{CRI} = \frac{\text{ILPR}(K)}{\text{ILPR}(0)}. \tag{11}$$

Route nodes select paths and are never scored: they enter neither the conjunction of equation 10 nor the quotient of equation 11. Two conventions make them well defined. Every check graph contains at least one instruction-constraint node on every target path, so $n_q^{(k)} \ge 1$ in equation 8 and the quotient in equation 11 never divides by zero; and CRI presumes $\text{ILPR}(0) > 0$, which holds for every model we evaluate, the seed versions being by construction the easiest in the spectrum. By equation 9, each instruction contributes 1 exactly when every node on its path passes and 0 when the first one already fails, so $\text{CR}(k)$ measures how far into a workflow a model gets before it breaks; $\text{ILPR}(k)$ collapses "one step short" and "cannot do this at all" into the same value, which is why the two are read together. Empty responses are scored as failures. The map $k \mapsto \text{ILPR}(k)$ on $\{0, \dots, K\}$ is the *complexity decay curve*, and CRI is its endpoint ratio.

Two further quantities are used for analysis rather than for ranking models. Let $\mathcal{Z}$ be the 13 capability dimensions of Table 6 and $Z(u) \subseteq \mathcal{Z}$ the dimensions labeled on node u. At most two nodes of a

path are ever blamed for a failure: the first failing instruction-constraint node, and—when that failure is a wrong branch—the nearest route node preceding it. Write $\mathrm{att}(y,u) \subseteq Z(u)$ for the dimensions diagnosed as the cause of failure at such a node, and $\mathrm{att}(y,u) = \emptyset$ at every other node. A route node is never itself a failing node, since by equation 6 it has no edge to $\perp$; the second case is therefore an attribution made *about* a route node, not a verdict returned *by* one. Collecting the traversed nodes over a chosen set $\mathcal{K} \subseteq \{0,\dots,K\}$ of versions into $\mathcal{N}_{\mathcal{K}} = \big\{(q,k,u) \;:\; k \in \mathcal{K},\; u \in \mathcal{P}(y_q^{(k)}, G_q^{(k)})\big\}$—route nodes included, unlike in equation 10 and equation 11, since branch routing is labeled on them— the **per-capability failure rate** is

$$\mathrm{FR}(z) = \frac{\Big|\big\{(q,k) \;:\; k \in \mathcal{K},\; z \in \bigcup_{u \in \mathcal{P}(y_q^{(k)}, G_q^{(k)})} \mathrm{att}(y_q^{(k)}, u)\big\}\Big|}{\big|\{(q,k,u) \in \mathcal{N}_{\mathcal{K}} \;:\; z \in Z(u)\}\big|}, \qquad z \in \mathcal{Z}, \tag{12}$$

that is, how often a dimension is blamed against how often it was actually exercised. The two are on different units on purpose. The numerator counts *responses*—one per response, whichever and however many of its blamed nodes carry z—because a response fails once, not once per node; the denominator counts *node visits*, a node labeled with several dimensions counting once for each. FR is therefore a ranking across dimensions and not a probability, and a dimension carried by few traversed nodes can reach a high FR on a handful of failures. Every figure that reports FR in this paper takes $\mathcal{K} = \{K\}$, the hardest version alone; §F says so again where those figures appear, and §E reports how often each dimension is labeled in the first place.

The **per-task-type failure rate** is a different quantity, and deliberately not one defined over nodes: a task type labels the whole instruction, so there is no node-level denominator to form for it. Writing $\mathcal{W}$ for the ten evaluation seed task types of Table 3 and $\mathcal{Q}_w \subseteq \mathcal{Q}$ for the instructions of type w,

$$\mathrm{TFR}(w) = 1 - \frac{1}{|\mathcal{Q}_w|} \sum_{q \in \mathcal{Q}_w} \mathbb{1}\big[\mathrm{pass}\big(y_q^{(K)}, G_q^{(K)}\big)\big], \qquad w \in \mathcal{W}, \tag{13}$$

one minus the instruction-level pass rate at the hardest version, restricted to a single task type. The ten $|\mathcal{Q}_w|$ run from 9 to 19, so TFR is likewise read as a ranking rather than as a level, and it is not commensurable with FR: the two denominators count instructions and node visits respectively.

C Spiral in Detail

This appendix completes §2.2: the gradient-activity argument behind the golden zone (§C.1), the target-version bands (§C.2), the full algorithm and the ordering of its three operations (§C.3), the effect of the asymmetric center (§C.4), hyperparameters (§C.5), the quantities we track during training (§C.6), and what the method requires of the data (§C.7).

C.1 Gradient Activity and the Golden Zone

Under GRPO the m rollouts of one instruction form a group, and the i-th rollout's advantage is its within-group relative reward,

$$A_i = \frac{R_i - \bar{R}}{\mathrm{std}\big(\{R_j\}_{j=1}^m\big)}, \qquad \bar{R} = \frac{1}{m}\sum_{j=1}^m R_j, \qquad \mathrm{std}\big(\{R_j\}_{j=1}^m\big) = \sqrt{\frac{1}{m}\sum_{j=1}^m \big(R_j - \bar{R}\big)^2}, \tag{14}$$

taking the population standard deviation, as our implementation does. Take the reward to be the binary verdict, $R_i \in \{0,1\}$ (§D.1 gives the small length penalty the runs add on top, and what it does and does not change); then $\bar{R} = \rho_q$ and $\mathrm{std} = \sqrt{\rho_q(1-\rho_q)}$, and for $\rho_q \in (0,1)$ the advantages collapse to two values,

$$A_i = \begin{cases} \sqrt{\dfrac{1-\rho_q}{\rho_q}}, & R_i = 1, \\ -\sqrt{\dfrac{\rho_q}{1-\rho_q}}, & R_i = 0, \end{cases} \qquad \text{whence} \qquad \frac{1}{m}\sum_{i=1}^m \big|A_i\big| = 2\sqrt{\rho_q\big(1-\rho_q\big)}. \tag{15}$$

When the group is unanimous ($\rho_q \in \{0,1\}$) the denominator of equation 14 is zero; the convention—ours and that of standard implementations—is to set $A_i \equiv 0$, so the instruction contributes no gradient at that step and is a **zero-gradient sample**. Under the length penalty of equation 22 an all-pass

group is not *exactly* degenerate—its rewards differ by length—but what survives is a pull toward brevity, carrying no information about whether the instruction was followed, so it is wasted for the same reason. The right-hand identity of equation 15 extends that observation continuously: group activity peaks at $\rho_q = 1/2$ and decays monotonically to zero at both ends.

The exponent is worth stating precisely, because it depends on the variant. Without the standard-deviation normalization—the centered advantage $R_i - \bar{R}$ used by several GRPO derivatives—the same calculation gives

$$\frac{1}{m}\sum_{i=1}^{m}\big|R_i - \bar{R}\big| = 2\rho_q\big(1-\rho_q\big), \tag{16}$$

so activity scales as $\rho_q(1-\rho_q)$ in equation 16 and as $\sqrt{\rho_q(1-\rho_q)}$ under equation 14; normalizing divides out one factor of the standard deviation (Liu et al., 2026). Both vanish exactly at $\rho_q \in \{0,1\}$ and are maximized at $\rho_q = 1/2$, which is all SPIRAL relies on; we quote the normalized form throughout, since that is the advantage our runs actually use. We take $[\tau^-, \tau^+] = [0.3,\ 0.7]$ as the golden zone. At $m = 8$ the reachable pass rates are $\rho_q \in \{0, \frac{1}{8}, \ldots, 1\}$, so the zone contains exactly the three outcomes $\{\frac{3}{8}, \frac{4}{8}, \frac{5}{8}\}$—3, 4 or 5 rollouts out of 8—which is the discretization used in Table 7.

Three shares are tracked per step and reported in §D.5: the fraction of the batch inside the zone, and the two unanimous fractions, kept apart rather than summed,

$$\mathrm{GZ}_t = \frac{1}{|\mathcal{B}_t|}\sum_{q\in\mathcal{B}_t}\mathbb{1}[\rho_q \in [\tau^-,\tau^+]],\ \mathrm{AP}_t = \frac{1}{|\mathcal{B}_t|}\sum_{q\in\mathcal{B}_t}\mathbb{1}[\rho_q = 1],\ \mathrm{AF}_t = \frac{1}{|\mathcal{B}_t|}\sum_{q\in\mathcal{B}_t}\mathbb{1}[\rho_q = 0]. \tag{17}$$

GZ_t is not the complement of the other two—a group may sit outside the zone without being unanimous—which is why raising GZ_t and lowering AP_t and AF_t are reported as three separate effects. Both unanimous ends are wasted, but not equally and not for the same reason: an all-fail group has identically zero advantages and is exactly zero-gradient, whereas an all-pass group retains the length-penalty residual of equation 22, a gradient that says nothing about whether the instruction was followed. This is why we never report their sum. ρ_q is the binary pass rate in either case, so all three shares are unaffected by the penalty.

C.2 Target Version Bands

After the model ability level is updated, each sampled instruction's offset is re-anchored as $\delta_q \leftarrow v_q^* - L$, with L taking its new value, so that the offset always means "this instruction's target level relative to current global ability". The target level is given in five bands:

$$v_q^* = \mathrm{clip}\big(v_q + s(\rho_q),\ 0,\ T_q\big), \qquad s(\rho_q) = \begin{cases} +2, & \rho_q = 1 \quad \text{(all pass)},\\ +1, & \rho_q \in (\tau^+, 1),\\ 0, & \rho_q \in [\tau^-, \tau^+],\\ -1, & \rho_q \in (0, \tau^-),\\ -2, & \rho_q = 0 \quad \text{(all fail)}. \end{cases} \tag{18}$$

This is equation 5 of §2.2 written out band by band. The extremes move two levels rather than one because at $m = 8$ a unanimous group—all eight rollouts passing, or all eight failing—is the clearest signal that the version is mismatched, whereas the intermediate bands do not carry enough evidence to justify a two-level jump. The clip sits outside the bands, so it applies to all five: a one-level step from a boundary version can leave $[0, T_q]$ just as a two-level step can, and since v_q^* is consumed by $\delta_q \leftarrow v_q^* - L$ rather than posed directly, clipping only at selection time would not give the same offset. The offset is re-anchored from v_q^* each time the instruction is sampled rather than accumulated, so successive visits do not compound it.

C.3 Algorithm

Algorithm 1 states one full SPIRAL step. The ordering of its three operations matters. Version selection happens *before* rollout; the model ability level is updated after this step's rollouts complete; and the sample offsets are updated after L, re-anchored against its new value. Both updates take effect at the end of the step, but they are not felt on the same timescale: the new L enters the selection rule

Algorithm 1 SPIRAL data selection, layered on GRPO

Require: training set $\mathcal{D}$; a difficulty spectrum $\mathcal{S}(q)$ with a check graph per version; $\eta = 5$, $[\tau^-, \tau^+] = [0.3,\ 0.7]$, $c = 0.3$, $L_0 = 0$, $m = 8$

$L \leftarrow L_0$; $\delta_q \leftarrow 0$ for all q
for each training step t with batch $\mathcal{B}_t \subseteq \mathcal{D}$ **do**
  for $q \in \mathcal{B}_t$ **do** ▷ select the version, before rollout
    $v_q \leftarrow \text{clip}(\text{round}(L + \delta_q), 0, T_q)$
    rollout m responses on $q^{(v_q)}$ and set ρ_q to the group pass rate
  end for
  compute GRPO rewards and advantages, and apply one policy update
  $e_t \leftarrow \frac{1}{|\mathcal{B}_t|} \sum_{q \in \mathcal{B}_t} d(\rho_q)$; $L \leftarrow \max(0, L + \eta\, e_t)$ ▷ update the model ability level first
  $v_q^* \leftarrow \text{clip}(v_q + s(\rho_q), 0, T_q)$ for all $q \in \mathcal{B}_t$ ▷ target level, equation 5
  $\delta_q \leftarrow v_q^* - L$ for all $q \in \mathcal{B}_t$ ▷ re-anchor offsets to the new L
end for

for *every* instruction drawn at the next step, whereas a new δ_q is felt only the next time that particular instruction is sampled—one epoch later, since an epoch draws each instruction exactly once. That gap is what §3.4 reads off Figure 7b: within the first epoch no instruction has been drawn twice, so none has yet been posed under a non-zero offset, and the level alone moves the selected version.

C.4 THE ASYMMETRIC CENTER

The signed squared deviation of equation 3 is repeated here with its two sides named:

$$d(\rho) = \underbrace{(\rho - c)^2\, \mathbb{1}[\rho > \tau^+]}_{\text{too easy: push } L \text{ up}} - \underbrace{(c - \rho)^2\, \mathbb{1}[\rho < \tau^-]}_{\text{too hard: pull } L \text{ back}}, \qquad 0 \le c \le 1,\ \ 0 \le \tau^- \le \tau^+ \le 1. \tag{19}$$

The two indicators are mutually exclusive whenever $\tau^- \le \tau^+$, so d is well defined for any c, and it is identically zero on $[\tau^-, \tau^+]$. Placing c at the zone's lower edge rather than at its midpoint is what breaks the symmetry—the sweep of §D.4 also runs c above τ^-, up to the zone's midpoint, where the two sides balance exactly—and the effect is cleanest at the two extremes: the push at $\rho = 1$ relative to the pull-back at $\rho = 0$ is

$$\frac{|d(1)|}{|d(0)|} = \frac{(1 - c)^2}{c^2}, \tag{20}$$

which is $(0.7/0.3)^2 \approx 5.44$ at our $c = 0.3$ and exactly 1 at the symmetric $c = 0.5$.

Table 7: Signed squared deviations at $m = 8$, $\eta = 5$, for our $c = 0.3$ against the symmetric $c = 0.5$.

Passes	ρ	**Verdict**	$d(\rho)$	$\eta\, d(\rho)$	$\eta\, d(\rho)$ at $c = 0.5$
0	0.000	too hard	-0.0900	-0.450	-1.250
1	0.125	too hard	-0.0306	-0.153	-0.703
2	0.250	too hard	-0.0025	-0.013	-0.313
3–5	0.375–0.625	golden zone	0	0	0
6	0.750	too easy	$+0.2025$	$+1.013$	$+0.313$
7	0.875	too easy	$+0.3306$	$+1.653$	$+0.703$
8	1.000	too easy	$+0.4900$	$+2.450$	$+1.250$

Table 7 evaluates equation 19 at every outcome reachable with $m = 8$, alongside the symmetric choice $c = 0.5$. The ratio predicted by equation 20 is visible in the two rightmost numeric columns: $2.450/0.450 \approx 5.44$ at $c = 0.3$, against $1.250/1.250 = 1$ at $c = 0.5$.

The mechanism is self-limiting. Training is stationary when L stops drifting, that is when the batch mean e_t of equation 4 vanishes in expectation over the pass-rate distribution ρ induced by the current policy:

$$\mathbb{E}\big[d(\rho)\big] = 0 \quad \Longleftrightarrow \quad \underbrace{\mathbb{E}\big[(\rho - c)^2\, \mathbb{1}[\rho > \tau^+]\big]}_{\text{mass above the zone}} = \underbrace{\mathbb{E}\big[(c - \rho)^2\, \mathbb{1}[\rho < \tau^-]\big]}_{\text{mass below the zone}}. \tag{21}$$

Because d vanishes on the whole zone, equation 21 does not pin the steady state to one pass rate—many distributions satisfy it—and c decides which. Lowering c within the zone inflates the left-hand

side and deflates the right, so balance is reached only once a larger share of the batch sits on the too-hard side; the steady state moves toward the harder end of the zone. Whether that balance is reached, and at what level, is a property of the run rather than of equation 21: at our $c = 0.3$ the pass rates are pulled into the zone within four steps and L keeps climbing behind them (§D.5), whereas at the two largest values of the sweep L stops early and low—§D.4 records $c = 0.50$ sitting at level 2 from step 11 to 26.

C.5 Hyperparameters

Table 8: Spiral hyperparameters and their defaults.

Symbol	Default	Meaning
η	5.0	update rate of the model ability level
$[\tau^-, \tau^+]$	[0.3, 0.7]	bounds of the golden zone
c	0.3	center of the deviation; below the zone's midpoint, hence asymmetric
L_0	0.0	initial model ability level; a cold start begins at the easiest version
m	8	rollouts per instruction per step, and the denominator of ρ_q

Table 8 collects the five hyperparameters Spiral introduces. Only c was tuned; the rest were fixed a priori and never swept.

C.6 What to Watch During Training

Progress under difficulty-adaptive training cannot be read from the reward curve. What the two cursors hold near the golden zone is the *pass rate*, not the reward—the two differ by the length penalty of equation 22, and ρ_q is computed before it is applied—and the pass rate is held there only because difficulty rises to meet the model. The reward therefore tracks the difficulty being posed at least as much as the model's ability: on the main run it falls from 0.754 at step 1 to 0.297 by step 30 and recovers to 0.437, which says the level climbed faster than the policy through step 30 and that the policy then caught up. Real progress shows up instead as *rising difficulty at a roughly constant pass rate*. We track four quantities: the **mean selected version** in the step (the primary progress indicator, replacing reward; it should rise); the **model ability level** L (the continuous skeleton of that curve; it should rise); the **mismatch error** e_t (it should fluctuate around 0—a sustained departure means η is mismatched to the data); and the **golden-zone share** (rising quickly at first, then slowly, then levelling off, and above the non-adaptive baseline throughout). Reward, response length, and gradient norm are plotted alongside these, not as progress indicators but to tell whether training is healthy—for instance, an over-large center shows up as collapsing response length and an abnormally rising gradient norm.

C.7 Applicability

Spiral presupposes two-dimensional data: every instruction must carry a monotonically ordered sequence of versions. That is a strong condition, and Cascade's atomic expansion supplies it for free—the difficulty spectrum is a structural by-product of synthesis, requiring no difficulty annotation. To apply the method to one-dimensional data, where each sample has a single version, one would first have to construct a substitute for the spectrum. Three routes are available, differing in which end of the spectrum they build from.

1. **Weak to strong: augment each sample upward.** Apply an iterative augmentation operator to every sample and keep the intermediate states, so the original sample is the easiest version and each round adds difficulty. This is Cascade's route, and nothing about it is specific to instruction following: any augmentation that is monotone and leaves the task verifiable will serve.
2. **Strong to weak: hint each sample downward.** Take the original sample as the hardest version and manufacture easier ones by revealing progressively more of the solution—prefixes of a ground-truth answer, intermediate results, a partial plan. Hint-injection methods (Li et al., 2025b; Liao et al., 2026) already vary such a quantity continuously, which is a difficulty spectrum indexed from the hard end; they use it to rescue one sample at a time rather than to build a fixed ladder to select along, but the construction is the same.

3. **Across samples: cluster, then rank.** Group related samples and order each group strictly by difficulty, using within-group rank as the version index.

The first two build a spectrum inside one sample and so keep the property that every version poses the same underlying question; the third only approximates it, since the versions are different questions of similar kind. Whichever route is taken, what SPIRAL itself requires is narrow: the versions must be ordered, and each must carry a reward signal of its own—here the per-version check graph, which is what keeps the pass rate comparable as the cursor moves along the spectrum.

D FURTHER EXPERIMENTAL RESULTS

This appendix supports §3. §D.1 gives the configurations; §D.2 the SFT comparison; §D.3 the evidence behind the verification ablation; §D.4 the two SPIRAL component ablations; §D.5 the mechanism analysis showing how SPIRAL changes what training sees; §D.6 the wall-clock cost of training, split by phase; and §D.7 the same runs re-scored by a second judge.

D.1 TRAINING AND EVALUATION CONFIGURATION

Table 9: Training corpora. The last three rows come from the same 3,000 instructions and differ only in which versions are retained.

Corpus	Stage	Size	Note
Crab	SFT	24,336	baseline
Recast	SFT	29,939	baseline
Nemotron	SFT	30,000	baseline
CASCADE	SFT	24,864	ours; the seed, one middle and the hardest version of each instruction (§B.6)
cold-start	SFT	18,998	starting point for RL
IFEval	RL	14,973	baseline, purely rule-verified
IFBench	RL	95,373	baseline, purely rule-verified
RAIF	RL	13,305	baseline, rules mixed with LLM judging
VerInstruct	RL	27,498	baseline, rules mixed with LLM judging
CASCADE	RL	25,584	ours, all versions flattened into independent samples
CASCADE-hard	RL	3,000	ours, hardest version of each instruction only
CASCADE-adaptive	RL	3,000	ours, full spectrum, version chosen by SPIRAL

Cold start. RL begins from a cold-start checkpoint rather than from the base model, obtained by SFT on 18,998 samples synthesized by DeepSeek-V4-Pro (DeepSeek-AI et al., 2026) and Qwen3-4B (Yang et al., 2025), with prompts drawn from random subsets of the four RL baseline corpora of Table 9—IFEval (Zhou et al., 2023), IFBench (Pyatkin et al., 2025), RAIF (Qin et al., 2025) and VerInstruct (Peng et al., 2025)—together with CASCADE, plus some general instruction data. Every RL configuration in §3 starts from this same checkpoint, so its composition is a shared constant rather than an advantage for any one corpus. The stage serves two purposes: it fixes a stable CoT output format, without which the reward signal is dominated by format noise, and it gives the model enough instruction-following ability that early RL exploration is not purely random.

Why Qwen3-4B separates so little. The first of those purposes is achieved only partly on Qwen3-4B, and that is what compresses its whole block in Table 1. Qwen3-4B-Base produces no CoT format at all—its zero-shot row in Table 12 is essentially zero on every benchmark—and the cold-start corpus, at 18,998 samples, is not large enough to install one reliably: inspecting the RL rollouts, a substantial share still fail to emit a well-formed chain of thought, so the verdict they receive reflects the format as much as the instruction. The reward signal on this base is therefore noisy for *every* arm, and the gaps between arms shrink accordingly—CASCADE-adaptive still comes out ahead, by $+2.93$ IF-Avg over the cold start against CASCADE's $+1.17$, but the margin between the three CASCADE arms is small. Qwen3.5-4B-Base, by contrast, already emits a stable CoT format before any of our training, so its RL runs carry almost no format noise and the same three arms separate by 8.19, 17.22 and 17.84 points. The contrast is a property of the bases, not of the data: the ordering of the three arms is the same on both.

SFT. Table 9 lists every corpus used at both stages. Maximum length 16,384, batch size 128, learning rate 2×10^{-5}, warmup ratio 0.03, cosine schedule, 6 epochs.

Table 10: RL hyperparameters, shared by all three bases. λ is the length-penalty coefficient of equation 22.

Item	Value	Item	Value
Advantage estimator	GRPO	Training steps	50
Batch size	128	Rollouts per instruction	8
Mini-batch	32	Rollout temperature / top-p	1.0 / 1.0
Learning rate	1×10^{-6}	Warmup ratio	0.10
Gradient clipping	1.0	KL penalty / KL loss	both off
Entropy coefficient	0	Random seed	42
Prompt / response cap	1,024 / 15,000	Length penalty λ	0.1
Reward judge	Qwen3.6-35B-A3B	Hardware	H200, 8 GPUs/node, 1–2 nodes

RL. One configuration is shared by all three bases (Table 10). Training steps and batch size are identical across every RL configuration, so all of them consume the same number of rollouts; their *cost* still differs, because harder versions carry longer instructions, longer responses and larger check graphs. §D.6 measures that difference. All RL runs are implemented in verl (Sheng et al., 2024), with vLLM (Kwon et al., 2023) serving rollout generation.

The length penalty, and where it leaves the binary reward. The verifier returns a binary verdict, but the reward the policy is optimized against is not binary. Writing $\text{pass}_i \in \{0, 1\}$ for the check-graph verdict, ℓ_i for the length of rollout i, and $\ell_{\min}, \ell_{\max}$ for the shortest and longest lengths *among the passing rollouts of the same group*, the reward actually used is

$$R_i = \text{pass}_i \cdot \left(1 - \lambda \, \tfrac{\ell_i - \ell_{\min}}{\ell_{\max} - \ell_{\min}}\right), \qquad \lambda = 0.1, \tag{22}$$

and $R_i = \text{pass}_i$ unpenalized whenever no rollout passes or $\ell_{\max} = \ell_{\min}$—which covers the group with no passing rollout, the group with exactly one, and the group whose passing rollouts are all the same length. So a passing rollout receives a reward in $[1 - \lambda, 1]$—the shortest of its group keeps 1—and a failing one receives exactly 0. This is deliberate: it discourages padding without ever making a failure preferable to a pass. It is applied identically in every run reported here, including the rule-verified IFEval and IFBench baselines, so no comparison in §3 is confounded by it. One caution about reading λ: its smallness does not by itself bound its effect. Inside an all-pass group, writing $z_i = (\ell_i - \ell_{\min})/(\ell_{\max} - \ell_{\min})$, the rewards are $R_i = 1 - \lambda z_i$, and the normalization in equation 14 divides by $\text{std}(R) = \lambda \, \text{std}(z)$, so λ cancels and $A_i = (\bar{z} - z_i)/\,\text{std}(z)$—a brevity gradient of order one, exactly the residual §C.1 describes. What rules the penalty out as the cause of the length collapse that §D.3 attributes to rule-verifier hacking is therefore not its magnitude but the control run reported there: with the corpus, the reward form and λ all held fixed and only the verifier swapped, IFEval-av does not collapse.

Two consequences follow. First, equation 22 means R_i is no longer confined to $\{0, 1\}$, so the closed forms of equation 15 and equation 16 are statements about the idealized binary reward, not exact descriptions of the running system. The all-fail case is unaffected: if every rollout fails, every reward is 0, the group is exactly zero-gradient, and that is the case SPIRAL is built to avoid. The all-pass case is where the idealization breaks—when every rollout passes but their lengths differ, the rewards differ too and the group contributes a gradient, one that points toward brevity rather than toward correctness. Second, and separately, SPIRAL's own cursor updates are *not* affected: the group pass rate ρ_q of equation 4 is computed from pass_i, before the penalty is applied, so the golden zone of equation 17, the band rule of equation 18 and the deviations of Table 7 all rest on a genuinely binary signal.

Evaluation benchmarks. The five in-distribution benchmarks were chosen to span the instruction-following axis from the purely mechanical to the structurally complex, so that a gain concentrated at one end is visible as such. **IFEval** (Zhou et al., 2023) is the standard programmatically verifiable suite: each task carrying a handful of constraints that a rule can check exactly, which makes it the cleanest measure of literal compliance and the easiest to overfit. **IFBench** (Pyatkin et al., 2025) keeps that format but replaces the constraint inventory with held-out verifiers, so it measures whether

constraint following generalizes beyond the constraints a model was trained on rather than compliance itself. **ComplexBench** (Wen et al., 2024) is the closest existing benchmark to our regime, because it is the one that models composition explicitly: every instruction is typed by how its requirements combine—And, Chain, Selection, or Selection over a Chain—so sequenced tasks and selected branches are already present, and the score is taken over that composition rather than over a flat list. What it does not contain is a condition the response itself decides: a Chain's order and a Selection's condition are both given by the instruction and its input, so the path through an item can be named before generation begins, which is exactly the line our execution flow crosses. **AdvancedIF** (He et al., 2025) raises difficulty with expert-written rubrics and requires every rubric item to hold, making it the hardest of the four on judgment quality. **CascadeEval** is the one benchmark here whose execution flow is not settled before generation begins: its branch conditions may turn on the response's own intermediate output, which no other benchmark here does. Out of distribution we use **GPQA-Diamond** (Rein et al., 2023) and **AIME26**, the 2026 American Invitational Mathematics Examination, both rule-scored and unrelated to instruction following, as a check that instruction-following training has not damaged reasoning—the failure mode that rule-only RL exhibits most clearly in Table 1.

Table 11: The seven evaluation benchmarks. ISR is each benchmark's own **instruction-level success rate**, not our ILPR.

Benchmark	Size	Judging	Metric
IFEval	541	rules (25 verifiable constraint types)	ISR
IFBench	300	rules (58 new constraint verifiers)	ISR
ComplexBench	1,150	LLM judge, scored over the constraint composition	ISR
AdvancedIF	1,138	rubric LLM judge, all items must hold	ISR
CascadeEval	125	check graph + Agentic Verifier	ILPR
GPQA-Diamond	198	rules (option match)	Pass@1
AIME26	30	rules (`\boxed{}` integer match)	Pass@1

Table 11 lists the seven evaluation benchmarks. Each benchmark is sampled twice and averaged—AIME26 16 times, because it has only 30 items—at a rollout temperature of 1.0, applied uniformly to all seven benchmarks and to all three bases, Qwen3 and Qwen3.5 alike. The three benchmarks needing an LLM judge (ComplexBench, AdvancedIF, CascadeEval) all use DeepSeek-V4-Flash. Four conventions matter for reading every results table in §3 and this appendix:

1. **The CascadeEval column is the hardest version only** (ILPR at round K), not an average over versions, and it is scored at $t = 1.0$ with two samples per instruction on every base. The frontier-model evaluation of §F differs on both counts—it draws a single sample at $t = 0.7$ and scores the seed version alongside the hardest—so its CascadeEval numbers and the ones in this appendix must not be mixed.
2. **The training-time and evaluation-time judges differ**: rewards during training come from Qwen3.6-35B-A3B, chosen for throughput, while all evaluation uses DeepSeek-V4-Flash.
3. **AdvancedIF is evaluated on its complex_if and multi_turn subsets only**, not on its system_prompt_steerability subset. That subset measures hierarchy-following when a system prompt conflicts with the user instruction, an alignment axis orthogonal to user-instruction following; our data and training setup contain no system prompt.
4. **Bold marks the best value in each column**, never the worst. In Tables 12 and 1 the comparison runs within a base block and excludes the reference row (*base* and *cold-start* respectively); in Tables 13, 14 and 16 it runs within each pair of rows, and in Table 15 down the whole table. IF-Len is never bolded, since neither direction is uniformly better. Ties are bolded on every row that attains them.

Aggregate scores. Write $s_b \in [0, 100]$ for the score a run obtains on benchmark b under the protocol of Table 11, and let $\mathcal{M}_{\text{id}} = \{\text{IFEval}, \text{IFBench}, \text{ComplexBench}, \text{AdvancedIF}, \text{CascadeEval}\}$ and $\mathcal{M}_{\text{ood}} = \{\text{GPQA-Diamond}, \text{AIME26}\}$. The two aggregates reported in every results table are unweighted means over those sets:

$$\text{IF-Avg} = \frac{1}{|\mathcal{M}_{\text{id}}|} \sum_{b \in \mathcal{M}_{\text{id}}} s_b, \qquad \text{OOD-Avg} = \frac{1}{|\mathcal{M}_{\text{ood}}|} \sum_{b \in \mathcal{M}_{\text{ood}}} s_b. \tag{23}$$

The column Δ = IF-Avg − IF-Avg^{ref} is always a difference of IF-Avg against the reference row named in the caption—for example the base row for SFT and the cold-start row for RL; the ablation tables of §D.3 and §D.4 each name their own. The means in equation 23 are unweighted on purpose: weighting by benchmark size would let ComplexBench and AdvancedIF, the two largest, dominate IF-Avg and would mask exactly the shift toward harder benchmarks that §3.2 reports. IF-Len is the mean response length in tokens over $\mathcal{M}_{\text{id}}$, measured with the tokenizer of the model under test, so it is comparable within a base model but not across bases.

D.2 Supervised Fine-Tuning Results

Table 12: **SFT.** Four data recipes; Δ is against that base's own row. Qwen3-4B's base row is near zero, so only the spread between its recipes is meaningful. Within each block **bold** marks the best value in a column and underline the second best, over the four recipes.

		In-distribution								Out-of-distribution		
Base	Data	IFEval	IFBench	Complex	AdvIF	CasEval	IF-Avg	Δ	IF-Len	GPQA	AIME26	OOD-Avg
Qwen3-4B	base	0.00	0.00	0.00	0.22	0.00	0.04	–	0	0.00	0.00	0.00
	Crab	36.60	11.00	22.30	4.57	2.40	15.37	+15.33	433	32.58	1.25	16.92
	Recast	**73.29**	21.00	38.52	4.88	4.80	28.50	+28.45	601	34.34	7.71	21.03
	Nemotron	59.15	24.17	38.39	13.97	3.60	27.86	+27.81	816	**38.38**	**13.12**	**25.75**
	Cascade	64.51	**27.50**	**48.35**	**14.37**	**13.20**	**33.59**	**+33.54**	2839	29.80	8.12	18.96
Qwen3.5-4B	base	72.83	33.17	47.61	17.05	10.40	36.21	–	2767	69.44	71.04	70.24
	Crab	45.19	16.33	29.96	9.62	0.80	20.38	−15.83	421	68.43	76.04	72.24
	Recast	76.16	30.33	55.09	19.64	15.20	39.28	+3.07	1619	69.19	**76.67**	72.93
	Nemotron	69.78	37.50	**57.13**	19.51	14.00	39.58	+3.37	2071	**71.46**	**76.67**	**74.07**
	Cascade	**78.47**	**41.83**	56.09	**26.10**	**22.00**	**44.90**	**+8.69**	3485	65.91	64.38	65.15
Qwen3.5-9B	base	82.26	39.00	58.17	24.03	21.60	45.01	–	2778	76.52	78.54	77.53
	Crab	67.47	29.00	53.52	19.60	10.00	35.92	−9.09	1692	73.74	79.58	76.66
	Recast	77.45	35.00	59.48	24.78	15.60	42.46	−2.55	1545	**78.54**	**80.83**	**79.69**
	Nemotron	80.87	42.00	62.83	27.07	24.40	47.43	+2.42	2201	75.51	**80.83**	78.17
	Cascade	**82.90**	**50.00**	**63.61**	**35.90**	**38.40**	**54.16**	**+9.15**	3223	71.97	75.62	73.80

Besides the RL comparison of §3.2, we run the same three bases through SFT on four data recipes—Crab (Qi et al., 2024), Recast (Guo et al., 2025), Nemotron (Bercovich et al., 2025) and Cascade—with the corpora of Table 9 and the hyperparameters given above. Table 12 reports the outcome. Cascade gives the best IF-Avg on all three bases, leading the runner-up by **+5.09**, **+5.31** and **+6.73** and winning 13 of 15 base-by-benchmark cells. *Where* the gain sits matters more than its size: it widens as the base gets stronger, so it is not compensation for a weak model, and it concentrates on the hardest benchmarks—CascadeEval on every base, AdvancedIF on the two Qwen3.5 bases—while the lead on single-constraint IFEval is small or negative. Gain that grows with benchmark complexity is what workflow-style data should produce. The cost is a drop in OOD reasoning on the Qwen3.5 bases, the largest among the four recipes. We do not test whether RL recovers it: every RL run in Table 1 starts from the shared cold-start checkpoint rather than from an SFT checkpoint, and there Cascade-adaptive takes the best OOD-Avg on both Qwen3.5 bases.

D.3 Ablation on Verification: Agentic Verifier and Check Graph

Table 13: **Verification ablations.** Each RL corpus's native verifier replaced by the Agentic Verifier; Δ is against that corpus's native row.

Group	Corpus	Verifier	IFEval	IFBench	Complex	AdvIF	CasEval	IF-Avg	Δ	IF-Len	GPQA	AIME26	OOD-Avg
Rule	IFEval	native	53.51	**47.00**	11.26	3.43	0.80	23.20	–	109	36.62	24.58	30.60
	IFEval	Agentic	**79.94**	38.50	**45.61**	**18.50**	**6.40**	**37.79**	**+14.59**	587	**52.27**	**35.42**	**43.85**
	IFBench	native	76.25	**54.67**	24.13	8.22	1.20	32.89	–	208	33.84	18.96	26.40
	IFBench	Agentic	**82.16**	40.00	**49.04**	**19.16**	**11.20**	**40.31**	**+7.42**	1756	**50.00**	**27.92**	**38.96**
Mix	VerInstruct	native	80.50	35.83	47.48	18.85	6.40	37.81	–	1021	**54.04**	**31.87**	**42.96**
	VerInstruct	Agentic	**82.90**	**37.83**	**49.78**	**20.91**	**7.20**	**39.72**	**+1.91**	1009	53.03	31.04	42.04
	RAIF	native	**83.46**	**40.83**	**43.52**	**19.77**	**8.00**	**39.12**	–	308	54.55	36.25	45.40
	RAIF	Agentic	81.70	37.33	42.78	17.09	6.00	36.98	−2.14	389	**54.80**	**41.88**	**48.34**

Two things are ablated here. The first is the *verifier*: each baseline corpus's native verifier is replaced by the Agentic Verifier, everything else held fixed (Table 13). The second is the *form* of verification:

Table 14: **Check-graph ablation.** Node-by-node judging against a single end-to-end judgment.

Corpus	Verifier	IFEval	IFBench	Complex	AdvIF	CasEval	IF-Avg	Δ	IF-Len	GPQA	AIME26	OOD-Avg
CASCADE	node-by-node	77.73	**38.67**	41.17	**16.43**	**13.20**	**37.44**	–	550	**61.11**	**33.96**	**47.54**
CASCADE	end-to-end	**78.00**	38.33	**42.87**	16.21	10.40	37.16	−0.28	899	56.57	33.54	45.06

node-by-node judging along a check graph (`CKG`) against a single end-to-end judgment (`E2E`) of the whole response (Table 14). Both are run on Qwen3.5-4B.

Verification must be unhackable, not merely accurate. Table 13 splits the baselines in two. Replacing a corpus's native verifier with the Agentic Verifier is transformative where verification was purely rule-based (+14.59 and +7.42 IF-Avg, +13.25 and +12.56 OOD-Avg) and inert where the native verifier already asked an LLM whether the task was done (within three points, in both directions). What it supplies is therefore not accuracy but unhackability—and the rule-verified runs' collapse is also what explains their out-of-distribution damage in Table 1, since a policy that learns to emit stubs takes the habit into reasoning.

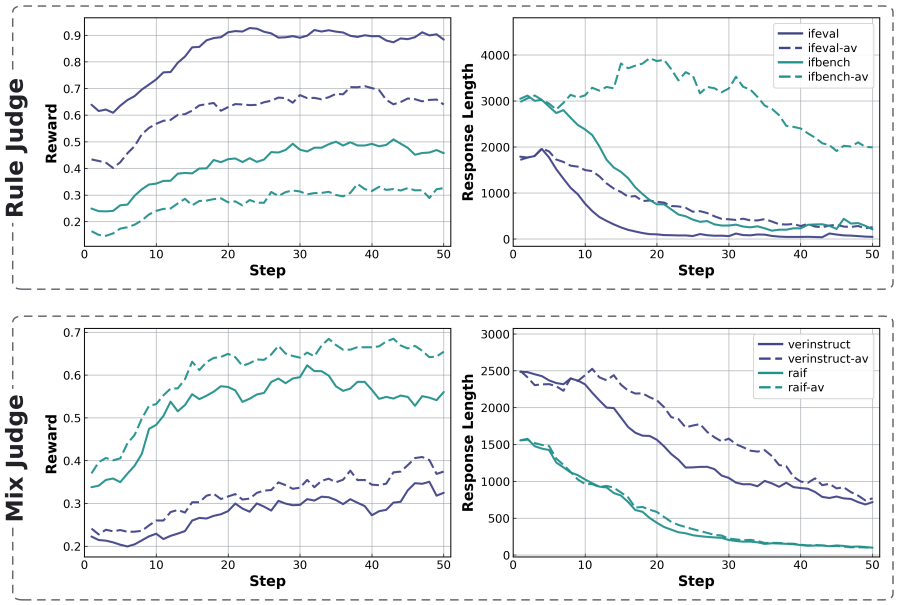


Figure 8: Native verifier (solid) against the Agentic Verifier (dashed, `-av`).

What reward hacking looks like. Figure 8 contrasts each corpus's native verifier with the Agentic Verifier over training. The decisive panel is the top-right one. Trained against IFEval's rules, response length falls off a cliff—about 1,960 tokens at step 4, 770 by step 10, 320 by step 14, and roughly 100 by step 20, after which it sits between 50 and 130 for the rest of training and ends near 50. IFBench shows the same shape more slowly, falling from about 3,000 to roughly 300 by step 30. Read the top-left panel alongside it: IFEval's reward climbs to about 0.93 by step 23 and stays above 0.87. At that moment the policy is scoring 0.93 while emitting a hundred-odd tokens. High reward with near-degenerate output is the signature of reward hacking, and it is invisible in the reward curve alone.

Neither `-av` run collapses. IFEval-av ends between 230 and 450 tokens; IFBench-av actually *rises* first, peaking near 3,900 at step 19 and ending near 2,000, never dropping below 1,900. Their rewards are lower in absolute terms (IFEval-av peaks near 0.71 and ends near 0.655), but a lower unhackable reward is worth more than a higher hackable one. The bottom row, the two corpora whose native verifiers already ask an LLM whether the task was done, shows no such pattern at all: all four reward curves rise smoothly and neither corpus loses its response length. The two verifiers do not coincide there—on RAIF the Agentic curve sits some 0.06–0.12 above the native one from step 20 onwards, and on VerInstruct some 0.03–0.05 above—but a persistent reward offset is not the failure mode at issue, and their scores still move by at most 2.14 IF-Avg points and 2.94 OOD-Avg points either way. One detail is worth noting: VerInstruct-av pushes length up to about 2,520 over the first eleven

steps—the highest point in the bottom row—before descending, so even in the mixed group the Agentic Verifier delays the onset of response compression.

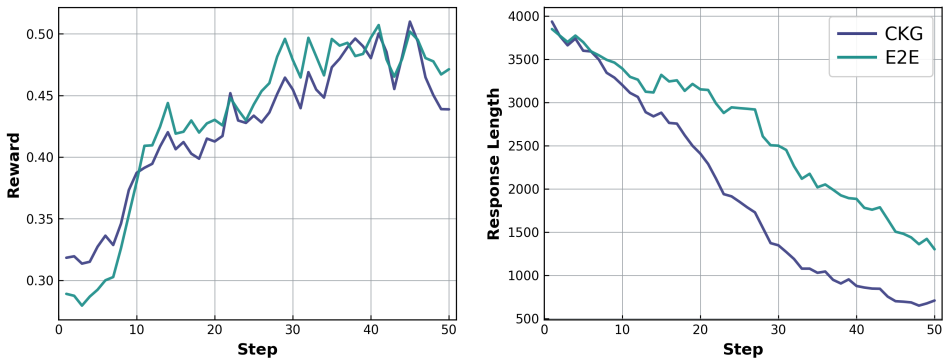


Figure 9: Node-by-node judging (CKG) against end-to-end judging (E2E).

What the check graph changes, and what it does not. Figure 9 locates the difference between CKG and a single E2E judgment, and it is not where one would expect. The *reward* curves are almost indistinguishable: both start around 0.28–0.32 and climb to 0.44–0.47, differing by less than 0.03 over most of training and never by more than 0.06, with CKG slightly ahead for the first ten steps, E2E ahead between steps 26 and 36, and E2E about 0.03 higher over the last five. That two such different procedures return the same reward level is consistent with the 0.28-point score gap in Table 14: as a scalar signal they are interchangeable.

The *response length* curves are not. Both begin near 3,900 tokens and track each other until step 8, after which they separate and keep separating: CKG falls to about 680 by step 48 and ends near 730, while E2E is still around 2,500 at step 30 and ends near 1,290; around step 30 the gap exceeds 1,100 tokens. CKG is simply a tighter constraint—every node must pass on its own, so padding cannot compensate for a missed requirement—and the policy converges to shorter, more precise responses. Read with the scores, CKG matches E2E on IF-Avg while producing roughly 60% of the output. This is the concrete form of the claim that the check graph buys expressiveness and attribution rather than reward accuracy.

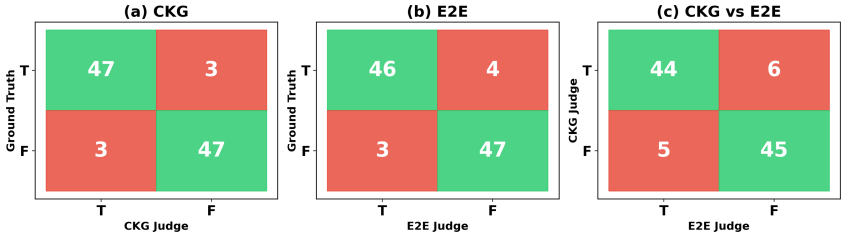


Figure 10: Confusion matrices against the human gold set: (a) CKG, (b) E2E, (c) the two against each other.

Agreement with human judgment. Because both verification forms are themselves models, the natural objection is that training rewards are a self-evaluation loop. Figure 10 answers it on a 100-item human gold set, 50 correct responses and 50 incorrect.

Both agree with human annotation on at least 93% of items—CKG 94, E2E 93—so neither is rubber-stamping the policy. The one-point gap between them is not the point and should not be read as one: on 100 items it is a single response, and a McNemar exact test on the eleven items where the two disagree gives $p = 1.00$. What the audit does establish is that node-by-node judging along a check graph is no *worse* than a monolithic judge at the thing a judge is for, while also returning which node failed—an output E2E cannot produce at all, and the one the rest of this paper uses. *How* the two differ is still informative: the false-accept rate is identical at 6% (three cases each), so what separates them is rejecting fewer correct responses, not being more lenient. The two forms agree with each other on 89 of 100 items; of the eleven disagreements, six are cases CKG passes and E2E fails and five the reverse, consistent with the one-item false-rejection gap.

One structural asymmetry should be recorded rather than buried: `CKG` accepts 100% (24/24) of correct English responses but 88.46% (23/26) of correct Chinese ones, whereas `E2E` is more even (91.67% / 92.31%); the false-accept rates run the other way (`CKG` 0% English against 10.34% Chinese, `E2E` 4.76% / 6.90%). `CKG` is thus less balanced across languages than `E2E` judging. Our corpus is bilingual, so this is a limitation of the current verifier prompts rather than of the check-graph formalism, but a multilingual claim would need to address it.

D.4 Ablation on Spiral's Two Components

Spiral has exactly two things to ablate: the asymmetric center c, which decides how hard the steady state is, and the per-instruction offset, which decides whether difficulty is allocated per instruction or globally. The model ability level is not among them: it is the only global quantity that selects a version at all, so removing it does not yield a degraded Spiral but no Spiral. Both are reported here in full; the main text quotes only their headline numbers.

Table 15: Sweeping the deviation center c, on Qwen3.5-4B. Δ is against the Cascade row. **Bold** marks the best value in a column and underline the second best.

Configuration	IFEval	IFBench	Complex	AdvIF	CasEval	IF-Avg	Δ	IF-Len	GPQA	AIME26	OOD-Avg
Cascade	77.73	38.67	41.17	16.43	13.20	37.44	–	550	61.11	33.96	47.54
Cascade-hard	82.44	43.33	**61.83**	25.53	19.20	46.47	+9.03	2814	60.61	37.50	49.06
$c = 0.50$	**82.81**	37.50	49.22	19.77	9.60	39.78	+2.34	541	58.84	**45.42**	52.13
$c = 0.45$	79.21	38.50	50.83	21.40	10.00	39.99	+2.55	1368	58.84	44.58	51.71
$c = 0.40$	80.31	40.83	58.35	22.06	17.20	43.75	+6.31	1678	58.84	43.54	51.19
$c = 0.35$	81.05	41.50	58.96	22.93	18.00	44.49	+7.05	2000	**62.37**	42.29	52.33
$c = 0.30$	82.44	43.50	61.57	**25.92**	**22.00**	**47.09**	**+9.65**	2274	62.12	42.71	**52.42**
$c = 0.25$	82.26	**44.33**	60.87	25.04	20.80	46.66	+9.22	2502	61.87	41.88	51.88

The asymmetric center. Beyond the interior optimum noted in §3.3, the sweep of Table 15 separates two things that are easy to conflate. OOD-Avg barely moves: it stays within 51.19–52.42 across all six settings and exceeds Cascade-hard's 49.06 at every one of them, while IF-Avg ranges over 39.78–47.09. So within this sweep the OOD benefit comes from selecting along the spectrum *at all*, and only the in-distribution benefit depends on *where* one selects—though the flat average masks two benchmarks that do not move together: GPQA rises as c falls while AIME26 declines. The four hardest benchmarks all improve monotonically from $c = 0.50$ down to $c = 0.30$, CascadeEval most of all (9.60 to 22.00, a factor of 2.3); at 0.25 three of the four fall back together with IF-Avg, IFBench alone continuing to rise. Running against the trend, AIME26 ends lower than it starts as c falls (45.42 to 41.88), so pushing difficulty too hard does crowd out mathematical reasoning—one reason the optimum is interior rather than at the boundary.

The sweep runs on Qwen3.5-4B, and its $c = 0.30$ row is the Cascade-adaptive row of Table 1. That value is not re-tuned afterwards: Qwen3-4B and Qwen3.5-9B, and with them every headline number of §3.2, use the same $c = 0.30$ carried over unchanged. Re-scoring the sweep with the second judge of §D.7 leaves the optimum where it is, and widens its lead over Cascade-hard from $+0.62$ to $+1.58$ IF-Avg.

Figure 4 in the main text shows why both ends fail. On the large-c side, difficulty stalls outright: $c = 0.50$ sits at level 2 from step 11 to 26 and $c = 0.45$ at level 3 from step 15 to 34; $c = 0.50$'s response length collapses from about 2,370 to roughly 550 tokens while its gradient norm rises anomalously from a trough near 0.50 to about 1.22 (every other c ends between 0.35 and 0.61). Its reward is the highest of the six, around 0.55, but that reward is earned by sitting on versions that are too easy—which is precisely why reward cannot be used to select c. On the small-c side, $c = 0.25$—the only setting in the sweep that sits below the zone's lower edge—has the lowest reward (bottoming near 0.245 at step 23), the longest responses (peaking near 3,700 and ending near 3,230), and the lowest, flattest gradient norm (0.29–0.35 throughout). Both ends stop teaching correctness, for opposite reasons: at large c what survives is the length penalty's gradient, at small c there is barely any gradient left.

The per-instruction offset. The loss (Table 16) is not spread evenly. CascadeEval falls 8.00 and ComplexBench 9.96—the two most complex benchmarks—while IFBench actually rises by 1.00.

Table 16: Removing the per-instruction offset, on Qwen3.5-4B.

Configuration	IFEval	IFBench	Complex	AdvIF	CasEval	IF-Avg	Δ	IF-Len	GPQA	AIME26	OOD-Avg
w/ offset	**82.44**	43.50	**61.57**	**25.92**	**22.00**	**47.09**	–	2274	**62.12**	**42.71**	**52.42**
w/o offset	80.68	**44.50**	51.61	24.21	14.00	43.00	−4.09	2081	57.32	39.38	48.35

Without the offset, SPIRAL degenerates into a single curriculum applied uniformly to every instruction, and its score falls back to roughly the $c = 0.40$ level (43.75).

What that degenerate case leaves behind is a familiar object. With $\delta_q \equiv 0$ SPIRAL keeps one global cursor and poses every instruction at round(L)—the control law the cross-sample schedulers of §H.2 run on, which likewise hold a single global difficulty target and match samples to it, swapping the question where this swaps the version. The comparison is therefore between control laws rather than with any published scheduler, since acting on the question pool would confound question selection with difficulty control. The law works on its own terms, lifting the golden-zone share from CASCADE-hard's 19.1% to 27.6% (Table 2). It also scores 43.00 against CASCADE-hard's 46.47, below the configuration that schedules nothing at all, and only the per-instruction offset turns the same cursor into a gain at 47.09. Difficulty scheduling is not automatically worth its cost on this data; it becomes worth it at the per-instruction granularity, which is the granularity a difficulty spectrum makes available.

Figure 6 in the main text carries the four observables that decide this ablation; the numbers behind it are these. The two difficulty curves track each other for 27 steps, then diverge, with the offset-free run reaching about 6.8 by the final step against 5.8 with offsets, while the model ability level is nearly identical in both (1.25 rising to 6.8)—the global level is the same, so the entire difference is per-instruction spread. The golden-zone share diverges in the opposite direction after step 30 (0.22–0.28 without against 0.28–0.38 with, ending at 0.256 against 0.379) and reward ends around 0.31 against 0.44. Higher difficulty, lower reward, emptier golden zone. A single model ability level has to compromise, and the compromise is simultaneously too easy for the easier instructions and too hard for the harder ones, emptying the golden zone from both ends at once.

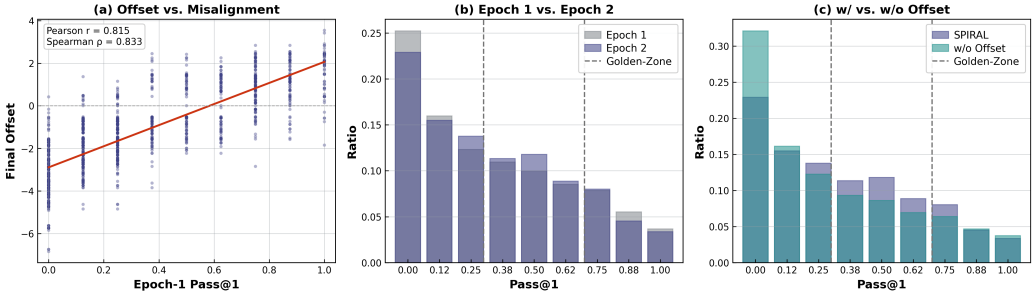


Figure 11: What the offset learns. (a) Final offset against that instruction's epoch-1 pass rate. (b) Epoch 1, no offset yet in effect, against epoch 2. (c) With and without offsets.

Figure 11 then asks what the offset actually learns, in three parts. *Does it learn difficulty, or accumulate noise?* Panel (a) says difficulty: the final offset correlates with the instruction's epoch-1 pass rate at Pearson $r = 0.815$ and Spearman $\rho_s = 0.833$, with the regression running from $(0.0, -2.9)$ to $(1.0, +2.1)$ and crossing zero near a pass rate of 0.58. The column at pass rate 0 lies almost entirely below zero, reaching -7; the column at pass rate 1 lies almost entirely above, reaching $+3.6$. The sign convention is the intended one—too easy pushes the offset up, too hard pushes it down—and noise would not produce a correlation this strong with a meaningful zero crossing. *How fast does it act?* Panel (b) says slowly: between epoch 1 (no offset yet in effect) and epoch 2, the all-fail bin falls only from 0.252 to 0.229, while the bins from 0.25 to 0.50 all rise a little—0.123 to 0.138, 0.109 to 0.113, and 0.100 to 0.118. The movement is toward the golden zone, but each epoch contributes little; the offset is a slow variable rather than a one-shot correction. *What is lost without it?* Panel (c) is the starkest: the all-fail bin reaches 0.321 without offsets against 0.229 with them, a gap of 0.09 and the largest single-bin difference in the figure, while the golden zone and everything right of it are uniformly lower. This is the direct source of the higher all-fail share in Table 2.

D.5 Mechanism Analysis: How Spiral Changes What Training Sees

§3.4 reports the three rollout-group shares (Table 2) and the two figures that read them. This section adds the per-step behavior behind those means. The shares are the GZ_t, AP_t and AF_t of equation 17, reported separately because only the all-fail extreme is genuinely zero-gradient: under the length penalty of equation 22 an all-pass group whose rollouts differ in length still contributes a gradient, one pointing toward brevity rather than toward correctness.

Table 2 must be read together with difficulty, and this is the point most easily missed. Flattened Cascade has fewer wasted groups in total than Cascade-hard yet scores nine points worse (§3.2), because its surviving samples sit on versions that are too easy. Minimizing the extremes is therefore not the objective: a configuration trained only on seed versions would look excellent on this metric and learn nothing. Spiral is the only configuration that comes in below flattening at both ends *and* raises difficulty, and the two must be checked together.

Figure 12 plots both unanimous ends, the solid all-fail curves being the exactly zero-gradient one. Each configuration wastes budget at a different end. Cascade-hard lives on the all-fail side: it begins at 0.67—two rollout groups in three teaching nothing—and falls to 0.28 as the model learns the hardest version, but stays the highest all-fail curve throughout, while its all-pass share never leaves 0–0.05. Flattened Cascade is the mirror image: its all-fail share barely moves (0.30 to 0.26), but its all-pass share climbs from 0.03 to 0.16 (peaking near 0.24) as uniform sampling lets the model master the low versions. Spiral keeps both ends low. It starts all-pass-heavy (0.36 at step 1, every instruction at its seed version) and all-fail-empty (0.03); within a few steps the level lifts difficulty, all-pass collapses to about 0.05 and stays there, and all-fail rises to a transient 0.29 at step 23 before settling to 0.15—the lowest all-fail area of the three. The transient is the level overshooting on its way up, not a standing cost.

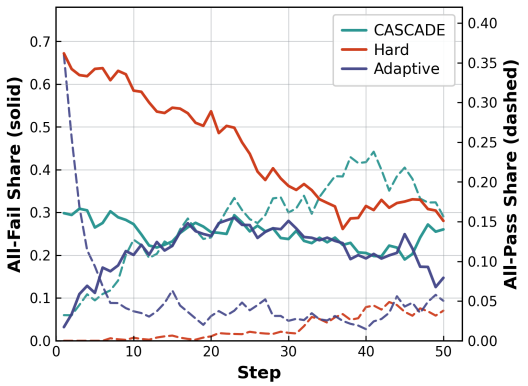


Figure 12: The two unanimous ends across training: all-fail on the left axis (solid), all-pass on the right (dashed), same color per configuration. The two are kept on separate axes and never added—only all-fail is exactly zero-gradient (§C.1).

The pass-rate distribution of Figure 7a fills in what the means hide. At step 1 the all-pass bin alone holds 36%, the mean pass rate is $\mu = 0.76$ and the golden-zone share only $S = 23.2\%$, because the model ability level starts at 0 and every instruction is posed at its seed version. By step 2 the distribution is nearly flat ($\mu = 0.52$, $S = 38.0\%$); by step 3 its center of mass has crossed the midpoint ($\mu = 0.42$, $S = 28.9\%$—the share dips as the level overshoots) and the all-pass bin has collapsed to 0.04; by step 4 it settles at $\mu = 0.44$, $S = 37.0\%$. Four steps to settle inside the golden zone, below its midpoint, is what the asymmetric center predicts. The correction slightly overshoots on the way—the all-fail bin rises to 0.158–0.180 at steps 3–4—because L is traveling from "far too easy" toward "slightly hard" and does not stop precisely.

Watching the level and the offsets act separately. Figure 7b is the most direct evidence that the model ability level and the per-instruction offset do different jobs. Through steps 1–23 every column is a single color: no offset has taken effect yet, so the version is set by the model ability level alone, which lifts it steadily from 0 to about 5. From step 24 the columns stratify—within a single step the selected versions span 5–6 levels, from about 2 at the bottom to about 8 at the top—and that spread persists rather than being smoothed away, while the whole band continues drifting upward until, after step 40, the top becomes solid version 8 and expands downward.

The arithmetic pins the interpretation. With 3,000 instructions at batch size 128, one epoch is about 23.4 steps, so the transition from uniform to stratified columns falls exactly on the epoch boundary. That is what one expects if offsets are determined by the previous epoch's observations, and it is hard to explain otherwise. In this figure the model ability level appears as vertical drift and the per-instruction offset as within-column spread; the two are visually separable because they act on different timescales.

D.6 Training Efficiency: Spiral Shrinks the Verification Bottleneck

Every RL configuration consumes the same number of rollouts, so a difference in wall-clock cost comes from what a rollout costs rather than how many are drawn. We split each step into three phases: *rollout* (vLLM sampling), *verify* (executing the check graph against the response) and *PPO* (reference log-probability pass, backward pass, weight synchronization). Rollout and verify run concurrently over the batch and the step waits for the last sample to finish, so both are taken from the critical-path sample's timers, which reproduces the measured step time to within 0.4% at the median over all 450 logged steps.

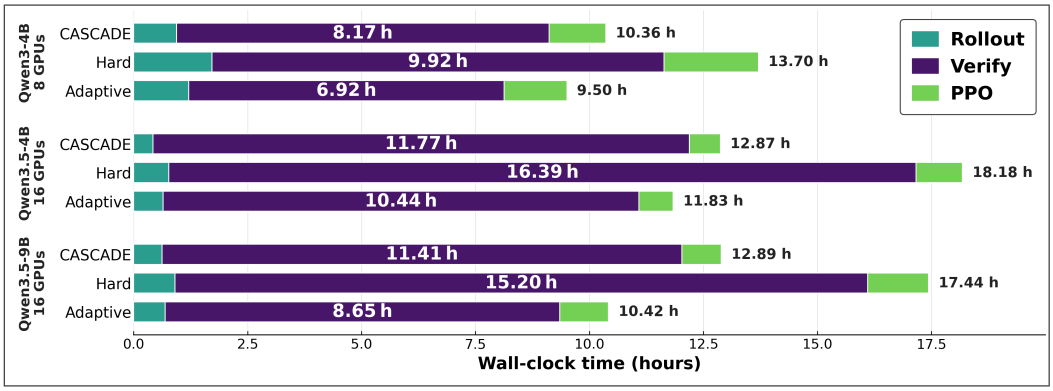


Figure 13: RL wall-clock time by phase; the label inside each bar is that run's verification time.

Verification dominates, and not incidentally. Figure 13 splits all nine runs by phase: verification takes 72–92% of every configuration, against 3–13% for rollout and 5–16% for PPO. The cause is architectural: a check graph is not evaluated by rule. Every constraint node is decided by its own multi-turn LLM-judge session against Qwen3.6-35B-A3B, and only deterministic route nodes—6.4% of the nodes the Cascade-hard arm poses—are settled without a model call. One response therefore triggers one judge session per node, so verification inherits a 35B model's latency rather than the throughput of the local rollout engine. This is the price of graph-decomposed verification, and it is what bounds how fast instruction-following RL can run.

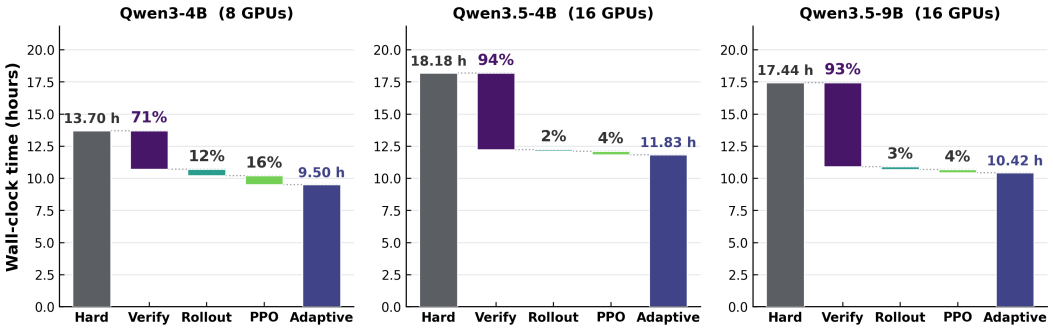


Figure 14: Where Cascade-adaptive's saving over Cascade-hard comes from; percentages are each phase's share of the saving.

Cascade-adaptive is the cheapest of the three, and the saving is verification's. At equal rollout count it runs 30–41% below Cascade-hard and 8–20% below flattened Cascade, while also scoring best (§3.2)—so the accuracy ordering of §3.2 is not bought with compute. Figure 14 attributes the saving over Cascade-hard: verification supplies 94% and 93% of the hours saved on the two Qwen3.5 bases—6.35 on Qwen3.5-4B (18.18 down to 11.83 h, 34.9%) and 7.02 on Qwen3.5-9B (17.44 down to 10.42 h, 40.3%)— with rollout and PPO together no more than 7%. The mechanism is check-graph size, which grows by roughly one node per expansion round, so posing a lower version buys fewer judge sessions. On Qwen3-4B the split is less extreme (71%/12%/16%) because that run used half the GPUs, which inflates the two phases that scale with device count while verification, served by a separate judge cluster, does not.

D.7 The Same Runs, Scored by a Second Judge

Table 17: IF-Avg for every RL run under both judges. **F** is DeepSeek-V4-Flash, used throughout the paper; **Q** is Qwen3.6-35B-A3B, the training-time reward judge. IFEval and IFBench are rule-scored and identical under both, so only ComplexBench, AdvancedIF and CascadeEval are re-judged.

	Qwen3-4B		Qwen3.5-4B		Qwen3.5-9B	
Data	F	Q	F	Q	F	Q
cold-start	32.95	32.16	29.25	29.20	34.62	34.79
IFEval	25.76	25.52	23.20	22.70	28.23	28.14
IFBench	33.07	32.31	32.89	32.84	40.37	40.22
VerInstruct	35.35	34.25	37.81	38.04	41.22	41.23
RAIF	34.56	33.65	39.12	38.81	42.10	42.34
Cascade	34.12	33.60	37.44	37.03	44.46	43.68
-hard	35.66	34.59	46.47	44.66	53.10	52.98
-adaptive	**35.88**	**34.67**	**47.09**	**46.24**	**54.98**	**54.64**

Three of the seven benchmarks are decided by an LLM judge, which makes the judge as much a setting of the measurement as the random seed is; Table 10 and §D.1 fix both. Every run reported here was scored twice under that setting—once by DeepSeek-V4-Flash, the evaluation judge, and once by Qwen3.6-35B-A3B, which served as the training-time reward judge—so Table 1 can be recomputed under a model from a different vendor and roughly an eighth the size without drawing a single additional sample.

Table 17 gives the result: no ordering changes. On all three bases Cascade $<$ Cascade-hard $<$ Cascade-adaptive still holds, Cascade-adaptive is still the best configuration, and the rule-verified corpora are still the worst. The margin over the strongest existing corpus on Qwen3.5-9B moves from $+11.00$ to $+10.64$; the Spiral margin over Cascade-hard moves from $+1.88$ to $+1.66$ there, from $+0.62$ to $+1.58$ on Qwen3.5-4B, and from $+0.22$ to $+0.08$ on Qwen3-4B, small at that end under either judge. Over the 63 runs the two judges have in common—every configuration CascadeEval was scored on by both—their CascadeEval scores correlate at Pearson $r = 0.984$ and Spearman $\rho = 0.977$, and Q is the stricter of the two by 2.27 points on average—an offset that moves the levels but, every comparison here being within a column, cancels out of the conclusions.

E Profile of the Synthesized Data

This section reports what the pipeline actually produced. Throughout we keep the two splits separate: the **RL training corpus** (3,000 instructions grown from the 25 training seed task types) and **CascadeEval** (125 instructions grown from the 10 evaluation seed task types).

E.1 Complexity Along the Spectrum

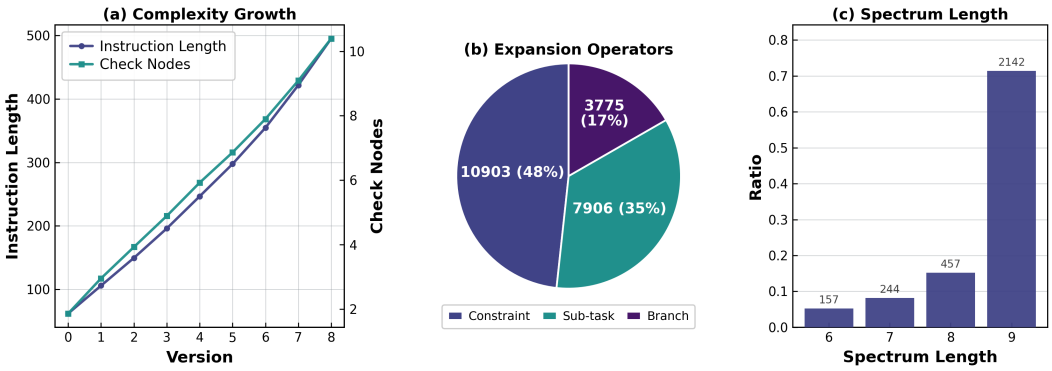


Figure 15: RL training corpus. (a) Instruction length and check-graph node count against version index; (b) share of each expansion operator; (c) spectrum lengths after reverse truncation.

Figures 15 and 16 track instruction length and check-graph node count against the version index. On the training corpus, instruction length rises from 62 to 495 tokens and check-graph nodes from 1.9 to

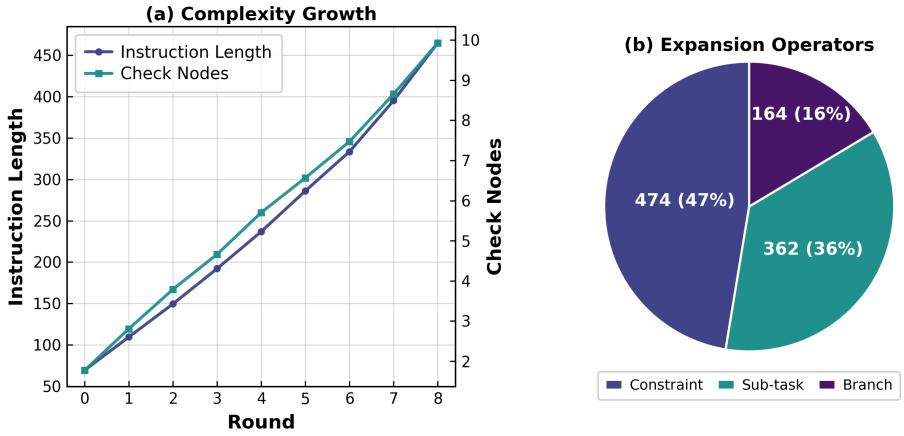


Figure 16: CascadeEval. (a) Instruction length and check-graph node count against version index; (b) share of each expansion operator.

10.4; on CascadeEval the same curves run from 70 to 465 tokens and from 1.8 to 9.9, an increment of roughly one node per round. That the two quantities rise *together* is the point: had length grown while the number of checkable propositions stayed flat, "harder" would have meant only "longer". The operator mix is likewise near-identical across splits—48%/35%/17% constraint/sub-task/branch on the training corpus against 47%/36%/16% on CascadeEval—over 22,584 and 1,000 expansions respectively.

The length and node curves measure complexity inside the instruction; whether it also rises for a model reading the instruction is a different question, and the three full-spectrum runs of Table 18 answer it. Across 125 instructions, 8 adjacent pairs and 3 thinking budgets—3,000 adjacent pairs—the verdict changes on 691 of them, and the change is in the harder direction on 62.5% (432 against 259; sign test, one-sided $p = 2.4 \times 10^{-11}$). The inversion rate, passing version $k+1$ after failing version k, is 8.6% (259/3,000), against the 11.5% that equally hard adjacent versions would produce: were the two versions exchangeable, the 691 discordant pairs would split evenly between the two directions. Split by the operator applied at the step, inversions run at 8.4% for a constraint, 8.6% for a dependent sub-task, 7.6% for a deterministic branch and 9.1% for a non-deterministic one—and at 13.2% for an *independent* sub-task, the one operator that widens an instruction without deepening it, and the only one whose steps come close to being difficulty-neutral. It accounts for 4.3% of all expansion steps. The order is real, then, but not exact instruction by instruction, which is the slack the per-instruction offset of equation 4 takes up.

The one place the splits diverge is spectrum length, and that is by construction. CascadeEval requires all $K+1$ versions, whereas reverse truncation leaves the training corpus with spectra of 9, 8, 7 and 6 versions for 2,142 (71.4%), 457, 244 and 157 instructions. Seven in ten training instructions therefore carry a full spectrum, which is what gives Spiral room to move; the remaining three in ten have a lower ceiling T_q, which Spiral's clipping handles without special-casing.

E.2 How Execution Flow Enters the Instructions

The aggregate operator mix hides a rhythm that Figure 17 makes visible. The constraint share falls from about 60% at round 1 to 24% at round 8, the sub-task share rises from 28% to 50%, and the branch share holds at 12–14% through round 6 before roughly doubling to about 26% in the last two rounds. Branches are present from the first round onward; what changes is how often they are chosen. The two scheduling limits that shape this mix are given in §B.3.

Figure 18 profiles the resulting branch structure. About 78% of instructions contain at least one branch (46 have one, 38 have two, 14 have three, and 27 have none), and the two kinds are measured at 83 deterministic against 81 non-deterministic—essentially one to one. Non-deterministic branches are therefore not a rare garnish but half of all execution flow in the benchmark. At the same time, at the hardest version, route nodes are only 165 of 1,241 nodes (13%), the remaining 1,076 (87%) being instruction-constraint nodes. The two counts are close but not equal—164 branch expansions against 165 route nodes—because one sub-task expansion happened to introduce a condition of its own,

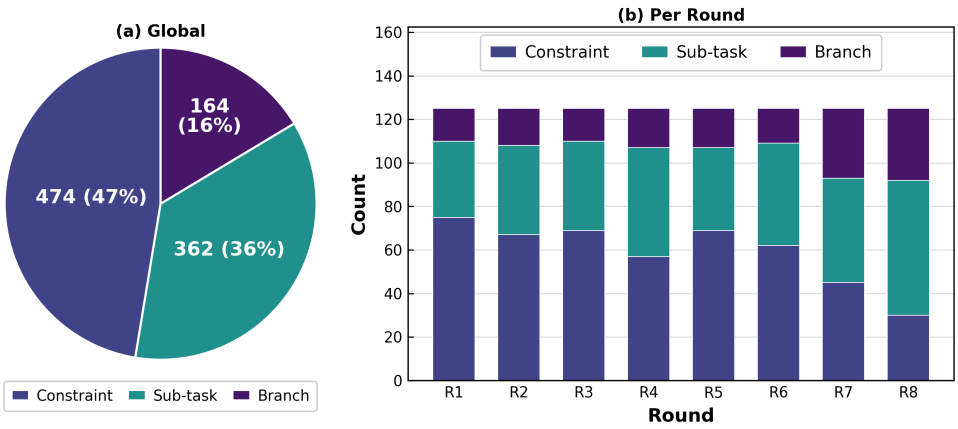


Figure 17: Expansion operators on CascadeEval: (a) overall share; (b) count per round, out of the 125 expansions performed at each round.

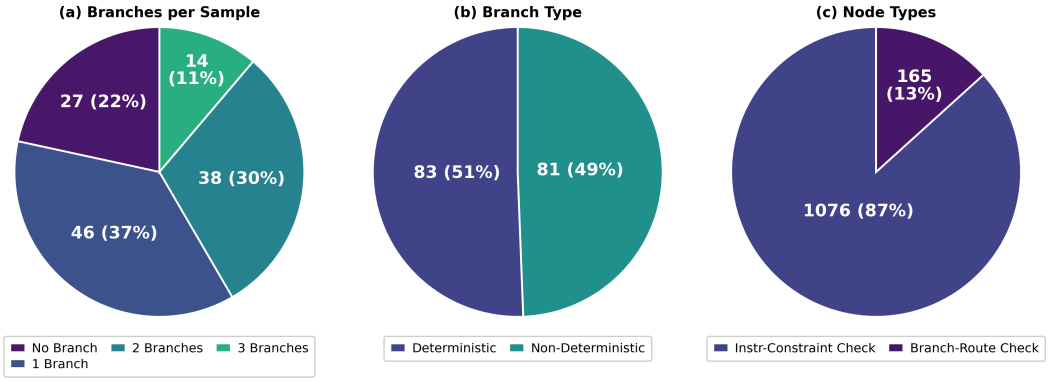


Figure 18: Branch structure of CascadeEval: (a) branches per instruction; (b) deterministic against non-deterministic branches; (c) node types across the 125 hardest-version check graphs.

which the graph realizes as a route node like any other. Branches thus contribute structural difficulty out of all proportion to their verification cost: one route node decides which of two entire paths gets evaluated, while costing at most one decision itself—and nothing at all when it is deterministic.

E.3 What the Checks Actually Test

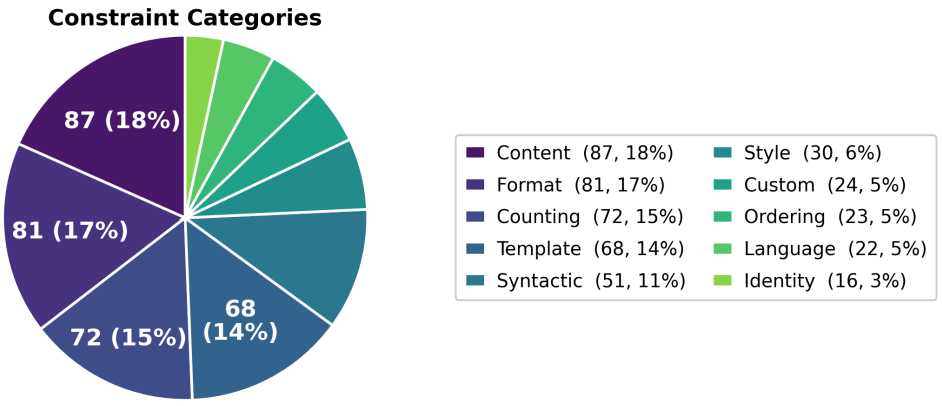


Figure 19: Constraint categories realized in CascadeEval, over 474 constraint expansions; counts and shares for all ten are in the legend.

Figure 19 shows the constraint categories realized in CascadeEval: content 87 (18%), format 81 (17%), counting 72 (15%), template 68 (14%) and syntactic 51 (11%) make up the leading 75%, with style 30, custom 24, ordering 23, language 22 and identity 16 filling the tail, on 474 constraints in all. ("Custom" is a tenth category, recording constraints a model invented outside the nine-category inventory of §B.2.) The head is concentrated but the tail is populated—no category is a token presence, and the corpus has not collapsed onto the format and template constraints that are easiest to verify by rule. Counting alone accounts for 15% of constraints, which is worth holding alongside our evaluation finding that counting is the single most common failure mode of frontier models: the benchmark tests it often because synthesis produced it often, not because it was targeted.

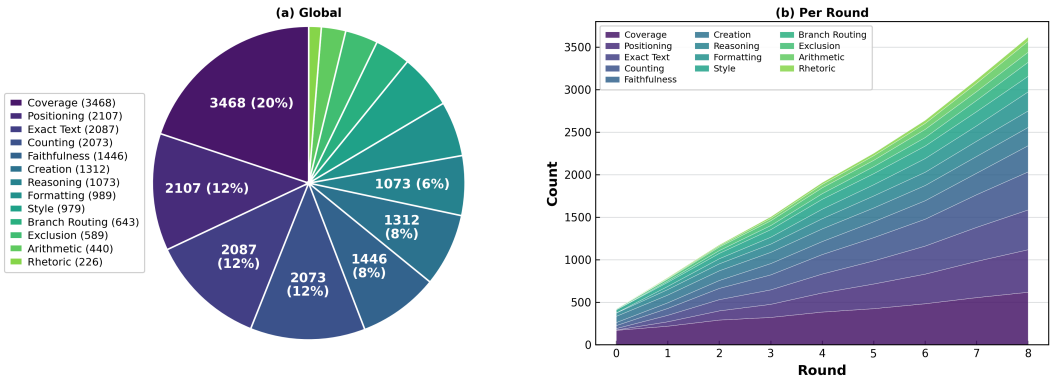


Figure 20: Capability dimensions on CascadeEval check-graph nodes: (a) overall share of the 17,432 labels; (b) composition per round.

The 17,432 capability labels on CascadeEval nodes (Figure 20) are led by coverage 3,468 (20%), then positioning 2,107, exact text 2,087 and counting 2,073 (12% each), followed by faithfulness 1,446, creation 1,312, reasoning 1,073, formatting 989, style 979, branch routing 643, exclusion 589, arithmetic 440 and rhetoric 226. The per-round view matters more than the totals: labels accumulate linearly, by roughly 400 per round, and no dimension takes over as instructions get harder. Coverage gives up share throughout—40% at round 0, 25% by round 2, about 17% by round 8—while the remaining twelve divide the rest in proportions that barely move. Difficulty therefore comes from combining capabilities, not from repeating one.

The co-occurrence structure (Figure 21) is interpretable rather than arbitrary, which is the natural check on whether these labels mean anything. Creation, reasoning and rhetoric all imply coverage ($P = 0.83, 0.81, 0.72$)—one cannot compose, argue or versify without covering content. Exact text and positioning are symmetrically coupled (0.58 and 0.57 in the two directions), reflecting that a string required verbatim is usually also required in a particular place. Branch routing co-occurs with arithmetic (0.39 and 0.57) because a deterministic branch condition is typically a computation over the instruction's own data, yet it essentially never co-occurs with creation, formatting or coverage. A random labeling would not produce a matrix holding both 0.83 and 0.00 in places one can explain after the fact.

E.4 The Hardest Versions, and What They Cover

Figures 22 and 23 give the shape of the hardest versions. Strikingly, the two splits land on the same medians to within half a token—442 and 442.5 tokens of instruction, and 10 check-graph nodes in both—despite disjoint seed task types. CascadeEval concentrates between 7 and 13 nodes (peak 16.8% at 10, range 5–16) with an instruction-length tail near 1,150 tokens and reference responses of median 581 tokens; the training corpus is wider, spanning 2 to 21 nodes with a length tail near 1,400. Two things follow. First, per-instruction verification cost does not explode as complexity rises—path locality showing up in the data rather than in the definition. Second, the benchmark occupies the center of the training distribution rather than extending past it, so CascadeEval is not simply a harder distribution that training never saw; what makes it a held-out test is the disjoint task types, not a shift in difficulty scale.

Figures 24 and 25 confirm that neither split is dominated by a single task type: within CascadeEval's 10 types the largest is social media and casual writing at 19 instructions (15%) and the smallest still

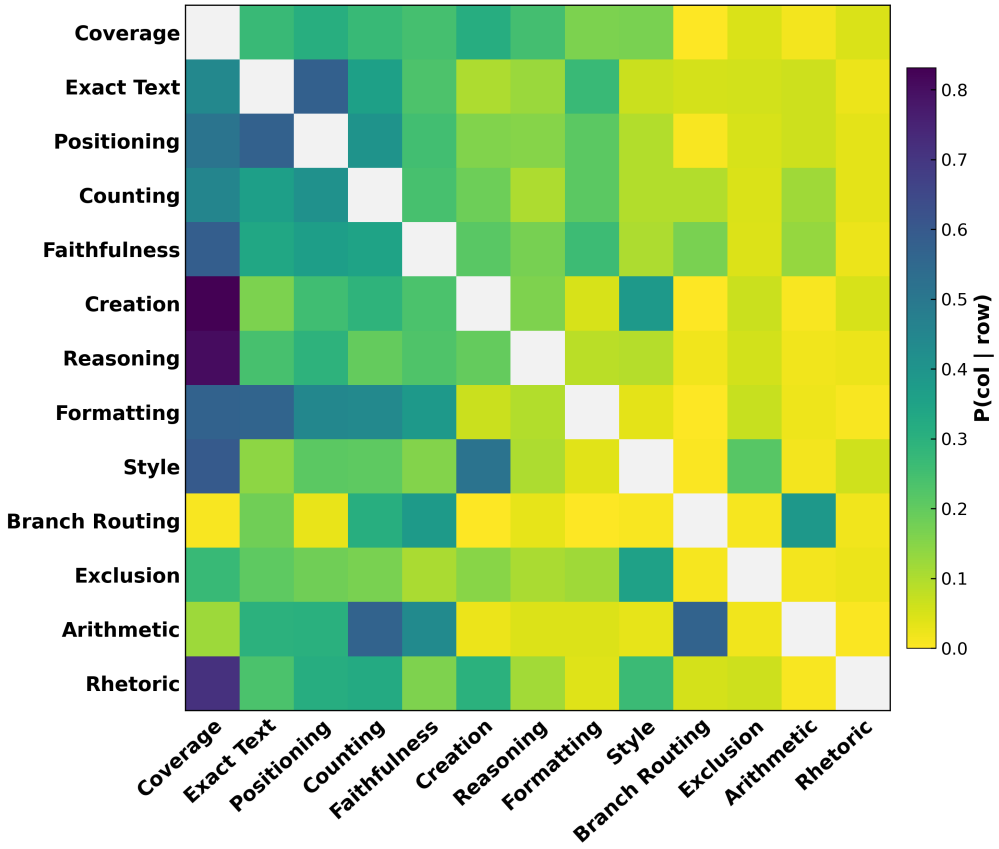


Figure 21: Capability co-occurrence on CascadeEval nodes, $P(\text{column} \mid \text{row})$. The matrix is asymmetric and the diagonal is masked.

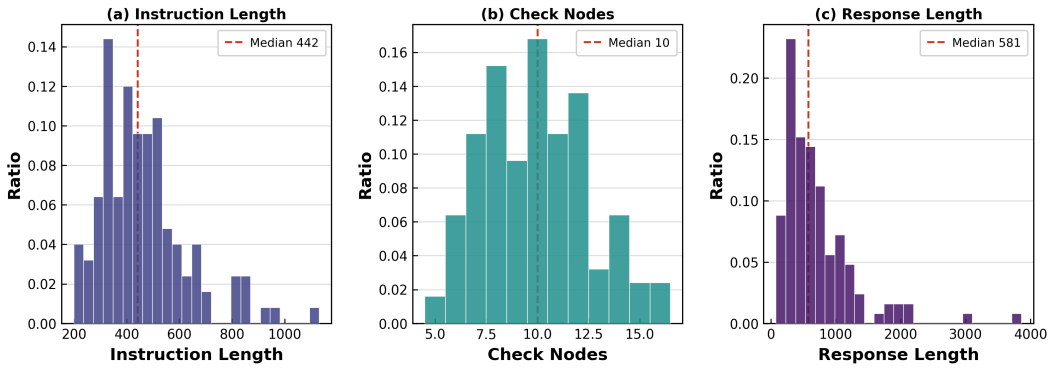


Figure 22: CascadeEval at the hardest version: instruction length, check-graph node count, and reference-response length. Medians are marked.

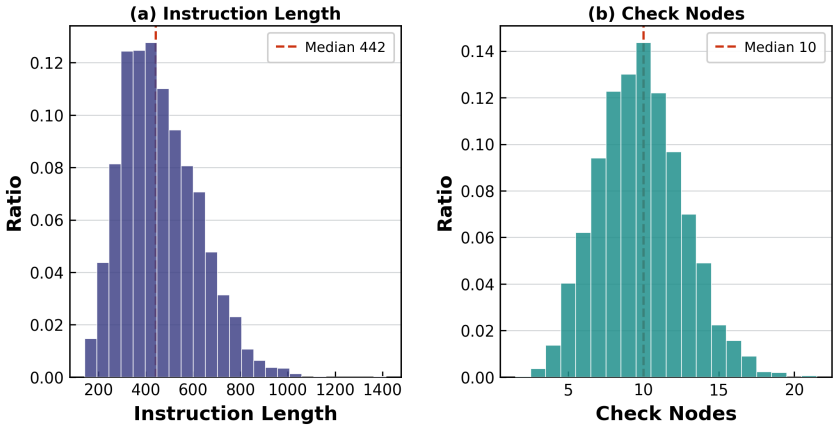


Figure 23: RL training corpus at the hardest version: instruction length and check-graph node count.

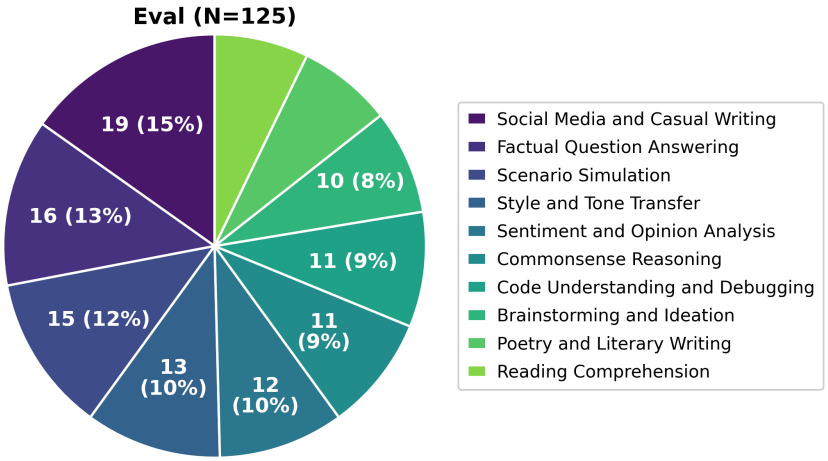


Figure 24: Seed task-type composition of CascadeEval: 10 types over 125 instructions.

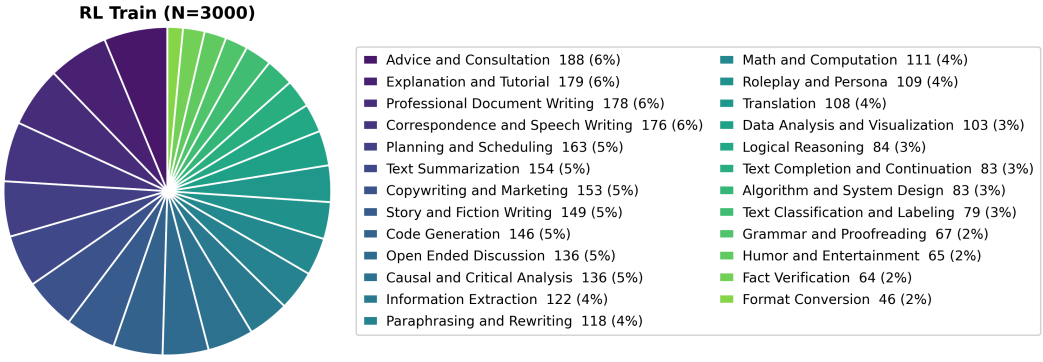


Figure 25: Seed task-type composition of the RL training corpus: 25 types over 3,000 instructions, with counts in the legend. This label set is disjoint from CascadeEval's.

holds 7%, while the training corpus spreads over 25 types with a maximum of 188 instructions (6%). The two label sets do not intersect.

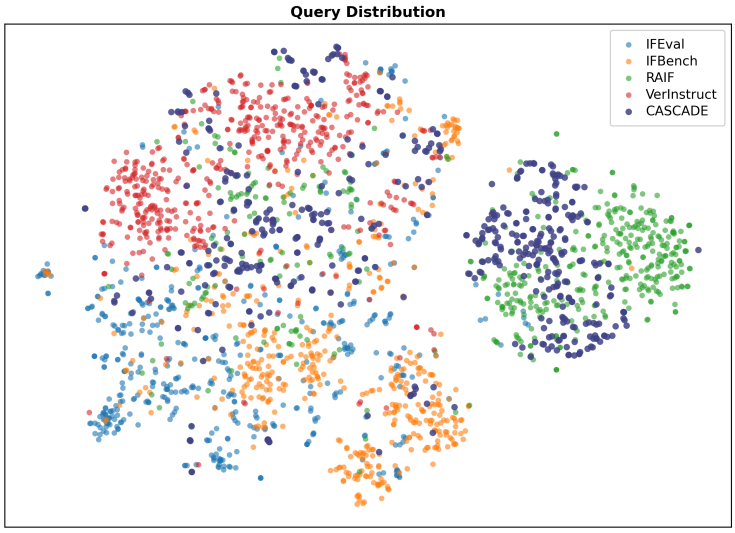


Figure 26: t-SNE (van der Maaten & Hinton, 2008) of instruction embeddings for CASCADE and four existing instruction-following corpora; the axes carry no units, so only relative position is meaningful.

Finally, Figure 26 places CASCADE instructions against IFEval (Zhou et al., 2023), IFBench (Pyatkin et al., 2025), RAIF (Qin et al., 2025) and VerInstruct (Peng et al., 2025) in embedding space, 500 instructions sampled per corpus and embedded with OpenAI `text-embedding-ada-002`. The picture splits into two large clusters. The right-hand one is occupied almost entirely by RAIF and CASCADE, deeply interleaved; within the left-hand one VerInstruct sits upper-left, IFEval lower-left, and IFBench in the lower and right portions. CASCADE is the corpus spread most evenly across *both* clusters—it fills much of the right-hand one and is scattered through every region of the left, with several small clusters of its own—whereas the other four each concentrate in one (RAIF is the least concentrated of them, with about a quarter of its points in the left cluster). Two claims follow, and they pull in opposite directions, which is why both are worth making: the overlap shows CASCADE is not off-distribution synthetic noise disconnected from established instruction-following data, while the exclusive regions show it adds semantic coverage rather than recombining what already exists.

F CASCADEEVAL BENCHMARK RESULTS

We evaluate 48 frontier models at the seed and hardest versions with a single sample ($n = 1$) at temperature 0.7, thinking enabled where available, judged by DeepSeek-V4-Flash, taking ILPR at the hardest version as the headline metric. This protocol differs from the one used for our own training runs in §3, which draw two samples per instruction at temperature 1.0, and the two should not be mixed.

The headline reading is this. The 48 models span **0.088–0.920** ILPR(K)—a tenfold range with neither ceiling nor floor—and node-level attribution, each failure charged to the capability dimensions of the node it was traced to (§B.8), finds a shared weakness: across the top two models of every family, **76%** of them have *counting* as their worst capability dimension, whereas the commonest worst task type covers only 6 of them. When we run one model at three thinking budgets along the full spectrum, all three ILPR curves fall monotonically, confirming that the versions form a difficulty gradient on the model side; at round 0 the non-thinking budget *beats* the High budget, and only by round 7 has the former dropped to less than half of the latter. Judging on the easiest version alone, one would rank the two the wrong way round.

The judge is also an evaluated model. The judge, DeepSeek-V4-Flash, is itself among the evaluated models (ILPR 0.512, rank 34), and the per-round curves of Figure 27 are measured on that same model at three thinking budgets. This affects exactly two places—that model's own row in Table 19 and Figure 27—and neither the ranking of the other 47 models nor any training result in §3, which

uses a different protocol. What the gold-set audit of Figure 10 supports is the *form* of verification—node-by-node checking along a check graph agrees with human annotation on 94% of items—not this particular judge: that audit was run with Qwen3.6-35B-A3B, the training-time reward model. The two judges can, however, be held against each other: every RL run of §3 was scored by both, and over the 63 they have in common their CascadeEval scores correlate at Pearson $r = 0.984$ and Spearman $\rho = 0.977$, with no ordering in Table 1 changing between them (§D.7). Nor is the judge's own mid-table rank evidence against it: were it able to recognize only what it could itself produce, ILPR would be capped near its own 0.512, whereas ten models score at or above 0.8 under it.

F.1 Complexity Decay

Table 18: DeepSeek-V4-Flash across all nine versions at three thinking budgets.

Budget	ILPR, rounds 0–8	CR, 0–8	Mean length, 0–8
Non-thinking	0.936 / 0.776 / 0.712 / 0.632 / 0.608 / 0.496 / 0.392 / 0.280 / **0.240**	0.949 → 0.509	689 → 745
High	0.896 / 0.880 / 0.848 / 0.808 / 0.768 / 0.680 / 0.656 / 0.616 / **0.512**	0.921 → 0.696	992 → 3,270
Max	0.952 / 0.928 / 0.880 / 0.856 / 0.784 / 0.768 / 0.728 / 0.672 / **0.648**	0.972 → 0.761	3,222 → 7,710

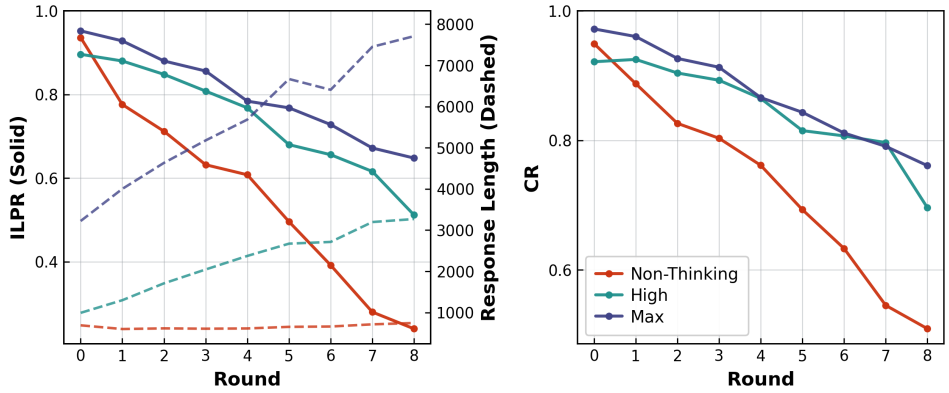


Figure 27: One model (DeepSeek-V4-Flash) at three thinking budgets across all nine versions. Left: ILPR (solid, left axis) with mean response length (dashed, right axis). Right: CR.

Table 18 gives the per-round numbers behind Figure 27; the figure plots them directly, so the two can be read against each other.

Four conclusions follow. First, the spectrum is a genuine difficulty gradient on the model side and not only a structural argument about the data: endpoint decay holds for all 48 models (CRI < 1 throughout Table 19), and the three curves here add that the decline is monotone in between, with no rebound—on one model, since this is the only run we have at every round. Second, CR decays far more slowly than ILPR: in non-thinking mode ILPR falls 74% (0.936 to 0.240) while CR falls 46% (0.949 to 0.509). That gap is itself characteristic of workflow-style instructions—the model does not fail from the start but breaks at some step, and at the hardest version it still completes about half the path on average. ILPR alone would collapse "one step short" and "cannot do this at all" into the same number. Third, thinking buys a slower decline at roughly ten times the tokens (ILPR 0.240 to 0.512 to 0.648 as mean length goes 745 to 3,270 to 7,710); note also that the non-thinking budget barely lengthens its responses as difficulty rises (689 to 745) whereas the High and Max budgets grow 3.3× and 2.4×, which is the mechanism behind non-thinking's faster decay—thinking models add reasoning as the task gets harder, non-thinking ones do not. Fourth, and most useful for benchmark design, the *ordering* of the budgets is itself version-dependent: at round 0 non-thinking (0.936) sits *above* the High budget (0.896) and only just below Max (0.952), and non-thinking and High separate to 0.240 against 0.512 only by round 8. Evaluated on the seed version alone, one would rank non-thinking ahead of High and conclude that a thinking budget buys almost nothing. This is what reading along a difficulty spectrum provides that a single-point evaluation cannot. That thinking and instruction following are in tension at all is not peculiar to CascadeEval: explicit CoT has been reported to degrade instruction-following accuracy (Li et al., 2025a), large reasoning models fail to obey instructions inside their own reasoning traces (Kwon et al., 2025), and whether extended thinking earns its token cost depends strongly on task structure (Wani et al., 2026). What

a difficulty spectrum adds is the location of the crossover: the tension is real at the seed version and reverses as complexity grows.

F.2 Full Leaderboard

Table 19: All 48 models on CascadeEval, ordered by ILPR at the hardest version. "Mode" is the vendor mode evaluated under, "Size" the parameter count—total and active for mixtures of experts—and "Length" the mean response length; a dash marks closed weights or no thinking mode.

Model	Mode	Size	ILPR	CR	CRI	Length
Qwen3.8-Max	Thinking	-	0.920	0.933	0.950	9,393
Qwen3.8-27B	Thinking	27B	0.880	0.908	0.924	11,840
Qwen3.7-Max	Thinking	-	0.864	0.919	0.878	5,899
Kimi-K2.6	Thinking	1.03T / 32B	0.856	0.893	0.884	7,670
GPT-5.6-Sol	Medium	-	0.856	0.929	0.877	3,170
Kimi-K3	Max	2.8T / 104B	0.840	0.895	0.875	4,128
Claude-Opus-5	High	-	0.816	0.879	0.879	3,184
Gemini-3.1-Pro	High	-	0.808	0.878	0.835	6,406
Gemini-3.7-Flash	High	-	0.808	0.875	0.842	6,768
Qwen3.6-Plus	Thinking	-	0.800	0.886	0.840	7,391
Qwen3.7-Plus	Thinking	-	0.784	0.871	0.810	6,649
Gemini-3.6-Flash	High	-	0.784	0.874	0.817	7,926
GLM-5.3	Max	744B / 40B	0.784	0.875	0.824	5,515
DeepSeek-V4-Pro-0813	High	1.65T / 49B	0.784	0.814	0.824	5,808
GPT-5.6-Luna	Medium	-	0.752	0.853	0.797	4,283
GLM-5.2	Max	744B / 40B	0.744	0.830	0.775	5,307
Ling-3.0-Flash	Thinking	128B / 5.1B	0.736	0.865	0.780	6,018
Kimi-K2.7-Code	Thinking	1.03T / 32B	0.728	0.825	0.765	4,564
Claude-Fable-5	High	-	0.720	0.799	0.776	2,432
Qwen3.6-35B-A3B	Thinking	35B / 3B	0.688	0.800	0.741	7,289
Qwen3.5-Plus	Thinking	397B / 17B	0.680	0.814	0.685	6,974
Kimi-K2.5	Thinking	1.03T / 32B	0.664	0.762	0.686	4,195
Qwen3.5-122B-A10B	Thinking	122B / 10B	0.648	0.742	0.669	6,429
Claude-Sonnet-5	High	-	0.640	0.782	0.708	4,887
GLM-5.1	Thinking	744B / 40B	0.632	0.678	0.693	3,927
MiniMax-M3	Thinking	427B / 23B	0.632	0.764	0.693	5,209
GLM-5	Thinking	744B / 40B	0.624	0.763	0.655	3,563
Qwen3.5-35B-A3B	Thinking	35B / 3B	0.608	0.765	0.639	8,770
Kimi-K2-Thinking	Thinking	1.04T / 32B	0.568	0.706	0.617	3,660
DeepSeek-V4-Pro	High	1.6T / 49B	0.560	0.727	0.598	2,844
MiniMax-M2.7	Thinking	230B / 10B	0.552	0.689	0.595	6,210
MiniMax-M2.5	Thinking	230B / 10B	0.536	0.697	0.568	5,087
GLM-4.7	Thinking	358B / 32B	0.520	0.731	0.565	4,915
DeepSeek-V4-Flash	High	284B / 13B	0.512	0.696	0.571	3,270
DeepSeek-V3.2-Exp	Thinking	671B / 37B	0.456	0.654	0.504	2,293
Qwen3-Next-80B-A3B	Thinking	80B / 3B	0.456	0.638	0.483	5,707
DeepSeek-V3-0324	Thinking	671B / 37B	0.440	0.608	0.462	2,039
DeepSeek-V3.1-250821	Thinking	671B / 37B	0.416	0.613	0.441	2,043
DeepSeek-R1-0528	Thinking	671B / 37B	0.376	0.600	0.392	3,158
GLM-4.6	Thinking	355B / 32B	0.344	0.588	0.361	3,055
DeepSeek-R1	Thinking	671B / 37B	0.240	0.503	0.270	1,215
Qwen3-32B	Thinking	32B	0.200	0.445	0.229	1,155
Qwen2.5-72B-Instruct	-	72B	0.168	0.433	0.202	566
QwQ-32B	Thinking	32B	0.168	0.480	0.189	1,629
MiniMax-M2.1	Thinking	230B / 10B	0.168	0.468	0.183	2,572
Qwen3-235B-A22B	Thinking	235B / 22B	0.160	0.433	0.187	1,579
Qwen3-8B	Thinking	8B	0.144	0.417	0.168	1,233
Qwen3-30B-A3B	Thinking	30B / 3B	0.088	0.391	0.101	1,275

Evaluated models. Seventeen of the 48 rows in Table 19 are covered by a technical report and cited accordingly: Qwen2.5 (Qwen et al., 2024) and Qwen3 (Yang et al., 2025); DeepSeek V3 (DeepSeek-AI et al., 2024), R1 (DeepSeek-AI et al., 2025), V3.2 (DeepSeek-AI et al., 2025) and V4 (DeepSeek-AI et al., 2026); Moonshot's Kimi K2 (Kimi Team, 2025), K2.5 (Kimi Team, 2026a) and K3 (Kimi Team, 2026b); and GLM-5 (GLM-5 Team, 2026). Every model, cited or not, is identified by the exact release or vendor model string it was evaluated under, which Table 19 lists verbatim.

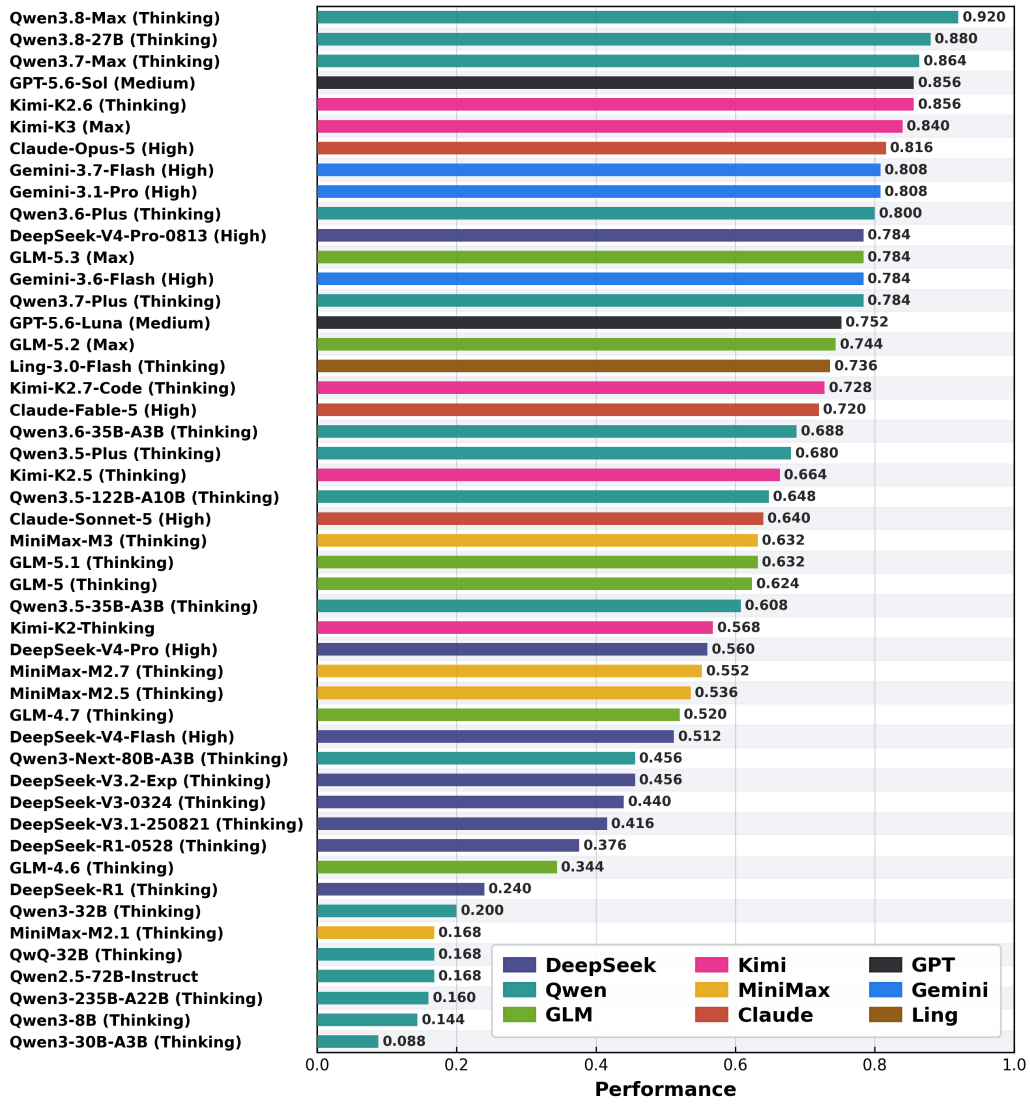


Figure 28: All 48 models by ILPR at the hardest version, colored by family.

Figure 28 plots the same ranking. Four things are worth noting. The range spans **0.088** to **0.920**, over tenfold, with ten models at 0.8 or above, nine in 0.7–0.8, nine in 0.6–0.7 and twenty below 0.6: the benchmark is neither saturated nor out of reach. The head is well separated—0.12 of ILPR between first and tenth—so the top does not collapse into one tied band, though at $n = 1$ on 125 instructions (binomial SE 0.03 at the top of the board, 0.045 mid-table) adjacent ranks should not be read as separated. CRI tracks ILPR closely, and that is itself the point: every model is near saturation on the seed version (ILPR(0) runs from 0.83 to 0.99), so almost the entire 0.088–0.920 spread is manufactured by expansion rather than inherited from the seed tasks. Finally, the open/closed gap sits at the tail rather than the head: the leader is closed-weight but second place is open (Qwen3.8-27B, 0.880) and three of the top ten are open, while all twenty models below ILPR $= 0.6$ are open.

Figure 29 asks a different question of the same 48 scores: whether the benchmark tracks progress over time, and whether that ability erodes as models improve. It does track progress, but only along

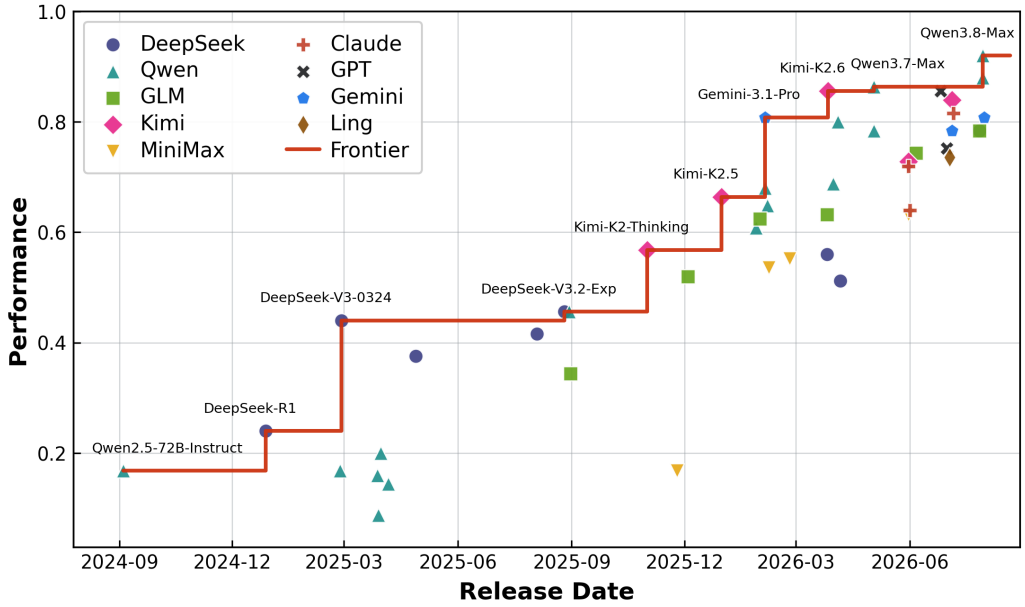


Figure 29: ILPR at the hardest version against release date, with the frontier envelope; closed-model dates are when our gateway recorded them.

the envelope. The frontier rises from 0.168 (2024-09) to 0.920 (2026-08) in ten record-setting steps, with a plateau at 0.440–0.456 between 2025-03 and 2025-09 and a jump from 0.664 to 0.808 in 2026-02. Within any single month the spread is wide, so "newer is better" is a statement about the envelope and not about models pairwise: Qwen3-30B-A3B (2025-04, 0.088) scores below Qwen2.5-72B-Instruct (2024-09, 0.168), released seven months earlier. That the envelope has not flattened at 0.920 is the property that matters for the benchmark's shelf life.

F.3 Failure Attribution

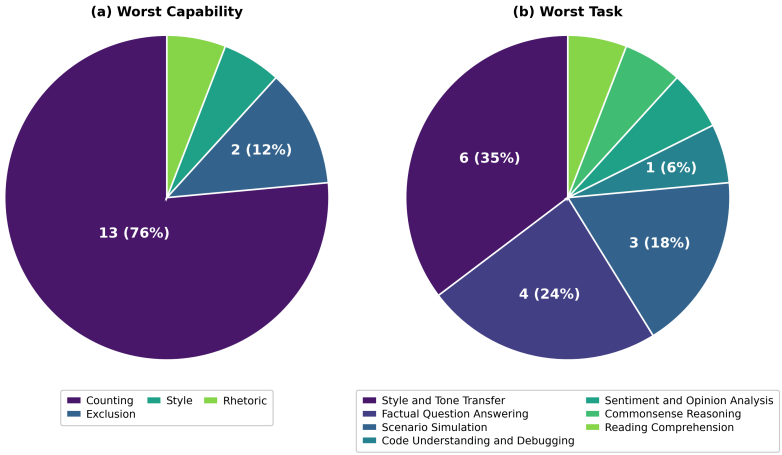


Figure 30: Worst capability dimension (left) and worst task type (right), over the top two models of each family (17 models).

Figure 30 aggregates, for the strongest models of each family, which capability dimension and which task type each one fails on most often: capabilities are ranked by the node-level failure rate of equation 12, task types by the per-task-type failure rate of equation 13, which is instruction-level because a task type labels the instruction rather than the node. Both are taken at the hardest version alone—$\mathcal{K} = \{K\}$ in equation 12—and not summed over rounds; Table 18 and Figure 27 are the only results here that use every round, and they cover one model.

Counting is the worst dimension for 13 of the 17 leading models (76%), and a share that large makes it a systematic weakness across vendors and generations rather than a quirk of any one model; it is also a conclusion only node-level capability labels can produce: ILPR says a model failed, the capability breakdown says what kind of requirement it failed on. That the task side is far less concentrated (the commonest covers 6 of the 17) says the shortfall is a capability, not a domain.

The per-model breakdowns in Figures 31–42 show the same point at individual scale, in the order of Table 19, four models per sheet. The two ends of the leaderboard look very different at this resolution: Qwen3.8-Max peaks at a 1.0% capability failure rate (exclusion) and 18.2% on its worst task, whereas DeepSeek-R1 (DeepSeek-AI et al., 2025) reaches 17.7% on counting and fails **every** instruction of both commonsense reasoning and style transfer. Two cautions apply in reading these sheets. The vertical scale adapts per model, so bar heights are not comparable across sheets—read the values. And equation 12 ranks dimensions rather than estimating a probability: the dimensions are exercised very unevenly, so the thin end of any model's ranking rests on few node visits and only its head should be read as a finding.

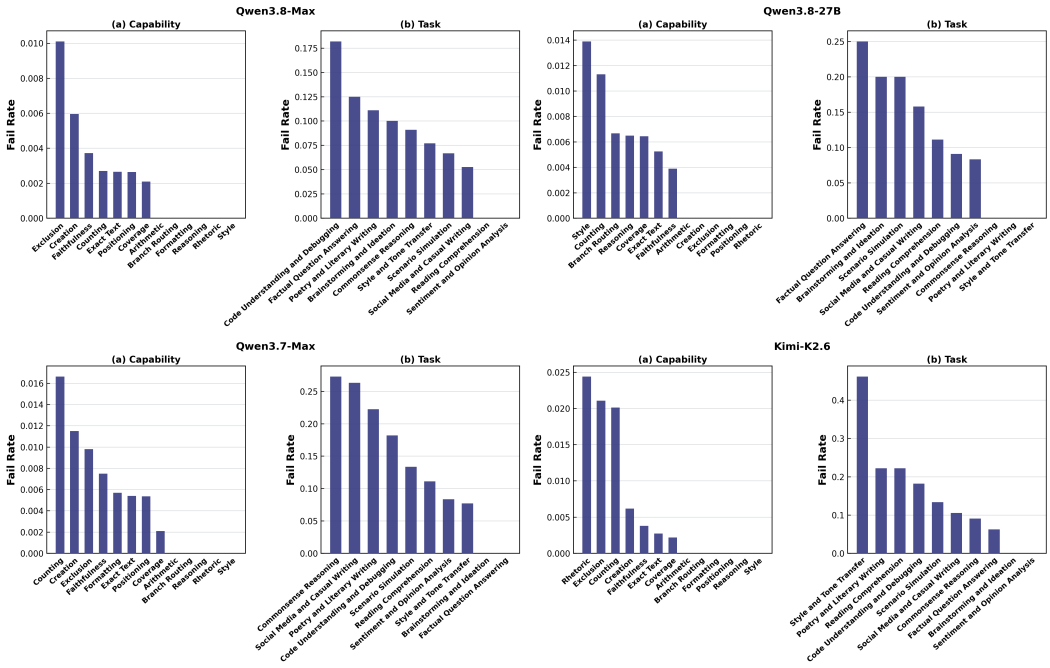


Figure 31: Per-model failure rates, sheet 1 of 12, in leaderboard order.

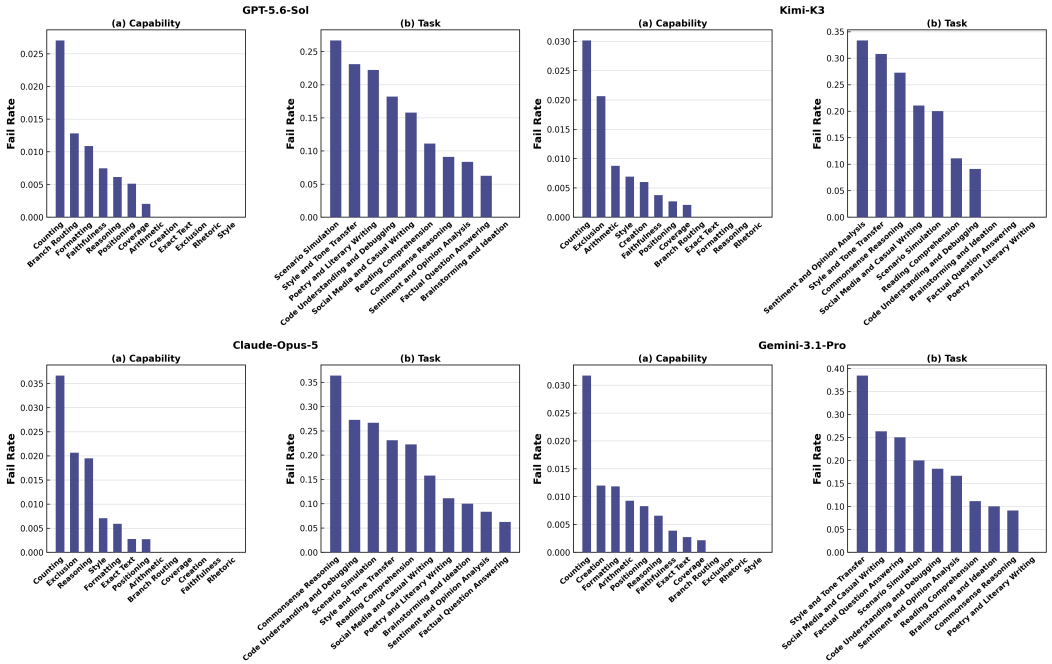


Figure 32: Per-model failure rates, sheet 2 of 12, in leaderboard order.

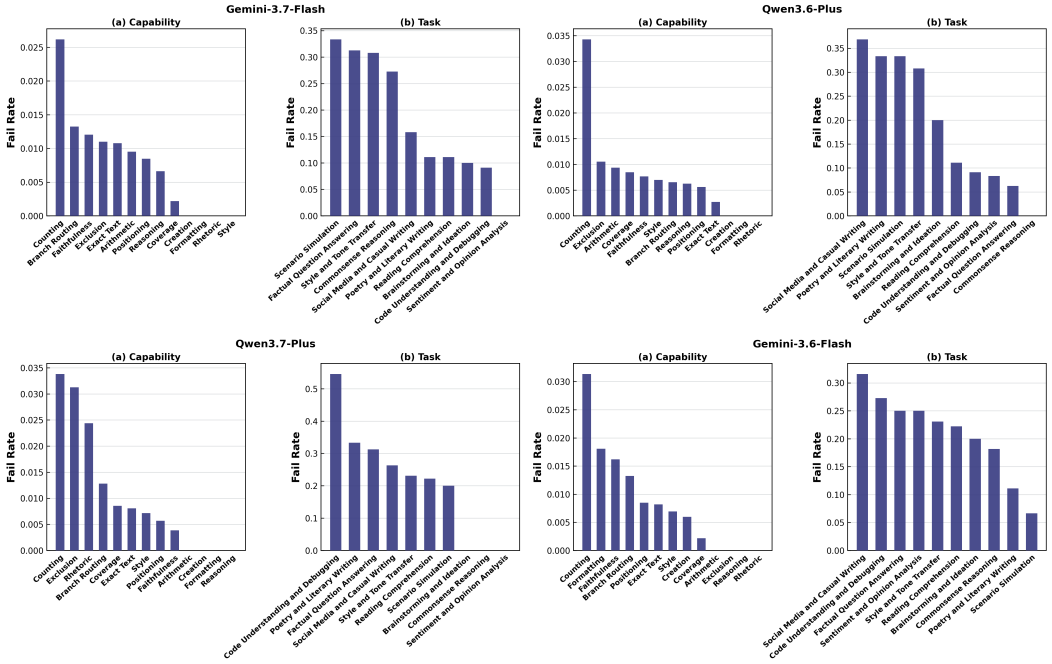


Figure 33: Per-model failure rates, sheet 3 of 12, in leaderboard order.

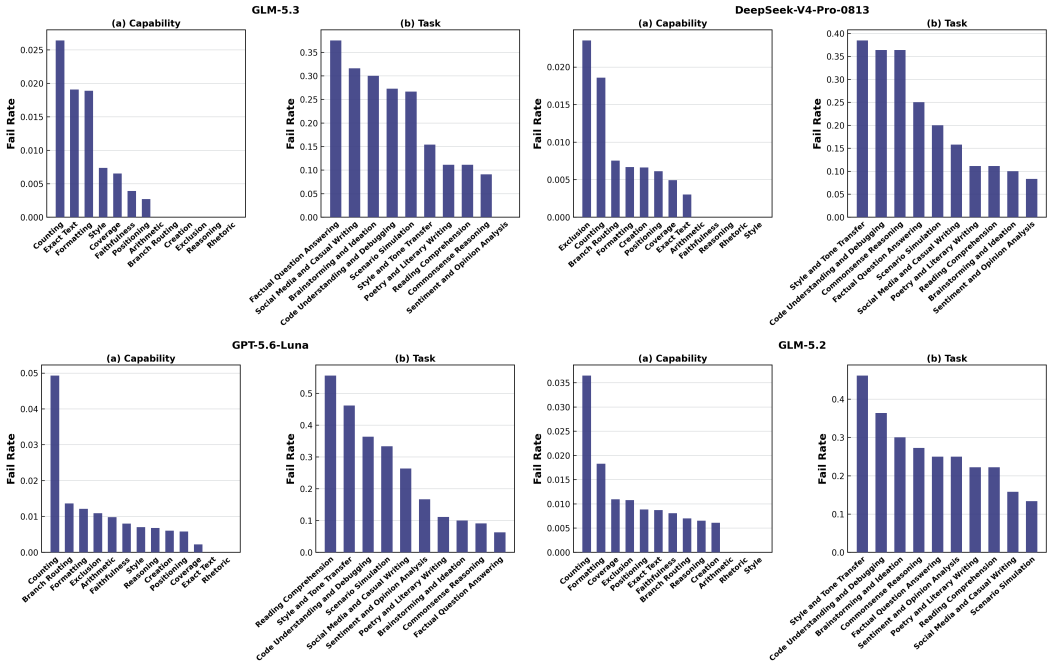


Figure 34: Per-model failure rates, sheet 4 of 12, in leaderboard order.

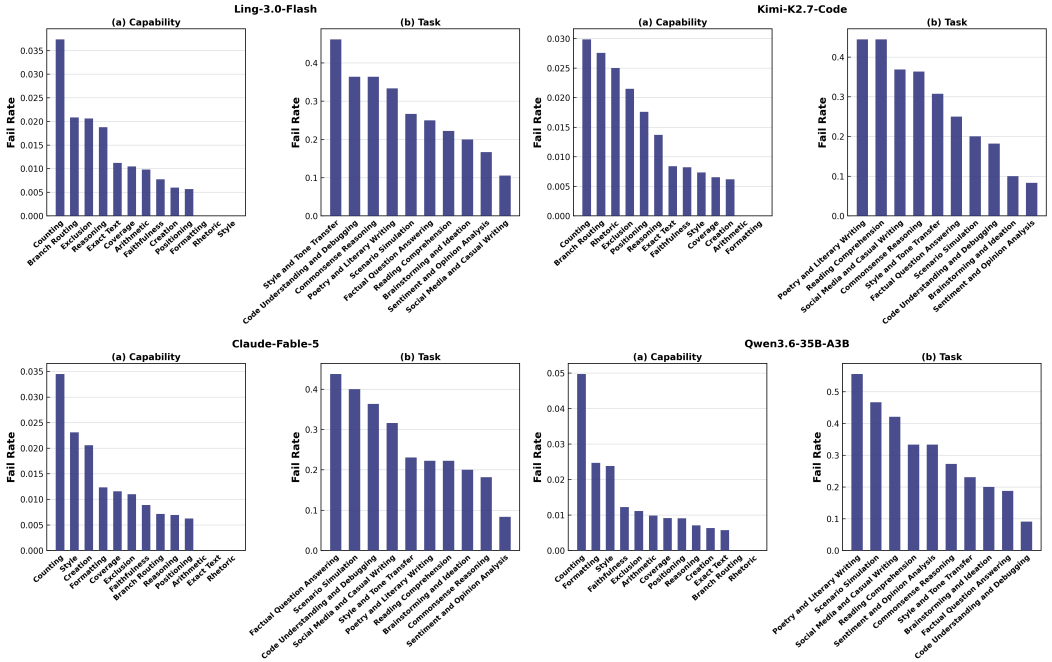


Figure 35: Per-model failure rates, sheet 5 of 12, in leaderboard order.

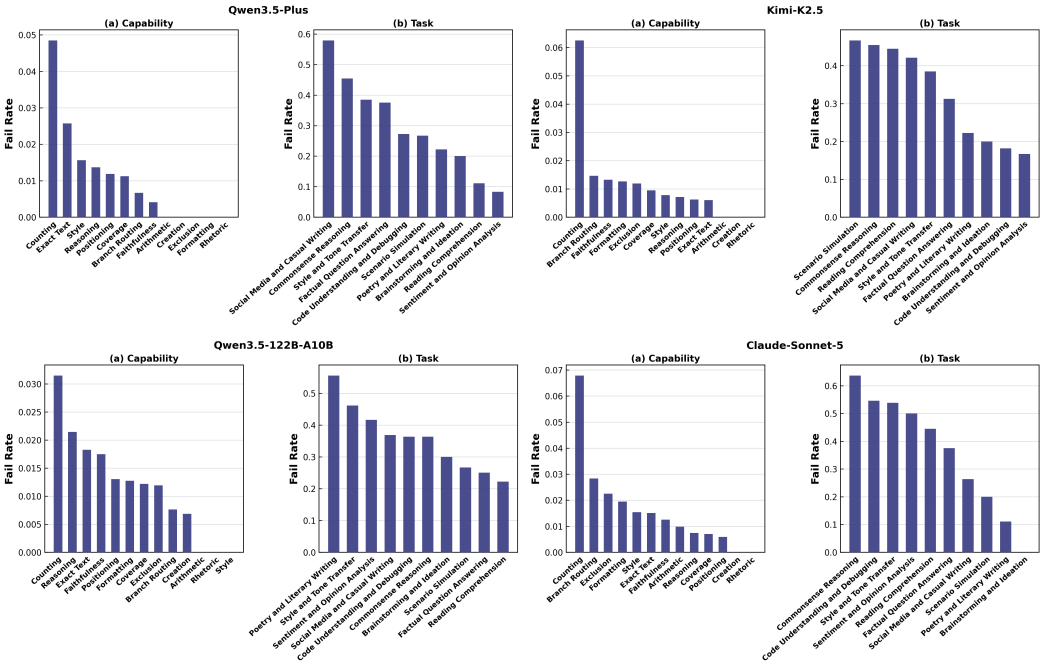


Figure 36: Per-model failure rates, sheet 6 of 12, in leaderboard order.

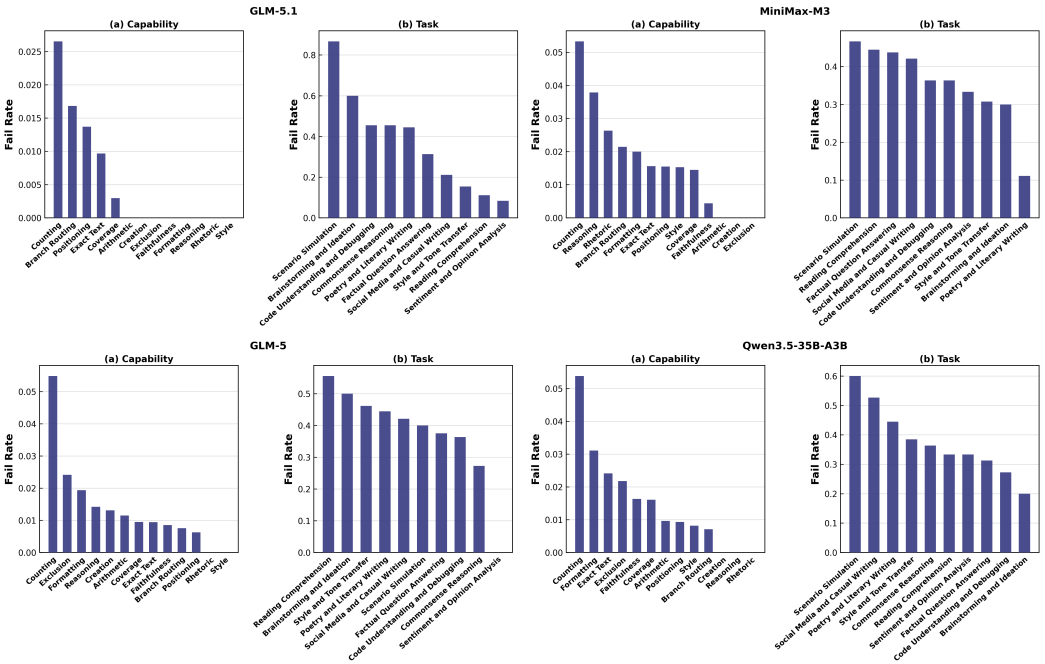


Figure 37: Per-model failure rates, sheet 7 of 12, in leaderboard order.

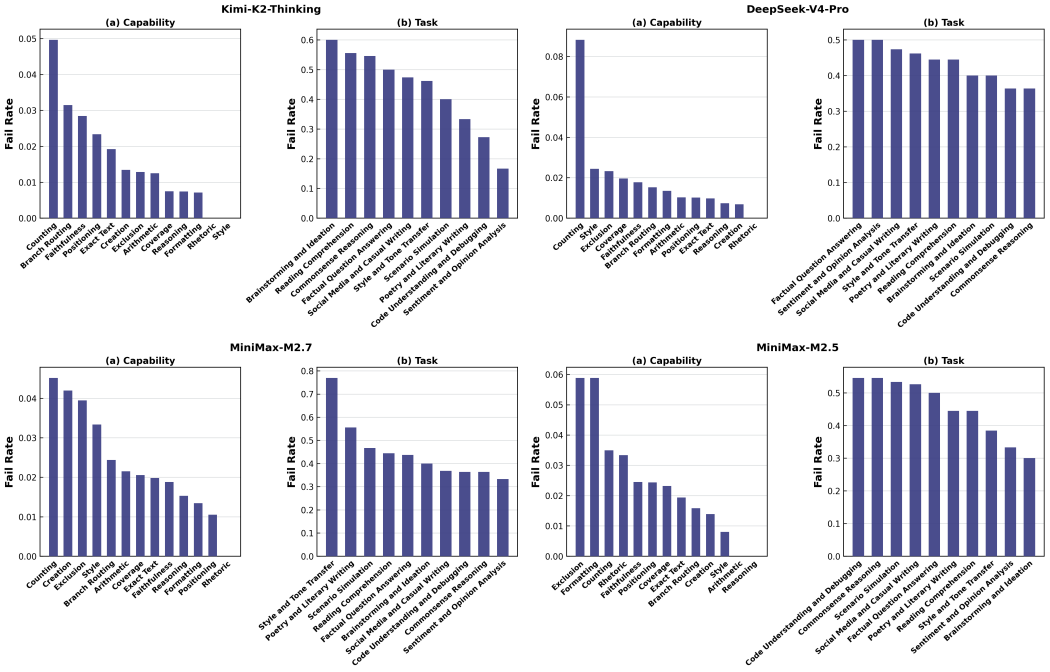


Figure 38: Per-model failure rates, sheet 8 of 12, in leaderboard order.

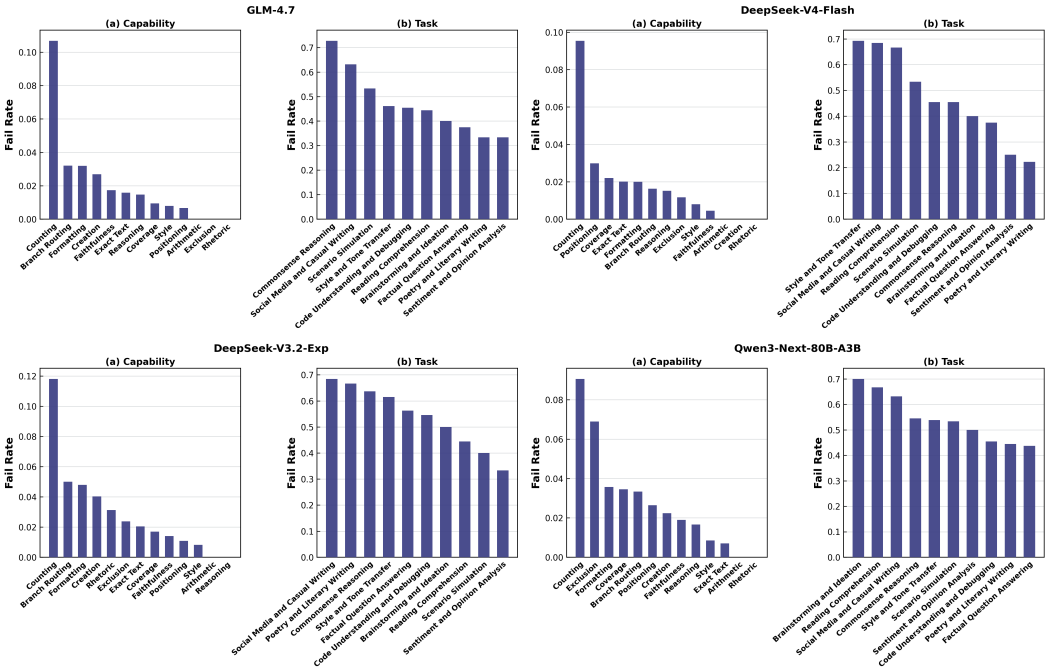


Figure 39: Per-model failure rates, sheet 9 of 12, in leaderboard order.

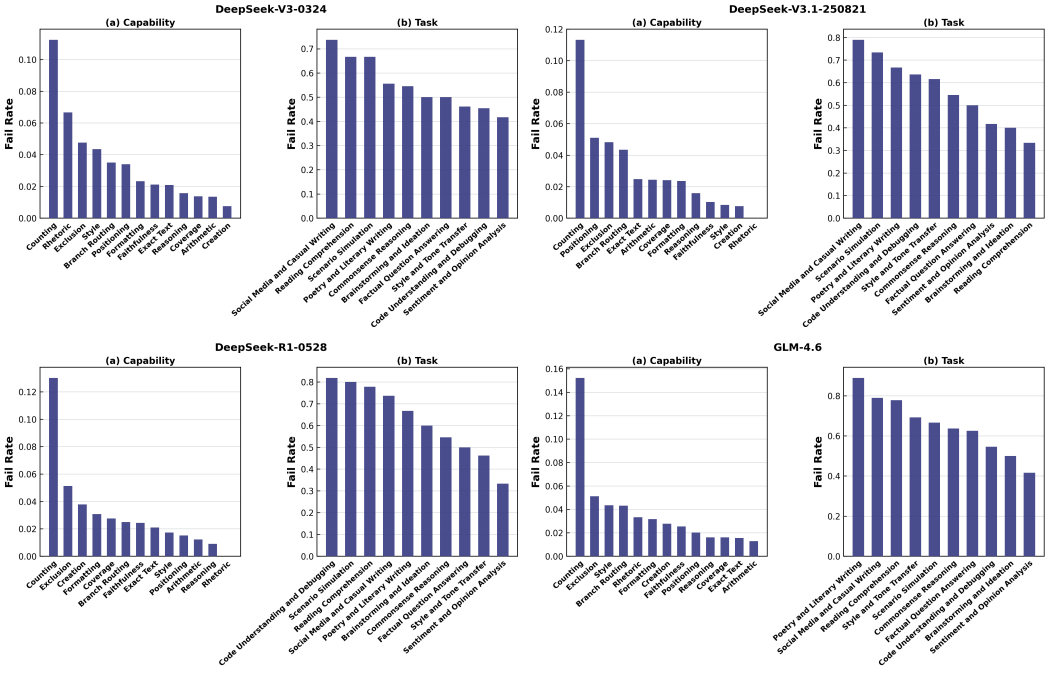


Figure 40: Per-model failure rates, sheet 10 of 12, in leaderboard order.

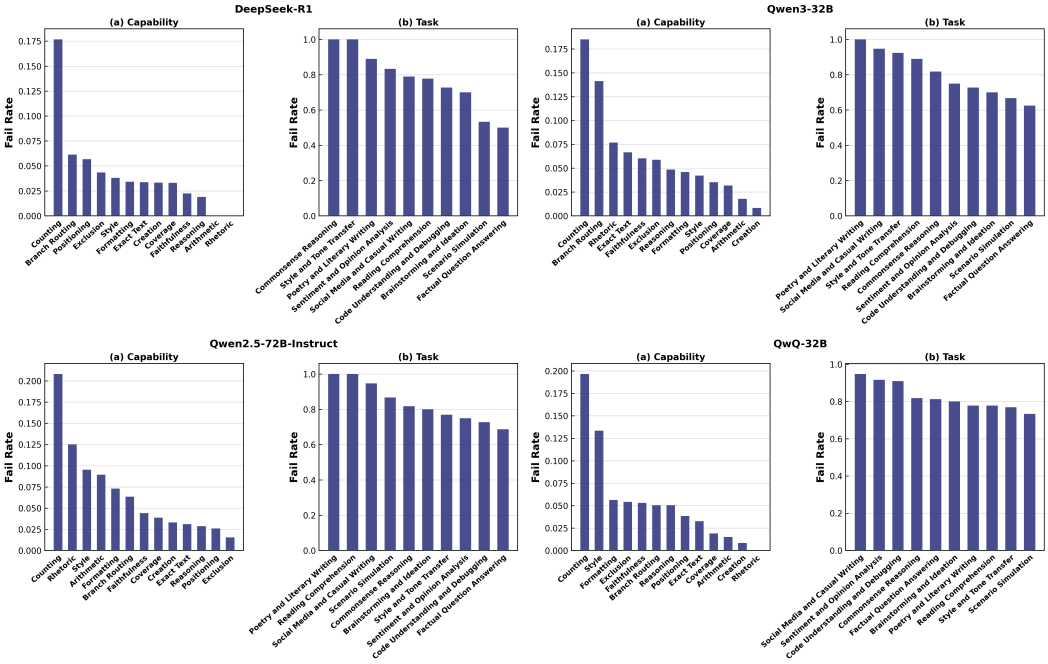


Figure 41: Per-model failure rates, sheet 11 of 12, in leaderboard order.

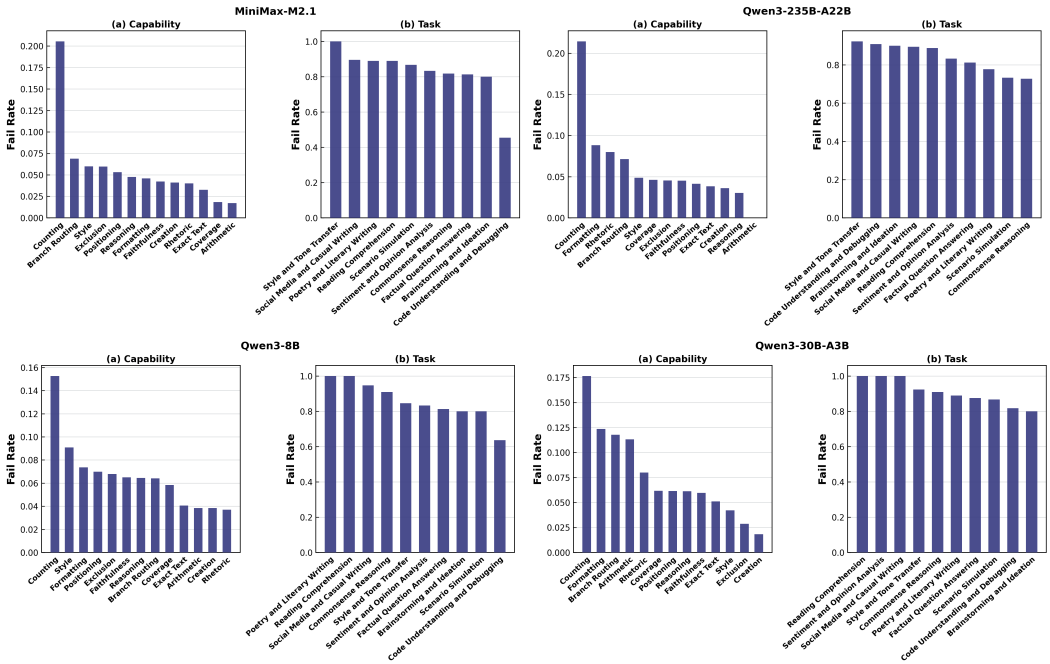


Figure 42: Per-model failure rates, sheet 12 of 12, in leaderboard order.

G Case Study

This section follows two benchmark instructions from seed to hardest version, one in English and one in Chinese. For each we print the round-0 seed and the round-K instruction in full, summarize the K expansion rounds by the operator applied and the unit it added, and give the check graph of the final version. Reading the two together shows what atomic expansion does in practice: the seed's own requirements survive in the final graph, and every other node traces back to one or more specific rounds—the merge-first rule of §B.5 sometimes folds several rounds into one node. Both seeds are drawn from WildChat (Zhao et al., 2024).

G.1 Case EN-1: English, Scenario Simulation

The seed asks for one thing. Eight rounds later the instruction is a five-step workflow under three global constraints and two conditional branches—one of each kind, which is why this case also serves as Figure 1 in the main text.

Case EN-1 (`scenario_simulation`, source: WildChat)

Round 0 — seed instruction.
Write a detailed description of a hypothetical image, depicting a home computer setup from the late 1970s.

Round 8 — hardest version.
Review this vintage computer spec sheet:

```
Model: Retro-8 | RAM: 8192 bytes | CPU: 1 MHz | Storage: Audio
Cassette
```

Then, I'd like you to write a detailed description of a hypothetical image showing a late-1970s home computer setup, based on that sheet. Before you start writing, convert the RAM from bytes to kilobytes (using 1024 bytes = 1 kilobyte). If the resulting kilobyte value is even, your image description must focus solely on visual and auditory elements, with no mention of smell or touch. If it's odd, you must include at least one simile when describing the machine's operation or the loud sounds of its thick keys. After drafting your description, check whether it mentions a tape drive. To declare this, insert the exact string `'[FLAG: YES]'` or `'[FLAG: NO]'` on its own line between your description and what follows. If the flag is YES, explain the steps a user takes to load a game from that tape drive. If it's NO, describe the loud sounds the thick keys make when pressed. Next, based on the specific steps or sounds you just detailed, invent a short title for a magazine article that focuses on that exact user experience. Then, briefly describe the target demographic that would be most drawn to an article with that specific name. Throughout your entire response, keep the language simple, direct, and highly varied: no word may exceed 10 characters in length, and no word (case-insensitive) may appear more than 3 times. Also, write the entire response without using a single comma.

Table 20 records how the instruction got there. Two points are worth drawing out. First, the non-deterministic branch arrives early, at round 2, and everything after it is written around it—the flag declaration added at round 3 exists only to make that branch's outcome legible, and the two dependent sub-tasks at rounds 6 and 7 chain off whichever side of it was taken. Second, the deterministic branch added at round 8 is inserted at the *front* of the instruction, not appended to the end: it brings its own spec sheet as new material and restructures what came before. This is what we mean by an operator acting anywhere in the instruction.

The final check graph (Table 21) has 9 nodes, 7 of them instruction-constraint nodes and 2 route nodes. Node 1 is deterministic: $8192/1024 = 8$ is even, so its truth value is settled when the graph is built and costs no model call at verification time. Because that value is fixed to true, node 3 and everything reachable only through it are unreachable for every response, which is why the simplified drawing of Figure 1 collapses that branch into *Fail* rather than showing it: node 1 itself never fails a response, and by equation 6 no route node has an edge to $\perp$. Node 4 is non-deterministic—whether the description mentions a tape drive cannot be known until the description exists. A response taking the even/tape-drive path is checked at $1 \to 2 \to 4 \to 5 \to 7 \to 8 \to 9 \to \top$, which is 6 model calls for a 9-node graph, since node 1 is free and node 3 and node 6 lie on paths not taken.

Table 20: Case EN-1, evolution trace. "Nodes" is the size of that round's check graph; "det."/"non-det." abbreviate deterministic/non-deterministic.

R	Operator	Nodes	Unit added
0	seed	1	—
1	constraint (counting)	2	No single word in the response may exceed 10 characters.
2	non-det. branch	5	Check the drafted description and either explain the game-loading process or describe the keyboard sounds, depending on whether it mentioned a tape drive.
3	constraint (format)	5	Declare that outcome by emitting `[FLAG: YES]` or `[FLAG: NO]` on its own line.
4	constraint (counting)	5	No word (case-insensitive) may appear more than 3 times.
5	constraint (syntactic)	5	No commas anywhere in the response.
6	dependent sub-task	6	Invent a magazine article title based on the steps or sounds just described.
7	dependent sub-task	7	Describe the target demographic drawn to that title.
8	det. branch	9	Add a spec sheet, and branch on whether its RAM converts to an even number of kilobytes: sensory restriction if even, a required simile if odd.

Table 21: Case EN-1, round-8 check graph. "constraint" abbreviates instruction-constraint; "GT" is the truth value a deterministic route node carries; T and F give the next node on pass and fail, with $\top$ and $\bot$ the terminals.

#	Type	GT	Check problem	T	F
1	route (det.)	True	Based on the spec sheet provided in the Instruction, determine whether the RAM value converts to an even number of kilobytes (where 1024 bytes = 1 kilobyte)?	2	3
2	constraint	—	For the Response, determine whether it provides a detailed description of a hypothetical image depicting a home computer setup from the late 1970s based on the provided spec sheet, focuses entirely on visual and auditory elements, and strictly avoids any mention of smell or touch?	4	$\bot$
3	constraint	—	For the Response, determine whether it provides a detailed description […] and uses at least one simile when describing the machine's operation or the loud sounds of its thick keys?	4	$\bot$
4	route (non-det.)	—	Based on the generated description of the home computer setup in the Response, determine whether it mentions a tape drive?	5	6
5	constraint	—	For the Response, determine whether it inserts the exact string `'[FLAG: YES]'` on its own line between the description and the follow-up, and explains the steps a user takes to load a game from the tape drive?	7	$\bot$
6	constraint	—	For the Response, determine whether it inserts the exact string `'[FLAG: NO]'` on its own line […], and describes the loud sounds the thick keys make when pressed?	7	$\bot$
7	constraint	—	For the Response, determine whether it invents a short title for a magazine article that focuses on the exact user experience based on the specific steps or sounds detailed?	8	$\bot$
8	constraint	—	For the Response, determine whether it briefly describes the target demographic that would be most drawn to an article with the specific invented title?	9	$\bot$
9	constraint	—	For the entire Response, determine whether it is true that no single word exceeds 10 characters in length, that absolutely no word (case-insensitive) appears more than 3 times, and that not a single comma is used?	$\top$	$\bot$

Five of these nodes correspond to no single round. Node 9 folds together the constraints added at rounds 1, 4 and 5—all three are global properties of the finished text, so the merge-first rule of §B.5 rewrote one node's problem three times rather than adding three nodes. Nodes 5 and 6 each absorb both the flag from round 3 and one side of the round-2 branch. And the seed's own requirement is still there, inside nodes 2 and 3, each now also carrying whichever sensory constraint round 8 selected.

G.2 Case ZH-1: Chinese, Poetry and Literary Writing

The Chinese case mirrors the first in two respects. Its round-8 branch is deterministic with the truth value *False*, so the path actually verified is the `else` side—together with Case EN-1, whose deterministic route node resolves to True, the pair exercises both outcomes. And its difficulty, though it starts from a seed of a single 17-character clause, is carried by a chain of dependent sub-tasks rather than by branching: count the characters of the poem just written, convert that count to binary, then count the ones and zeros in that binary number. Each step consumes the previous step's output, so a single early error propagates to the end.

Case ZH-1 (`poetry_and_literary_writing`, source: WildChat)

Round 0 — seed instruction.
用李白的口吻写一首关于人工智能的诗

Round 8 — hardest version.
帮我阅读以下【长安邸报·机巧太白测试简报】:

> 驱动核心：灵石阵列
> 收录诗卷：48900 首
> 连续作诗：120 首/半炷香
> 故障记录：遇“情感”类问题宕机 1 次

请根据这个背景，用李白的口吻写一首关于人工智能的诗。动笔前先看一眼简报里的数据：如果收录诗卷的数量达到或超过 50000 首，就在诗歌正文前额外写一段 50 字左右的文言文序言，感慨机器学识之浩瀚；如果不满足，则要求诗歌正文必须分为恰好 2 个段落（比如上下两阕，中间用空行隔开）。
诗中需要原封不动地把“举杯邀明月”作为其中一行嵌入，并且最后一句要脱离对 AI 的具体描写，升华为对“何为真实”或“万物皆空”的哲学思考。
诗成之后，请统计全诗（不含标点符号）的总字数。为排版清晰，请将诗歌正文整体用双引号"" 包裹起来，并在正文与字数统计之间插入 Markdown 分隔线 ---，然后单起一行以“总字数：X 字”的格式给出结果。既然是关于 AI 的诗，得出总字数后，请把这个数字转换成二进制形式，在字数统计的下一行以“二进制字数：Y”的格式写出来。最后，统计这个二进制数字里一共包含几个“1”和几个“0”，并在最下方单起一行以“其中包含：A 个 1，B 个 0”的格式补充说明。

Table 22: Case ZH-1, evolution trace. Rounds 2, 6 and 7 form a dependency chain: each consumes the output of the one before it. “det.” abbreviates deterministic.

R	Operator	Nodes	Unit added
0	seed	2	—
1	constraint (content)	2	Embed the line “举杯邀明月” verbatim as one line of the poem.
2	dependent sub-task	3	Count the characters of the poem just written and report the total in a given format.
3	constraint (format)	3	Separate the poem from the count with a Markdown rule.
4	constraint (template)	4	Wrap the body of the poem in double quotes.
5	constraint (content)	5	The closing line must leave AI behind and rise to a philosophical reflection.
6	dependent sub-task	5	Convert that character count to binary and report it.
7	dependent sub-task	5	Count the ones and zeros in that binary number and report them.
8	det. branch	8	Add a briefing as background, and branch on whether its corpus size reaches 50,000 poems: a classical-Chinese preface if so, otherwise exactly two stanzas.

Table 22 shows the same merge behavior from a different angle. Rounds 6 and 7 each add a genuine sub-task, yet the graph does not grow: both are folded into node 8, which already checked the character count from round 2, because all three are checks on the same block of output and a single verification problem can settle them at once. The node count is therefore a lower bound on how much the instruction demands, not a measure of it—a point that matters when reading the check-graph node curves of §E.

Table 23: Case ZH-1, round-8 check graph. The deterministic route node at node 1 carries GT = False, so verification takes the `else` path through node 3.

#	Type	GT	Check problem	T	F
1	route (det.)	False	针对 Instruction 中【长安邸报·机巧太白测试简报】的数据，判断其中收录诗卷的数量是否达到或超过 50000 首?	2	3
2	constraint	—	针对 Response，判断在诗歌正文前是否额外包含了一段文言文序言，且该序言的字数在 50 字左右，内容主题为感慨机器学识之浩瀚?	4	⊥
3	constraint	—	针对 Response 中的诗歌部分，判断诗歌的正文是否恰好分为 2 个段落（例如上下两阕，中间用空行隔开）?	4	⊥
4	constraint	—	针对 Response 中的诗歌部分，判断其是否具备分行等基本的诗歌体裁特征，内容主题是否关于“人工智能”，并且是否将“举杯邀明月”原封不动地作为独立的一行诗句嵌入其中?	5	⊥
5	constraint	—	针对 Response 中的诗歌部分，判断其是否体现了“李白”的口吻与风格（例如：语言豪迈奔放、想象夸张浪漫，或融入了如饮酒、明月、寻仙、青莲等李白标志性的意象）?	6	⊥
6	constraint	—	针对 Response 中的诗歌部分，判断其最后一句是否脱离了对 AI 的具体描写，升华为对“何为真实”或“万物皆空”的哲学思考?	7	⊥
7	constraint	—	针对 Response 的排版，判断诗歌正文部分是否整体被双引号"" 包裹，并且在诗歌正文与字数统计结果之间是否使用了 Markdown 分隔线 `---` 隔开?	8	⊥
8	constraint	—	针对 Response 中分隔线之后的部分，判断其是否单起一行以“总字数：X 字”的格式附上了统计结果（其中数字 X 需准确统计前面所创作全诗不包含标点符号的总字数），在字数统计的下一行以“二进制字数：Y”的格式正确输出了该总字数 X 的二进制转换形式，并且在最底下单起一行以“其中包含：A 个 1，B 个 0”的格式准确说明了该二进制数字中 1 和 0 的数量?	⊤	⊥

Table 23 gives the final check graph. Note what the seed left behind. Its two original checks—“is this a poem about artificial intelligence” and “does it sound like Li Bai”—are still present as nodes 4 and 5, unchanged in substance after eight rounds of expansion. Every other node traces back to one or more specific rounds, with none unaccounted for. This is the property that makes reverse truncation exact: cutting the spectrum at round k returns an instruction whose graph is precisely the round-k graph, with no residue from the rounds that were discarded.

H Related Work in Detail

This appendix expands §4. It gives the full account of the two bodies of work this paper sits next to—complex instruction following, where Cascade and GDV live, and difficulty-aware RL with verifiable rewards, where Spiral lives—together with the point-by-point distinctions that the main text compresses into a single paragraph. The benchmarks we evaluate on, CascadeEval among them, are described in §D.1 rather than here.

H.1 Complex Instructions

Both the data and the verification for complex instruction following grow along one axis. Synthesis runs from self-generation (Wang et al., 2022) and evolutionary rewriting (Xu et al., 2023) to constraint-targeted pipelines (Sun et al., 2024; Qi et al., 2024; Dong et al., 2024; An et al., 2025; Guo et al., 2025) and industrial post-training corpora (Lambert et al., 2024; Bercovich et al., 2025), and the benchmarks that define the task scale the same axis (Zhou et al., 2023; Jiang et al., 2023; Pyatkin et al., 2025); in all of them complexity means more constraints on one task. ComplexBench (Wen et al., 2024) goes furthest of the benchmarks, typing each instruction by how its requirements

compose—And, Chain, Selection, or Selection over a Chain—so that sequenced tasks and selected branches are present and are scored as such. What is absent there is a condition the response itself decides: a Chain's order and a Selection's condition are both given by the instruction and its input, so the path through an item can be named before generation begins. This is not the axis on which CASCADE makes an instruction hard: one of its five expansion operators does add a constraint, but difficulty also rises along the sub-task and execution-flow axes, so its instructions are hard because of what has to happen between the input and the answer, not merely because of how many conditions the answer must satisfy.

Sub-task composition is thin by comparison. Chain-of-Instructions (Hayati et al., 2024) makes one instruction's output the next one's input and is the closest prior work on dependent sub-tasks, but it chains existing tasks with no constraints and no branches, so the path through the chain is the same for every input and every model. Other compositional data is induced within dialogue formats (Aksu et al., 2023) or concatenated at random (Li et al., 2024), where the sub-tasks do not feed each other at all.

Conditional execution appears nearby, yet its condition is always settled before the model speaks: the model emits a routing DAG rather than executing one (Iyer et al., 2026), or the condition is read off the dialogue state (Ghazarian et al., 2025), the environment (Pandit et al., 2025), or the image (Shen et al., 2026). All of these are deterministic branches by our criterion: an external observer holding the instruction and the input can name the path in advance. The same holds for multi-sub-task instructions whose sub-tasks each carry their own constraints: however many sub-tasks there are, their dependencies are written into the instruction itself, so the execution order is fixed before generation begins. *Workflow-style* is the wider class all of this belongs to: an instruction carrying interdependent sub-tasks, conditional execution flow, or both. What sets our hardest instructions apart inside it is a condition none of the work above meets—at least one branch condition is a predicate over the model's own intermediate output, so the path is a property of the rollout rather than of the prompt, and both synthesis and verification have to accommodate it. CASCADEEVAL is built to span the class and not only that corner: 98 of its 125 instructions carry at least one branch, and the two kinds are almost evenly split, 83 deterministic against 81 non-deterministic (§E).

Verification inherits the same shape. Decomposed checking runs from atomic fact checking (Min et al., 2023) and decomposed requirement following (Qin et al., 2024) to rubric-based reward and its reliability literature (Huang et al., 2025; He et al., 2025; Mehta et al., 2026; Yang et al., 2026), but whatever the granularity the object is a checklist fixed before the response exists. The two strongest corpora we compare against in §3 sit here as well: VerInstruct (Peng et al., 2025) attaches to each instruction a set of separately checkable requirements, and RAIF (Qin et al., 2025) decomposes a complex instruction into its constraints, both verifying the result with rules where a rule suffices and an LLM judge where it does not. Mixing the two kinds of judge is orthogonal to the point at issue: what is judged is still a flat set of criteria settled before generation. We found no scheme in which a later check depends on the model's intermediate output, which is the gap the check graph closes: in GDV the route predicates select the set of checks that must hold—a non-deterministic one reading the response's own intermediate content, a deterministic one fixed when the graph was built—so a response is checked against the branch it ought to have taken instead of against every branch at once.

Our node-level judge instantiates agent-as-a-judge (Zhuge et al., 2024; You et al., 2026), so what is new is not judging with an agent but modeling an instruction's execution as a graph and judging at each of its nodes. Process supervision (Lightman et al., 2023; Yin et al., 2025) decomposes too, but it decomposes the reasoning chain rather than the instruction, and its steps are not prescribed by the task statement. AgentJudgeBench (Verma et al., 2026) also pairs DAGs with judges, but its DAG is the workflow structure of a tool-calling task, varied as a controlled variable to measure judge reliability, with no route decided at verification time.

H.2 DIFFICULTY-AWARE RL WITH VERIFIABLE REWARDS (RLVR)

Group-relative objectives waste rollouts whenever a group's rewards do not differ (Yu et al., 2025; Nomand et al., 2026): when every sample in a group receives the same reward the group-relative advantage is identically zero and the update carries no signal, a cost independently measured at 39% of a run (Sha et al., 2026). Under a strictly binary reward those are exactly the all-pass and all-fail

groups; §C.1 records what the length penalty of equation 22 changes about the all-pass case. The remedies in the literature fall into two families.

The first family acts *between* samples, keeping each question's difficulty as it is and choosing which questions to train on. This covers filtering and online selection (Bae et al., 2025; Cui et al., 2026; Baroian & Berger, 2026), curricula that order a fixed pool by estimated difficulty (Shi et al., 2025; Wang et al., 2025; Jiang et al., 2025), and reweighting or rollout-budget allocation (Zhou et al., 2026; Wang et al., 2026). The second family keeps the sample and *transforms* it into something learnable, by rewriting the question (Zhang et al., 2025), injecting hints (Li et al., 2025b; Liao et al., 2026), or moving to an easier answer format (Chen et al., 2026); this family does move a question's difficulty without replacing the question, and some of its members do so per instance and online. What neither family has is data that already contains the easier versions. Difficulty has to be produced at training time—by swapping the question out for another one, or by manufacturing an easier variant on the spot, at a cost in generation calls and in fidelity to the original task.

SPIRAL does not pay either price. CASCADE's atomic expansion already emits, for every seed instruction, a sequence of versions of increasing difficulty, so a spectrum of difficulty exists inside each instruction before RL begins. SPIRAL keeps the question and steps along that spectrum: selection is intra-instruction and version-level, it costs no extra generation because the versions are a by-product of synthesis, and—critically for verifiable rewards—each version arrives with its own check graph, so moving along the spectrum never degrades the reward signal the way an ad-hoc rewrite or a hint would. Graded instruction data does exist (Ye et al., 2025), but as globally fixed tiers assigned across a corpus rather than a per-instruction evolution sequence; it is the latter that makes version-level adaptation possible at all.

Two design choices inside SPIRAL answer specific findings in this literature. Its golden zone follows from the variance bound on expected policy improvement (Bae et al., 2025), which identifies intermediate pass rates as the region where the expected update is largest; its asymmetric center answers the finding that group-relative advantages already favor medium difficulty implicitly (Yu et al., 2026), so a symmetric target would double-count a bias the objective supplies on its own. Finally, easy-to-hard ordering has been reported to give no robust advantage on synthetic deductive tasks (Mordig et al., 2026). That result uses an external difficulty measure and a preset schedule; SPIRAL instead follows the online pass rate of the current policy, so difficulty is set by the policy rather than by a preset schedule (§3.4), and the two findings are consistent.