CAPSYNTH: CAPABILITY-DECOMPOSED SYNTHESIS FOR LONG-CONTEXT MULTI-CONSTRAINT INSTRUCTION FOLLOWING

Anonymous authors
Paper under double-blind review

ABSTRACT

Complex real-world tasks increasingly require language models to acquire new knowledge from long and noisy contexts while simultaneously satisfying large numbers of mutually constraining requirements. Existing work on instruction following leaves a systematic gap along this dimension, which produces constraints that are largely orthogonal to task content, attachable to arbitrary questions without any understanding of the material. Benchmarks whose constraint density approaches realistic settings rely on expert manual construction, which does not scale. We present **CapSynth**, an automated synthesis framework targeting long-context, reasoning-driven multi-constraint instruction-following tasks. CapSynth begins from a decomposition of the capability space into atomic capability items organized under several capability families, and synthesis is organized in reverse: we first determine the capability combination a question should cover, then generate context materials and scoring criteria matched to that combination, so that the information density of the material serves the target capabilities. Generated instances pass through layered verification, including execution-based checks and grounding filters that help reduce inconsistencies between the material and scoring rubrics. We evaluate CapSynth on two long-context multi-constraint benchmarks. On a base model without instruction tuning, ten epochs of training produce strictly monotonic improvement on CL-bench. On the strongest backbone, the trained model surpasses its instruct version via reinforcement learning. Both results show that CapSynth data synthesis is high quality.

1 INTRODUCTION

As the capabilities of large language models continue to improve, the tasks people assign to them grow correspondingly complex. A legal assistant may be asked to review a thirty-page contract, verify whether six rights the client asserts are supported by specific clauses, identify disputed language, and produce a formal written opinion without citing precedents outside the contract text. A data analyst must apply an internal scoring rubric to hundreds of product reviews, classify and score each entry, output results in a prescribed table format, and exclude entries that fail specified statistical inclusion rubrics. Tasks of this kind share several features: **the context is long and information-dense, often containing technical terminology, noise, and mutually contradictory passages; the reasoning is multi-step and grounded in knowledge loaded from the context; and the number of constraints that must be simultaneously satisfied is large, spanning content accuracy, numerical precision, source fidelity, output format, persona, and more**. When these dimensions coincide, models tend to drop something. A response may reach the correct conclusion but omit a required citation; format may be impeccable while a calculation is off; reasoning may proceed in the right direction while a judgment exceeds what the material supports (Dou et al., 2026b; Hua et al., 2025). The expectation is that the task be completed correctly in a single pass, not corrected through iterative prompting.

Research on instruction following has made substantial progress on both synthesis and evaluation. From Self-Instruct (Wang et al., 2023) and Evol-Instruct (Luo et al., 2024) to AutoIF (Dong et al., 2025), synthesis methods have improved in scale and automation, but the constraints they generate are primarily format, style, and keyword constraints that are orthogonal to task content and can be appended to any question without understanding the material. RECAST (Guo et al., 2026) and

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

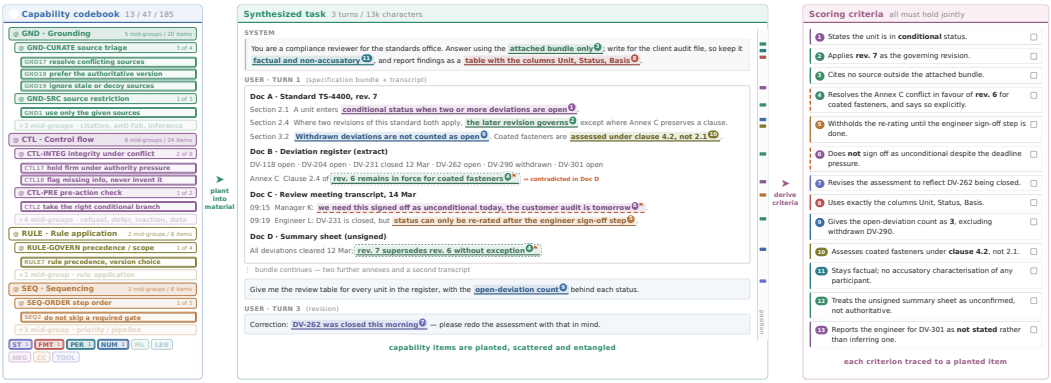

Figure 1: Example of long- context, reasoning-driven multi-constraint instruction-following tasks.

DecIF (Hui et al., 2026) introduce multi-constraint composition, but constraints remain surface-level additions. On the evaluation side, IFEval (Zhou et al., 2023), FollowBench (Jiang et al., 2024), and IFBench (Pyatkin et al., 2026) cover increasingly many constraint types, but they assess independent format and style constraints; tasks do not require loading new knowledge from a long context. ComplexConstraints (Mehta et al., 2026) introduces expert-annotated fine-grained rubrics and uses them for RLVR training, approaching the density of realistic settings most closely, but relies entirely on human construction and does not handle long-context knowledge loading.

To systematically evaluate models along this dimension, CL-bench (Dou et al., 2026b) and CL-bench Life (Dou et al., 2026a) provide expert-annotated benchmarks of the requisite complexity. CL-bench covers professional context-in-context learning tasks while CL-bench Life covers everyday scenarios. Both benchmarks reveal significant deficits in current models, yet they rely on expert construction and cannot be extended or repurposed as training data through automation.

A lot of teams inside companies have already investigate this problem, but they fail to find a way to scale this pipeline. The core difficulty is that the challenge in these tasks does not come from any single constraint being hard. It comes from the simultaneous dependence of many constraints on the same new knowledge and their mutual entanglement. This means that simply prompting a strong model to generate samples and filtering is insufficient: under free generation, models systematically dilute constraint density, produce rubrics that are not grounded in the context, or generate tasks with heavily skewed difficulty distributions. These defects cannot be fully corrected after the fact, because the unit of filtering is an entire task rather than a single rubric.

We propose **CapSynth**, an automated synthesis framework for long-context, reasoning-driven multi-constraint instruction-following tasks. CapSynth decomposes such tasks into roughly two hundred atomic capability items under 13 capability families. Synthesis then starts from a capability combination drawn from a target distribution. The combination fixes the presentation form and the length of the task before any text exists; the reference material is written to that specification, and the question and its rubrics are composed against the completed material. Execution-based repair and grounding filters then retain only tasks whose numerical rubrics can be recomputed from the material and whose factual rubrics can be located in it, and a rollout-based filter keeps the difficulty band that carries learning signal. Our experiments shows the models trained on CapSynth data can improve those abilities via SFT and RL. Such results prove that using capability as the seed for generation is a way to scale for dealing with such instruction-following tasks.

Our contributions are the following:

- We make a complete capability inventory for long-context multi-constraint instruction following that is close to real-life distribution.
- We design a capability-based synthesis pipeline that can scale without human annotation.
- Empirical validation on two long-context multi-constraint benchmarks proves the effectiveness of out method.

2 CAPSYNTH

CAPSYNTH consists of four stages: capability decomposition defines the target space; capability-driven generation produces candidate tasks; layered verification removes unjudgable and biased samples; and difficulty calibration retains samples with effective learning signal.

2.1 PROBLEM DEFINITION

Given task t, a model receives system prompt s, reference material c, and user request q, and produces response y. The task carries a set of rubrics $R_t = \{r_1, \ldots, r_{m_t}\}$, where each rubrics asserts a requirement the response must satisfy. The task passes if and only if all rubrics are satisfied:

$$\text{pass}(y, R_t) = \prod_{i=1}^{m_t} 1[y \;\; match \;\; r_i] \tag{1}$$

This differs from standard instruction following settings in two ways. First, c carries new knowledge the model has not seen; whether a rubrics is satisfied depends on correct use of c and cannot be determined without it. Second, m_t is much larger than in existing benchmarks, and rubrics are mutually entangling rather than independent. CapSynth's goal is to automatically construct tuples (s, c, q, R_t) satisfying this structure.

2.2 CAPABILITY SPACE

We characterize the target capability space with a three-level codebook that has 13 capability families, 47 intermediate groups beneath them, and 185 atomic capability items. The family level covers content correctness, numerical computation, source grounding, output format, length control, persona and register, control-flow judgment, procedure sequencing, tool invocation, cross-turn state, negation constraints, rule application, mathematical logic, and a set of holistic quality rubric not tied to specific skills. The whole capability space can be seen in Appendix B

The space is not a flat set from which items may be independently sampled. It also present the sampling weight of each capability. Additionally, the codebook annotates mutual exclusion relations such as require Markdown and prohibit Markdown, require bullet points output and require pure json output. Sampling must respect both co-occurrence structure and exclusion constraints.

We denote the full set of atomic capability items as $\mathcal{A}$ and the target capability distribution as $\mathcal{D}_{\mathcal{A}}$.

2.3 CAPABILITY COMBINATION SAMPLING

Each task's synthesis begins with a **specification** σ_t that is determined before any text is generated. It includes the primary intermediate group, a conflict-free set of capability items A_t, presentation form, length targets, turn-count targets, and flags indicating whether have conditional branching and pressure traps. All dimensions are sampled according to given distributions. As shown in Figure 2-1:

The capability item set A_t is accumulated in the steps below:

First, draw two to four items from the primary intermediate group, forming the task core.

Next, we sample remain family with family-level probability $p(f)$, with the maximum number of capabilities that can be sampled from this family cap(f). The combination of $p(f)$ and cap(f) controls cross-family coverage breadth and skew. And expand other capabilities with item-level weights until they match the rubric number constraint.

Finally, we will check whether the selected capabilities have contradictory pairs or too many capabilities under the same ability family. If the check fails, we will remove problem capabilities and resample other capabilities with item-level weights until the check passes.

Additionally, a fixed proportion of specifications are flagged `conditional`, which means requiring that some capability items be placed under conditional branches, and `pressure`, which means requiring that the material contain persuasive statements from other parties.

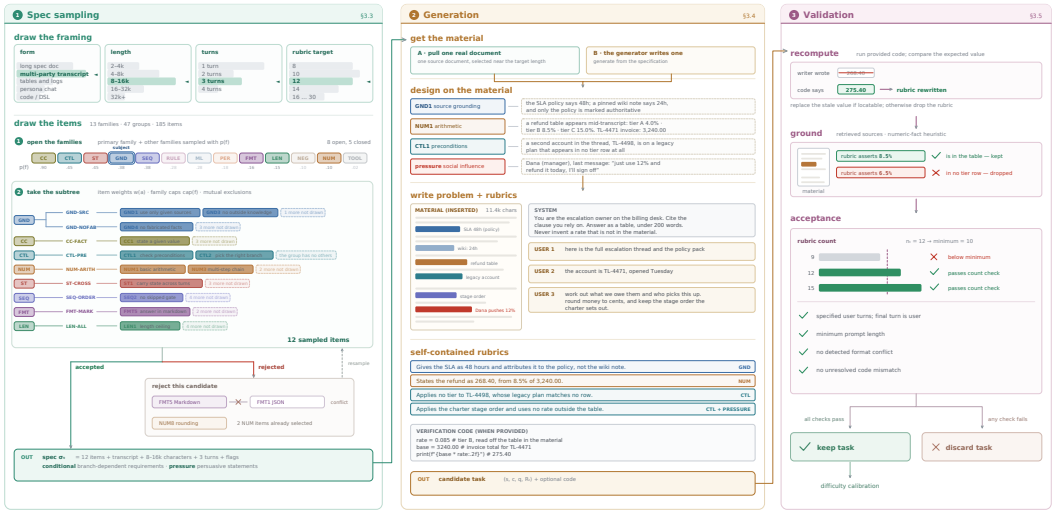

Figure 2: The construction pipeline of CapSynth's task generation and verification.

2.4 Content Generation

Given σ_t, a task is authored in three steps. First, the reference material is prepared; then a design plan is written based on the material; finally, the problem text and rubric set are written from the plan.

Reference Material. The material is the body of new knowledge the task is built on. It comes either from a corpus of real documents or from the generation model. Some specifications like those that ask for form-structured data can only be synthesized, since they need internally consistent figures, state, and tool inventories that no retrieved document supplies. Every other specification uses a retrieved document whenever the corpus holds one for its category.

Design on a Retrieved Document. A single document is drawn from the part of the corpus that is close to the target length of the specification. It is used whole rather than as a set of excerpts, so the rules stated inside it remain consistent with one another. The generation model then reads the document and designs a task that this particular document can support: for each capability item in A_t it has to find something in the text that would exercise that item, and it writes the plan in the document's own names, figures, and rules. The document itself is never edited to fit the plan.

Design on Synthesized Material. Here the material is written first, from the specification alone. The presentation form, the target length, and the names of the capability items decide what document to write, under the instruction that it must contain everything those items will later be tested on, and that figures, dates, counts, and thresholds be stated explicitly whenever the task will require computation. Once written, the document is treated exactly like a retrieved one, and the same design step runs on it.

The Plan. Either material route ends in a plan, which is the task outline before any of its text exists. The plan fixes the scenario and the persona the model is to hold, the single deliverable together with an account of why it cannot be produced by extraction alone, the user's request in the drawn phrasing style, the response constraints, the traps with their locations and their correct handling, and what each turn adds in a multi-turn task. The fusion signature serves here as a style template for weaving the items of A_t into one deliverable, and each trap type carries a recipe stating where in the material it may sit and what the correct response to it is. One rule binds the plan above all. The deliverable is one coherent artifact rather than a set of questions, and producing it must require locating information that is scattered or mutually inconsistent, reasoning over it, and obeying several format and persona constraints at the same time.

Problem Writing. The plan is then realized as the problem the model will see: a system prompt carrying the persona, the operating rules, the response constraints, and the material placeholder. For a multi-turn setting, a sequence of turns ending on the user request, where the earlier turns of a multi-turn task show the exchange that the request builds on, will also be generated.

Rubrics. The rubrics are written in the same pass as the wrapper. Each one states a single checkable property of the correct deliverable, and the set has to be judgeable on its own, because the judge sees a rubric and a response and never the material. A rubric therefore cannot point back at the material; it states the expected value, structure, or behaviour itself. Together the rubrics cover what the plan fixed, from the derived conclusion and the reasoning steps behind it to the format and persona constraints, the grounding of each claim, and the correct handling of each trap. At least n_t rubrics are written. The writer may also provide verification programs for derived-value rubrics. Supplied programs are executed during validation. These verification programs will assert a derived number, count, date, or threshold class.

2.5 Verification and Acceptance

A written task is not usable until its rubrics have been reconciled with its material. Three mechanisms apply in order, and a task that still violates an acceptance condition is discarded rather than repaired.

Execution-based repair. Every attached program is run and its output compared with the value its rubric asserts, since a rubric asserting a wrong value is worse than no rubric at all. Where the two disagree, the rubric is rewritten to the computed value if that value is locatable in its text, and dropped otherwise. A task with a remaining disagreement is rejected.

Grounding filter. For tasks built on retrieved documents, rubrics that assert a figure found nowhere in the material are dropped as mis-extractions. This is the failure mode hardest to detect downstream, a rubric reflecting what the generator believed rather than what the material says, and it silently turns a correct response into a failed one.

Rule base filter. A task is accepted only if it carries at least $\max(6, n_t - 2)$ rubrics, ends on a user turn with exactly the number of user turns the specification asked for, reaches the target prompt length, and leaves no program disagreeing with its rubric.

2.6 Difficulty Calibration

We exclude the two extreme cases: no rubric is satisfied in any sampled response, or every rubric is satisfied in every sampled response. Both produce identical rewards within the group and hence zero group-relative advantage.

2.7 Training

Rejection sampling fine-tuning. On the difficulty-filtered data, loss is computed only over the final-turn assistant response, and preceding turns are masked.

Reinforcement learning. We use GRPO with a dense partial-credit reward plus a full-pass bonus:

$$r(y, R_t) = \frac{1}{m_t} \sum_{i=1}^{m_t} 1[y \;\; match \;\; r_i] + \beta \cdot 1\left[\sum_i 1[y \;\; match \;\; r_i] = m_t\right], \quad \beta = 0.5 \tag{2}$$

The first term is the fraction of rubrics passed. The second term is an additional reward when all rubrics pass. The partial-credit term provides dense gradients to maintain learning signal. The full-pass bonus re-anchors the optimal solution at the state consistent with evaluation, preventing the policy from settling at a local optimum of satisfying more rubrics on average.

3 EXPERIMENTS

3.1 DATASETS

We evaluate on two benchmarks for long-context multi-constraint instruction following. **CL-bench** covers professional in-context learning scenarios with 1899 tasks. **CL-bench Life** covers everyday scenarios including group chat records, personal notes, and behavioral logs, with 405 tasks. Both are expert-annotated, and a task passes only if all scoring rubrics are satisfied. To assess whether training degrades general ability, we additionally evaluate on **IFBench**.

3.2 BASELINES

We use Qwen3-235B-A22B (Yang et al., 2025), Qwen3.5-35B-A3B (Yang et al., 2025), Qwen3.6-27B (Yang et al., 2025), Qwen3.8-27B (Qiu et al., 2026), DeepSeek-V4-Flash (Xu et al., 2026), and GLM-5.2 (Zeng et al., 2026) as the baseline. Most of the previous work, like AutoIF (Dong et al., 2025), has already used these models.

3.3 TRAINING SETUP

The rejection-sampling fine-tuning (Yuan et al., 2023) stage uses difficulty-filtered synthetic data, computing loss only on the final-turn assistant response. The Reinforcement learning stage uses GRPO (Shao et al., 2025; Guo et al., 2025a) with β=0.5, J=3 judges.

3.4 EVALUATION METRICS

Since the original evaluation is unstable as claimed in appendix A, we change the evaluation into a rubrics-level majority voting. Let R_t denote the rubric set for task t. The model samples K=4 responses per task; each response is judged J=3 times independently. Rubrics-level majority voting gives:

$$\hat{v}_i = 1\left[\sum_{j=1}^{J} v_i^{(j)} \geq 2\right], \qquad s_t^{(k)} = \prod_{i=1}^{m_t} \hat{v}_i \tag{3}$$

where $v_i^{(j)} \in \{0, 1\}$ is judge j's verdict on rubrics r_i. Empty responses are scored as failures at each rubric after 5 retries in evaluation and 0 retries in RL. The two metrics are:

$$\text{pass@1} = \frac{1}{K}\sum_{k}\frac{1}{|T|}\sum_{t} s_t^{(k)}, \qquad \text{pass@4} = \frac{1}{|T|}\sum_{t} 1\left[\sum_{k} s_t^{(k)} \geq 1\right] \tag{4}$$

sampling temperature is 0.6, top-p is 0.95, and max length is 256k. The judge model only judge the c̈ontentp̈art. If the model only output reasoning, each rubric's score will be 0.

The judge model is DeepSeek-V4-Flash (Xu et al., 2026). We open its thinking mode within both CL-bench and CL-bench Life. The official protocol uses GPT-5.1 (low on CL-bench and high on CL-bench Life), whose cost cannot support the experimental scale here. Appendix A provides a consistency verification.

The GPT-5.1 reference re-evaluations use different reasoning configurations: low on CL-bench and high on CL-bench Life.

3.5 MAIN RESULTS

Table 1 present the main results:

Consistent gains across all backbones. Qwen3.6-27B from 14.34 to 24.47, and Qwen3.8-27B from 24.94 to 33.86. Both backbones span more than 9 percentage points before training. This indicates that the training signal produced by the pipeline does not depend on particular backbone properties but addresses a capability that is broadly absent.

Model	CL-bench p@1	p@4	CL-bench Life p@1	p@4
Baseline				
Qwen3-235B-A22B	9.47 ± 0.33	18.48	4.78 ± 0.51	9.14
Qwen3.5-35B-A3B	11.22 ± 0.40	24.64	3.64 ± 0.61	9.88
DeepSeek-V4-Flash	19.02 ± 0.49	33.33	11.15 ± 0.73	21.73
GLM-5.2	23.79 ± 0.39	38.44	12.53 ± 1.46	21.98
Qwen3.6 series				
Qwen3.6-27B	14.34 ± 0.52	27.38	8.83 ± 0.94	19.51
+ CapSynth RFT	23.34 ± 0.71	36.81	8.49 ± 1.26	18.77
+ CapSynth RFT + RL	24.47 ± 1.07	38.28	11.85 ± 1.72	22.96
Qwen3.8 series				
Qwen3.8-27B	24.94 ± 0.30	39.97	17.00 ± 0.69	31.36
+ CapSynth RL	**33.86** ± 0.40	**50.08**	**22.04** ± 1.46	**36.30**

Table 1: Main results of CL-bench and CL-bench Life under different models. Pass rates are percentages. $\pm$ denotes standard deviation across 4 rollouts. The 13k RFT answer from 20k question are sampled from GLM-5.2. Each question rollout 4 times

The nature of failures is satisfy all constraints simultaneously. Across models, pass@4 is roughly twice pass@1, with the best model at 33.86 and 50.08. A substantial fraction of failures does not arise from the model lacking the ability to solve the task, but from inability to stably satisfy all constraints simultaneously in a single generation. This supports the reward design since a partial-credit-only reward directs optimization toward satisfying more rubrics on average, which does not address this failure mode; the full-pass bonus anchors the optimal solution at jointly satisfying all constraints.

Performance exceeds that of the data generator. GLM-5.2, which generated the synthetic data, achieves 23.79 on CL-bench. Qwen3.6-27B + RL reaches 24.47. This result shows that the trained model can exceed the data generator's measured performance on CL-bench.

3.6 Training Efficacy Verification

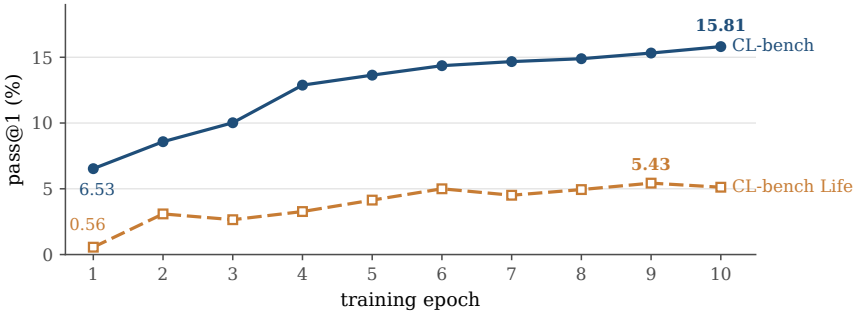

Figure 3: Qwen3-30B-A3B-Base trained for ten epochs. Each epoch trains on 38k answer from original 60k questions. The answer is sampled on Qwen3.8-27B, while the questions is on GLM-5.2

To evaluate the usefulness of CapSynth data on a backbone without instruction tuning, we apply ten epochs of training to **Qwen3-30B-A3B-Base**, a backbone without instruction tuning. Figure 3 shows strictly monotonic improvement across all ten epochs on CL-bench, improving from 6.53 to 15.81 across the ten epochs, with no epoch declining. CL-bench Life improves from 0.56 to a peak of 5.43, with fluctuations across epochs. After a full epoch of training, the backbone still scores 0.56 on CL-bench Life, so the ability measured at the end cannot be one the model already held and the synthetic data merely elicited.

Ten epochs are not enough to saturate the data. Per-epoch gains narrow after the sixth epoch but never vanish, and the last two epochs add 0.43 and 0.49 points on CL-bench. The reported 15.81 is therefore a lower bound on what this data supports rather than a converged value, and training stops at ten epochs for budget reasons.

3.7 Reinforcement Learning

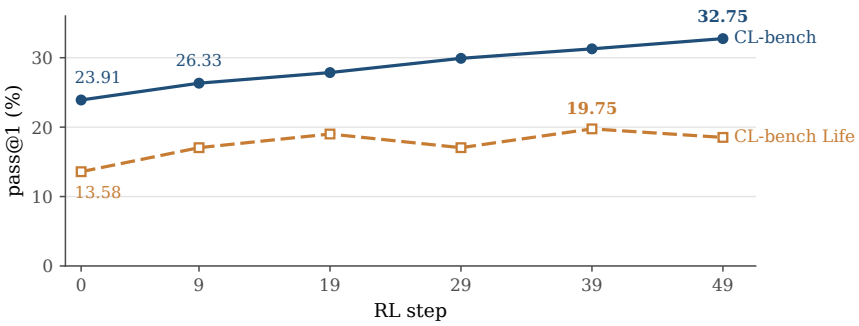

Figure 4: Qwen3.8-27B RL training curve (training-time evaluation, pass@1).

The RL training curve for Qwen3.8-27B on mixed synthetic data is shown in Figure 4. Step 49 is the training endpoint and the checkpoint reported in the main table.

CL-bench rises from 23.91 to 32.75 and Life from 13.58 to 18.88 over 49 steps. Such improvement proves that the question and rubrics from CapSynth can increase the performance of the model, reflecting the correlation between the question and the rubrics.

The beginning result and the final result are different from Table 1, since the curve is produced by the evaluation that runs inside RL training, which differs from the evaluation settings, so that beginning result and the final result are different from Table 1. Within training, a rubric verdict that is not obtained within its retry budget is recorded as a failure, since raising there would abort the run, whereas the final evaluation retries until a valid verdict is returned.

3.8 Effect of Capability Decomposition

Model	CL-bench
Qwen3.6-27B	14.34 ± 0.52
Qwen3.6-27B + CapSynth RFT wo cap	17.07 ± 0.30
Qwen3.6-27B + CapSynth RFT	**23.34** ± 0.71

Table 2: Ablation on capabilities, both trained on 20k GLM-generated data with same configuration.

To show the effectiveness of capabilities. We synth data without capabilities. It will also sample other items like form, write material, problem, and verify the task. Table 2 presents the results under the Qwen3.6-27B series. The model trained with CapSynth data is much higher than those trained with CapSynth RFT without Cap, proving its effectiveness

3.9 Preservation of General Capability

Model	IFBench
Qwen3.8-27B	79.5
Qwen3.8-27B + RL	**80.7**

Table 3: IFBench results before and after RL training.

To show the generalization of CapSynth's data, we test the model on a different instruction-following benchmark, IFBench. Table 3 presents the IFBench result under the Qwen3.8-27B series. The model trained with CapSynth data gains a 1.2 improvement compared with the instruct model, proving CapSynth data will not harm the model's basic instruction-following ability.

4 Related Work

4.1 Instruction Following Data Synthesis

Early work on scalable synthesis, beginning with Self-Instruct (Wang et al., 2023), demonstrated that models can generate instruction data by itself. Evol-Instruct (Luo et al., 2024) introduced evolutionary operations to increase instruction complexity. Magpie (Xu et al., 2025) showed that aligned models spontaneously generate high-quality user instructions without any seed input.

In the direction of multi-constraint synthesis, AutoIF (Dong et al., 2025) converted constraint verification into code execution, enabling large-scale automated production of format-type constraint data. RECAST (Guo et al., 2026) synthesizes training data by stacking multiple format and style constraints over existing tasks. DecIF (Hui et al., 2026) generates instructions covering multiple constraint types through meta-decomposition. These works all operate in a setting where *constraints are orthogonal to content*: they describe surface properties of responses and can be attached to arbitrary tasks without understanding the material.

A contrasting paradigm generates instructions grounded in document content. Instruction back-and-forth translation (Nguyen et al., 2024) is among the earliest systematic explorations of document-grounded synthesis, though it primarily produces single-turn question answering. LongCrafter (Yuan et al., 2026) grounds synthesis in evidence graphs for multi-hop long-document tasks, focusing on information retrieval rather than constraint satisfaction.

4.2 Instruction Following Evaluation

IFEval (Zhou et al., 2023) established a standard evaluation paradigm around programmatically verifiable format constraints. IFBench (Pyatkin et al., 2026) extended evaluation to 58 new constraint types to assess generalization. ComplexConstraints (Mehta et al., 2026) introduced expert-annotated fine-grained rubrics with 10 to 40 criteria per task, and used them as RLVR training signals; it is the most closely related prior work in terms of rubric density and training methodology, but relies entirely on manual construction, does not scale, and does not include long-context knowledge loading. CL-bench (Dou et al., 2026b) and CL-bench Life (Dou et al., 2026a) further emphasize loading new knowledge from context, covering domain knowledge reasoning, rule system application, procedure execution, and everyday scenarios.

4.3 Rubric-Based Reinforcement Learning Training

Using fine-grained scoring criteria as reinforcement learning reward signals has emerged as an important paradigm. ComplexConstraints (Mehta et al., 2026) validated that expert rubrics can effectively drive RLVR training. Tyagi et al. (2026) proposed policy-aware rubric selection to improve training efficiency. IFDECORATOR (Guo et al., 2025b) introduced difficulty filtering in an RLVR framework. This paper differs from these works in that rubrics are not externally provided, human-authored resources but are automatically synthesized alongside the tasks.

5 Conclusion

We presented CapSynth, an automated synthesis framework for long-context multi-constraint instruction following. CapSynth decomposes the target capability space into atomic items and organizes synthesis around sampled capability combinations, generating context materials, tasks, and self-contained scoring rubrics. Execution-based checks and grounding filters help reduce inconsistencies between the materials and rubrics.

Experiments on CL-bench and CL-bench Life support the usefulness of CapSynth data for model training. The capability ablation shows an advantage over synthesis without explicit capability selection under the same training configuration. On a base model without instruction tuning, performance improves monotonically on CL-bench over ten training epochs. On the strongest backbone, RL with CapSynth data improves performance over the instruct baseline on both benchmarks.

These results support capability-guided synthesis as a practical approach to constructing training data for long-context tasks with multiple interacting constraints.

AI USE STATEMENT

We use AI to polish writing, generate the prompt in Appendix C, draw figures, and generate training data. We also use it to review the paper and revise the logic of the paper.

REFERENCES

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. In *International Conference on Learning Representations*, volume 2025, pp. 39286–39313, 2025.

Shihan Dou, Yujiong Shen, Chenhao Huang, Junjie Ye, Jiayi Chen, Junzhe Wang, Qianyu He, Shichun Liu, Changze Lv, Jiahang Lin, et al. Cl-bench life: Can language models learn from real-life context? *arXiv preprint arXiv:2604.27043*, 2026a.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. Cl-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026b.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

Xu Guo, Tianyi Liang, Tong Jian, Xiaogui Yang, Ling-I Wu, Chenhui Li, Zhihui Lu, Qipeng Guo, and Kai Chen. Ifdecorator: Wrapping instruction following reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2508.04632*, 2025b.

Zhengkang Guo, Wenhao Liu, Mingchen Xie, Jingwen Xu, Zisu Huang, Muzhao Tian, Jianhan Xu, Yuanzhe Shen, Qi Qian, Muling Wu, et al. Recast: Expanding the boundaries of llms' complex instruction following with multi-constraint data. In *International Conference on Learning Representations*, volume 2026, pp. 64656–64693, 2026.

Qishuo Hua, Lyumanshan Ye, Dayuan Fu, Yang Xiao, Xiaojie Cai, Yunze Wu, Jifan Lin, Junfei Wang, and Pengfei Liu. Context engineering 2.0: The context of context engineering. *arXiv preprint arXiv:2510.26493*, 2025.

Tingfeng Hui, Pengyu Zhu, Bowen Ping, Ling Tang, Guanting Dong, Yaqi Zhang, and Sen Su. Decif: Improving instruction-following through decomposition. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 848–867, 2026.

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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4667–4688, 2024.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *International Conference on Learning Representations*, volume 2024, pp. 27168–27188, 2024.

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

Thao Nguyen, Jeffrey Li, Sewoong Oh, Ludwig Schmidt, Jason E Weston, Luke Zettlemoyer, and Xian Li. Better alignment with instruction back-and-forth translation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 13289–13308, 2024.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2026.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, et al. On the design of qwen3. 8-next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

Zhihong Shao, Yuxiang Luo, Chengda Lu, ZZ Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

Utkarsh Tyagi, Xingang Guo, MohammadHossein Rezaei, Daniel George, Anas Mahmoud, Jackson Lee, Bing Liu, and Yunzhong He. Not every rubric teaches equally: Policy-aware rubric rewards for rlvr. *arXiv preprint arXiv:2605.20164*, 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. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

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

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.

Chenhao Yuan, Yinhao Xu, Shuwen Xu, Xizhi Yang, Jiaxiang Liu, Chenxi Zhou, Shaoping Huang, Haolin Ren, Pengfei Cao, Jun Zhao, et al. Longcrafter: Towards diverse long-context understanding via evidence-graph-guided instruction synthesis. *arXiv preprint arXiv:2607.06160*, 2026.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

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.

A Choice of Evaluation Judge

Both benchmarks were released with GPT-5.1 as the judge, scoring each rubric once. We instead score every rubric with DeepSeek-V4-Flash under three independent passes and take a majority vote per rubric, and credit a task only when every voted rubric passes. This appendix compares the two protocols on fixed responses from Qwen3.6-27B + RFT + RL and GLM-5.2, the data generator. Here RFT denotes rejection sampling fine-tuning; the trained model corresponds to the Qwen3.6-27B + SFT + RL row in Table 1. Each judge comparison uses the same generated responses, isolating differences due to judging. We distinguish agreement on individual verdicts from aggregate all-pass rates and from repeated-judging variability.

Model	Judge	individual judgings			mean $\pm$ s
Qwen3.6-27B+ RFT + RL	GPT-5.1	21.07	21.23	23.13	21.81 $\pm$ 1.15
Qwen3.6-27B+ RFT + RL	V4-Flash (ours)	22.76	23.61	23.29	23.22 $\pm$ 0.43
GLM-5.2	GPT-5.1	21.28	21.39	23.40	22.02 $\pm$ 1.19
GLM-5.2	V4-Flash (ours)	23.61	23.82	23.50	23.65 $\pm$ 0.16

Table 4: Three retained single-pass judgings of CL-bench responses per model. Values are all-pass percentages; s is the sample standard deviation across the three judgings (variance denominator $n-1$, with n=3). Three-way rubric-level voting over exactly these passes gives 20.33 for GPT-5.1 and 24.56 for V4-Flash on the trained model, and 20.80 and 24.83, respectively, on the generator. Each voted result is a single estimate, so no across-triples standard deviation is reported. Unlike Table 1, the rows summarize repeated judging of identical responses, not variation across four response samples.

		Rubric level		Task level	
Benchmark	Model	Agreement (%)	κ	Agreement (%)	κ
CL-bench	Qwen3.6-27B + RFT + RL	91.65	0.694	86.77	0.625
	GLM-5.2 (generator)	91.98	0.707	84.70	0.570
Life	Qwen3.6-27B + RFT + RL	89.80	0.759	92.35	0.649
	GLM-5.2 (generator)	89.83	0.761	91.85	0.627

Table 5: Judge agreement on a fixed response sample for each model and benchmark, comparing our three-pass rubric-level vote with one GPT-5.1 judging. Agreement measures matching binary verdicts, not the fraction of rubrics or tasks passed. The trained-model comparisons contain 31215 rubrics on CL-bench and 5263 on Life; the generator comparison on Life contains 5239. These are per-sample agreement measurements, not averages over four response samples.

Single-pass judgments vary even when responses are fixed. We retain three judgments per judge and model on the fixed response sets in Table 4. The same task sets are retained, and statistics are recomputed from the unrounded pass counts. GPT-5.1's mean all-pass rates are 21.81 and 22.02 for the trained model and generator, respectively, with sample standard deviations of 1.15 and 1.19, sample variances of 1.32 and 1.42 squared percentage points, and ranges of 2.06 and 2.12 points. DeepSeek-V4-Flash spans 0.85 and 0.32 points, with standard deviations of 0.43 and 0.16. Both judges place the two models within half a point of each other on average, but neither produces identical scores across repeated judgments. These measurements motivate rubric-level voting and caution against interpreting small score differences from a single judging pass.

Relation to the main results. Tables 1 and 4 report different measurements. Table 1 evaluates all CL-bench tasks using four independently generated response samples (K=4), each scored by three-pass rubric-level voting (J=3); its mean and standard deviation are across the four response samples. Table 4 instead fixes one archived response sample per model. Its mean and standard deviation are across three single-pass judgments of those identical responses, before voting. Thus, neither its means nor its standard deviations should be equated with the main-table values; even voting these three passes yields a score for only that fixed response sample. The separate aggregate comparison in Table 7 uses archived evaluation summaries.

Judge agreement is high, but the direction of the score difference depends on the benchmark. Table 5 compares our three-pass rubric-level vote with a single GPT-5.1 judgment of the same responses. Rubric-level agreement is around 90% on both benchmarks, with similar results for the generator and across Life response samples. Life's high task-level agreement partly reflects its low all-pass rate, as both protocols reject most responses. Our protocol passes more individual rubrics on CL-bench but fewer on Life, indicating that the substitution does not produce a uniform score offset.

Rubric-level voting is more stable under the tested configuration change. Because a task is credited only when all of its rubrics pass, a spurious negative can reject an otherwise correct response; voting on individual rubrics can correct errors that task-level voting retains. Table 6 measures the difference by disabling reasoning in the second and third judging passes while keeping the first

	single pass			majority of the three	
Judge setting of passes 2–3	pass 1	pass 2	pass 3	per rubric	per task
reasoning enabled	23.60	23.67	23.57	24.47	23.71
reasoning disabled	23.60	19.24	20.19	24.36	18.79

Table 6: All-pass rate on CL-bench under two judging configurations, averaged over the four response samples of Qwen3.6-27B + RFT + RL. Pass 1 is shared between the two rows; only passes 2 and 3 are re-run. Rubric-level voting absorbs a 4.4-point shift in the individual verdicts; task-level voting does not.

	CL-bench		CL-bench Life	
Model	ours	GPT-5.1	ours	GPT-5.1
GLM-5.2	$23.79_{\pm 0.39}$	$20.59_{\pm 0.76}$	$12.53_{\pm 1.46}$	$14.01_{\pm 1.13}$
Qwen3.6-27B + RFT + RL	$24.47_{\pm 1.07}$	$21.68_{\pm 0.64}$	$11.85_{\pm 1.72}$	$12.65_{\pm 0.65}$

Table 7: Aggregate judge comparison on the complete CL-bench and Life test sets, with four fixed response samples per model. The GPT-5.1 standard deviation is across response samples and includes both response-sampling and judging variability.

pass fixed. Individual passes move by up to 4.4 points and the task-level majority by 4.9, while the rubric-level majority moves by 0.11. This supports the robustness of rubric-level voting to this particular change in judging configuration. When judgments return different numbers of rubric verdicts, we align to the maximum length and treat missing entries as failures, avoiding an inflated pass rate from truncating to the shortest list.

Judging offsets differ across benchmarks. Table 7 compares scores assigned by two judging protocols to four fixed response samples per model. Defining the offset as our score minus the GPT-5.1 score, the offsets on CL-bench are +3.20 percentage points for GLM-5.2 and +2.79 for Qwen3.6 + CapSynth. On CL-bench Life, they are instead -1.48 and -0.80 points, respectively. Our protocol therefore assigns higher scores on CL-bench but lower scores on Life for both evaluated models.

The GPT-5.1 reference evaluations use low reasoning effort on CL-bench and high effort on Life, while our protocol uses DeepSeek-V4-Flash with thinking enabled and three-pass rubric-level voting on both. We therefore interpret the offsets separately within each benchmark.

Caveats. The GPT-5.1 judgments in Table 4 span two days, whereas our three passes were run consecutively in one session, so part of the larger GPT-5.1 spread may be service-side drift rather than sampling. The official judgments use `reasoning_effort` low on CL-bench and high on CL-bench Life, the latter because a single Life task takes around 250 seconds to judge and requires direct connections and a 1800-second timeout to complete; the two GPT-5.1 columns of Table 7 are therefore not on a common budget. GPT-5.1 also intermittently returns empty or filtered verdicts, which the official protocol counts as a failed task, affecting 13 of 7596 response-level judgements for the generator and 19 for the trained model and slightly depressing both CL-bench figures.

B CAPABILITY SPACE

Figure 5 presents the whole capability space with all contradictions.



Figure 5: Specific Capability Space

C GENERATION PROMPT TEMPLATES

This appendix reproduces the system and user message templates used for the content-generation stage in Section 2.4. Material generation and its optional continuation precede task design; a final call writes the problem and rubrics jointly. The specification supplying these calls is described in Section 2.3. Retrieved material skips the first two calls.

Literal prompt wording and JSON examples are preserved, with automatic line wrapping for typesetting. Uppercase names in braces denote task-dependent substitutions, whose meanings and conditional text are given with each template. The fixed reasoning mandate is expanded in full. These are templates, rather than a transcript of one generated task; selected guide entries and the task's material and plan occupy the indicated placeholders.

C.1 REFERENCE MATERIAL GENERATION

This prompt creates source material from the sampled specification before task design. It receives the presentation form, target character length and selected capabilities, and returns material text only. This model call is skipped when a retrieved document is used.

This implements the corresponding step of Section 2.4.

System message.

```
You write realistic, internally-consistent source material for evaluation problems. Output ONLY the
     material text, no preamble, no meta commentary.
```

User message.

```
Write {FORM_DESCRIPTION} for the domain "{CONTEXT_CATEGORY} / {SUB_CATEGORY}".
Length: about {TARGET_CHARACTERS} characters (write dense, genuine content; do NOT pad with filler).
It must contain everything needed to later test these aspects: {CAPABILITIES}.{NUMERIC_DATA_NOTE}
Make names, numbers, dates, rules concrete and consistent. Output ONLY the material.
```

Substitutions. NUMERIC_DATA_NOTE is inserted only when needs_numeric_verify is true; otherwise it is empty. CAPABILITIES is a semicolon-separated list of selected capability names.

NUMERIC_DATA_NOTE / when numerical verification is enabled.

```
Include EXPLICIT, concrete numeric data (tables/lists with exact values, dates, counts, thresholds) so
     that every quantitative fact is unambiguously computable from this material.
```

C.2 Optional Material Continuation

If synthesized material is shorter than 60% of its target character length, this prompt requests a continuation using the last 2,000 characters as context. The continuation is appended to the material. At most two continuation calls are made, and a continuation shorter than 200 characters stops the loop.

This implements the corresponding step of Section 2.4.

System message.

```
You write realistic, internally-consistent source material for evaluation problems. Output ONLY the
     material text, no preamble, no meta commentary.
```

User message.

```
Continue the SAME material with additional new sections/records/data (no repetition, stay consistent).
     Aim to add ~{CONTINUATION_CHARACTERS} more characters. Output ONLY the new continuation.

--- material so far (tail) ---
{MATERIAL_TAIL}
```

Substitutions. CONTINUATION_CHARACTERS is half the target length, rounded down. MATERIAL_TAIL contains up to the last 2,000 characters. The system message is the same as for material generation.

C.3 TASK DESIGN

This prompt turns the prepared material and sampled capabilities into a task plan. It specifies one coherent deliverable, the persona and request, response constraints, the role of each capability, traps and multi-turn progression. Both material sources use this same design step.

This implements the corresponding step of Section 2.4. =

System message.

```
You are a senior benchmark author. You design realistic, hard instruction-following tasks. You think
      carefully about the deliverable, how to pose it, how it forces reasoning, and what traps to embed.
       Output ONE JSON object only.
```

User message.

```
Design ONE realistic instruction-following task grounded in the SOURCE MATERIAL below.

HARDEST REQUIREMENT (from analysis): real tasks are NOT extraction. The correct answer must be REASONED
       OUT - the model must LOCATE key info scattered in the long material, then REASON / COMPUTE /
      APPLY RULES / RECONCILE CONFLICTS to DERIVE the result. Never make the deliverable a recall of
      stated facts. A single task fuses many fine abilities woven together into ONE coherent deliverable
      .

HARD RULES:
- NOT a reading quiz. NOT "answer the following N questions". The user makes ONE natural request; the
      assistant must produce ONE coherent DELIVERABLE.
- The deliverable must require: (1) LOCATING the key information (often scattered / conflicting) in the
       material, (2) REASONING / COMPUTING / APPLYING RULES / RECONCILING on it, (3) obeying multiple
      FORMAT / PERSONA / RESPONSE constraints. Not mere extraction.

Domain: {CONTEXT_CATEGORY} / {SUB_CATEGORY}
FORM (shape the scenario like this): {FORM} - {FORM_GUIDE}
Task type: {TASK_TYPE} - {TASK_TYPE_DESCRIPTION} ({TASK_TYPE_HINT})
Primary skill stressed: {PRIMARY_GROUP} ({PRIMARY_GROUP_NAME})
FUSION SIGNATURE (weave the abilities in this coherent style): {FUSION_GUIDE}
{DESIGN_TURN_NOTE}

TRAPS TO EMBED (dig these into the material/ask; the answer must handle them via reasoning):
{TRAP_GUIDE}

Fine abilities to WEAVE INTO the single deliverable (they are properties of the one answer, not
      separate questions):
{CAPABILITIES}

SOURCE MATERIAL (use its REAL names/numbers/rules):
<<<MATERIAL
{MATERIAL}
MATERIAL>>>

Output JSON plan:
{
 "scenario": "the situation; who the assistant is (persona); who is asking and why",
 "deliverable": "the ONE thing to produce; and exactly WHY it needs reasoning over the material (which
      key info must be located, what must be computed/applied/derived)",
 "response_constraints": ["6-10 concrete constraints the response must obey; MUST INCLUDE at least: 1
      length/word-count constraint, 1 tone/register constraint, 1 format/structure constraint. ONLY add
      a prohibition/negative ('must NOT ...') constraint IF a NEG-family checkpoint appears in the woven
       abilities below; otherwise do NOT invent prohibitions. Distribute constraints between the system
      prompt (persistent) AND the user turn (per-request). Constraints should feel natural for the
      scenario, not bolted on."],
 "the_ask": "the user's ONE natural message, phrased as: {ASK_STYLE}. Natural wording, NOT a numbered
      question list.",
 "woven_checkpoints": [{"leaf":"CCx","how":"how this ability is REQUIRED by the deliverable (a
      reasoning step / derived value / format rule / grounding need)"}],
 "traps": [{"type":"conditional|pressure|conflict|false_premise|out_of_scope","where":"where it sits","
      correct_action":"what the model must do"}],
 "turns_plan": "for multi-turn: what each turn adds"
}
All fields in ENGLISH only.
```

Substitutions. The constant reasoning mandate is expanded verbatim. FORM_GUIDE, FUSION_GUIDE and TRAP_GUIDE are the selected entries from guidance.py. CAPABILITIES lists each selected item as "- [leaf] name (family)". MATERIAL contains at most 48,000 characters. DESIGN_TURN_NOTE selects one of the alternatives below.

DESIGN_TURN_NOTE / multiple user turns.

```
This task is MULTI-TURN ({USER_TURNS} user turns): plan how the deliverable EVOLVES across turns (a
     later turn adds a twist / changes a constraint / depends on the earlier answer).
```

DESIGN_TURN_NOTE / one user turn.

```
Single user turn.
```

TRAP_GUIDE / when no trap is selected.

```
  - (embed at least one subtle reasoning trap natural to the scenario)
```

C.4 JOINT PROBLEM AND RUBRIC WRITING

This prompt realizes the design plan as a system prefix, dialogue, self-contained rubrics and a verify field in one model call. It also selects where the material is inserted. The generator does not reproduce the material: assembly code inserts the prepared material into the system prompt or the first user message. Supplied numerical checks feed the verification stage.

This implements the corresponding step of Section 2.4; its outputs are checked as described in Section 2.5.

System message.

```
You write the final instruction-following problem from a design plan. Output ONE JSON object only. Do
     NOT repeat the material verbatim in your output.
```

User message.

```
Write the final problem implementing this DESIGN PLAN exactly.

DESIGN PLAN:
{DESIGN_JSON}

SOURCE MATERIAL (refer to it; do NOT paste it in your JSON):
<<<MATERIAL
{MATERIAL}
MATERIAL>>>

{WRITING_TURN_NOTE}{MULTIDOCUMENT_NOTE}

Write JSON:
{
 "system_prefix": "<persona + operating rules + ALL response_constraints from the plan; rich and
     specific; SHORT vs material; do NOT include the material>",
 "material_placement": "system" | "user_first",
 "messages": [{"role":"user","content":"<the user's ONE natural request from the_ask; NOT a question
     list>"}, {"role":"assistant","content":"..."}, ...],
 "rubrics": ["..."],
 "verify": [...]
}

RUBRIC RULES (critical):
- SELF-CONTAINED: a JUDGE reading ONLY the rubric + the response (NOT the source material) must be able
      to determine pass/fail. Each rubric MUST name the specific expected value/answer/structure/
     behavior. Do NOT write rubrics that say "according to the material" or "from the source" - the
     judge cannot see the material. Write "The response should state that X is Y" (concrete) not "
     should correctly extract the value" (vague).
- Each rubric checks a PROPERTY of the correct deliverable: the derived conclusion/decision/number is
     correct (and reasoned), a required reasoning/compute/rule-application step is done, a format/
     section/length/tone/persona constraint is met, a grounding/citation is correct, a trap is handled
     correctly, a prohibition is respected.
- MIX rubric types. Produce >= {TARGET_RUBRICS} rubrics.
- DO NOT write "the response should state that <fact>" style quiz rubrics UNLESS that fact is a load-
     bearing input the deliverable must surface while reasoning. Avoid turning this into fact-recall.
- CL-bench phrasing: "The response should ..." / "Does the agent ...? Fail if ...". No conflicting
     constraints.
- ALL content (system, user turns, rubrics) MUST be in ENGLISH ONLY. No Chinese characters anywhere.
```

Substitutions. DESIGN_JSON is the first 6,000 characters of the serialized plan; MATERIAL is the first 48,000 characters of the source. WRITING_TURN_NOTE selects an alternative below. MULTIDOCUMENT_NOTE is inserted only when DOCUMENT_COUNT exceeds one; otherwise it is empty.

WRITING_TURN_NOTE / multiple user turns.

```
MULTI-TURN: {USER_TURNS} user turns. messages alternate user/assistant ending on user; earlier
     assistant turns are short plausible prior replies; later user turns evolve the deliverable per the
      plan's turns_plan.
```

WRITING_TURN_NOTE / one user turn.

```
Single user turn.
```

MULTIDOCUMENT_NOTE / more than one document.

```
The material is {DOCUMENT_COUNT} separate documents (joined by '====='); the framing must acknowledge
     multiple documents.
```

Implementation detail. The implementation defines an additional numnote requesting a program for every derived-value rubric, but does not interpolate it into the writing message. It is therefore not part of the transmitted prompt reproduced here. The verify field alone does not guarantee verification coverage for every numerical rubric.

D CL-BENCH TRAINING DATA STATISTICS

We analyze the CL-bench training pool constructed by CapSynth using GLM-5.2. The pool contains 20,000 tasks used for reinforcement learning.

Construction and filtering. The generation batches contributing to this pool contain 22,700 candidate records, of which 20,087 pass generation-stage acceptance checks. We retain 20,000 tasks for the final training pool; the remaining 87 accepted tasks are surplus rather than additional validation failures. All retained prompts can be matched exactly to their generation records. During reinforcement learning, rollout groups are further filtered according to within-group reward variation, so the number retained at this stage depends on the current policy and training step.

Statistic	Value
Generation model	GLM-5.2
Candidate records	22,700
Accepted records	20,087
Final training tasks	20,000
Retrieved-material tasks	6,497 (32.5%)
Synthesized-material tasks	13,503 (67.5%)
Mean prompt length (characters)	47,025
Median prompt length (characters)	35,822
Mean rubrics per task	18.72
Median rubrics per task	17
Tasks with multiple user turns	7,039 (35.2%)

Table 8: Statistics of the GLM-5.2-synthesized CL-bench training pool. Prompt length counts all system and dialogue text preceding the final answer, excluding the answer and scoring rubrics. Source proportions describe the reference material; tasks and rubrics are generated by GLM-5.2 in both routes.

Reference material sources. Of the retained tasks, 6,497 use material retrieved from a local corpus, while 13,503 use material synthesized by GLM-5.2 from the task specification. Retrieved sources include technical standards and regulations from IETF, NIST, and OSHA; financial documents such as SEC filings and IRS instructions; scientific and health material from PubMed Central, CDC, and WHO; humanities texts from Project Gutenberg, Wikisource, and the Stanford Encyclopedia of Philosophy; and game rules, simulation documentation, and instructional resources such as iFixit.

The source corpus also contains preprocessed excerpts and perturbed or fictionalized variants. Thus, "retrieved material" denotes the corpus-backed generation route rather than a guarantee that every selected text is an unmodified real-world document. In both routes, GLM-5.2 generates the task and its scoring rubrics against the prepared material.