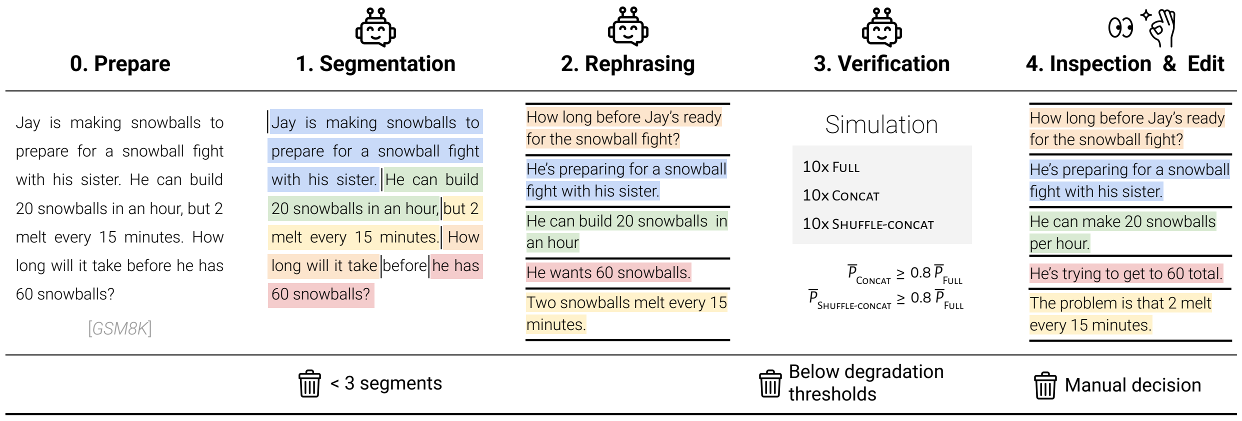
0. Prepare
1. Segmentation
2. Rephrasing
3. Verification
4. Inspection & Edit
Jay is making snowballs to prepare for a snowball fight with his sister. He can build 20 snowballs in an hour, but 2 melt every 15 minutes. How long will it take before he has 60 snowballs?
[GSM8K]
How long before Jay's ready for the snowball fight?
He's preparing for a snowball fight with his sister.
He can build 20 snowballs in an hour
He wants 60 snowballs.
Two snowballs melt every 15 minutes.
Simulation
10x FULL
10x CONCAT
10x SHUFFLE-CONCAT
$\overline{P}_{\text{CONCAT}} \geq 0.8\,\overline{P}_{\text{FULL}}$
$\overline{P}_{\text{SHUFFLE-CONCAT}} \geq 0.8\,\overline{P}_{\text{FULL}}$
How long before Jay's ready for the snowball fight?
He's preparing for a snowball fight with his sister.
He can make 20 snowballs per hour.
He's trying to get to 60 total.
The problem is that 2 melt every 15 minutes.
< 3 segments
Below degradation thresholds
Manual decision

Figure 7: Process diagram of the four-step semi-automatic process to transform fully-specified instructions into a sharded instruction. The first three steps (segmentation, rephrasing, verification) are automated, while the fourth (inspect and edit) was manually completed by the authors of the work. The last row represents the rejection criteria for a sample.

P3: Order Insensitive. Apart from the first shard, the other shards should be decontextualized [13] and not refer to each other in a way that implies an order. As a result, the shard set presented in any order reveals equivalent information. Let $\rho(\mathbf{s}_{2..k})$ refer to a permutation of the shard ordering, then $I(q) = I(\tilde{q})\ \forall \tilde{q} = [s_1, \rho(\mathbf{s}_{2..k})]$

P4: Maximal Sharding. The sharding process should strive to maximize the number of shards extracted from the original instruction (maximize k). This can be achieved by producing shards that introduce a single, specific piece of information.

P5: Minimal Transformation. The sharded instruction should maintain the instruction language and avoid simplifying, altering, or interpreting elements of the original instruction as much as possible. Apart from modifications to satisfy properties P1-P4, the sharding process should attempt to limit modifications such that the shards ($[s_1, \cdots s_k]$ are semantically similar to the atomic content units $I(q)$.

Appendix C Semi-Automatic Sharding Process

We rely on a semi-automatic process to transform fully-specified instructions into their sharded equivalents. The process – illustrated in Figure 7 – consists of a sequence of three automated steps (Segmentation, Rephrasing, Verification) followed by a manual step that was conducted by an author of the paper.

We now detail each step of the process, then go over task-specific details we implemented as needed. We note that as part of our open-source release, we provide all the prompts used in the first three LLM-based steps.

Step 1: Segmentation Given an original fully-specified instruction (left-most column in Figure 7), the LLM is prompted to extract *segments* of the instructions. Segments are intended to correspond to the atomic content units (defined in Appendix B). In particular, the prompt indicates that segments must not overlap, and that not all words in the original instruction must belong to a segment. Prompts are task-specific and incorporate at least three few-shot examples of segmentation, to allow for the concept of segmentation to be illustrated through examples. At this stage, any instruction that yields fewer than three segments are filtered out and does not proceed to the next stage.

Step 2: Rephrasing Given the original fully-specified instruction and the extracted segments, this stage consists in rewriting each segment to be decontextualized [13] and conversational. In other words, dependencies between segments are resolved, and the ordering is changed such that obtained *shards* adhere to properties P2 and P5. In the example above, the fourth segment (highlighted in orange) becomes the first shard as it reveals the overall intent, and light rephrasing occur in other shards. The rephrasing prompt is task-specific, and includes few-shot examples of rephrasing segmented instructions.

Step 3: Verification Steps 1-2 produce a sharded instruction that can be used to simulate SHARDED and CONCAT conversations. To verify the property P1 (Information Preservation) that no information has been lost during segmentation and rephrasing, we conduct preliminary simulations to evaluate the original and sharded instruction side-by-side. Specifically for each pair of the original and the sharded instruction, we simulate ten FULL conversations with the original instruction, ten CONCAT conversations with the sharded instruction (by concatenating the shards), and ten

SHUFFLE-CONCAT conversations. SHUFFLE-CONCAT is a variant of the CONCAT simulation in which all shards (except Shard 1) are randomly permuted before being concatenated. This variant can be seen as a more adversarial version of CONCAT, verifying the property P3 (Order Insensitive). For each simulation type, we calculate the averaged performance $\overline{P}$ over ten runs and filter out instructions that are below an acceptable degradation threshold. Specifically, instructions are acceptable if the following conditions are met:

$$\overline{P}_{\text{CONCAT}} \geq 0.8\,\overline{P}_{\text{FULL}}$$
$$\overline{P}_{\text{SHUFFLE-CONCAT}} \geq 0.8\,\overline{P}_{\text{FULL}},$$

where $\overline{P}_{\text{X}}$ denotes the averaged performance of the simulation type X. If more degradation is observed (*i.e.*, below 80%), it indicates a potential loss of information during sharding, or that decontextualization was not implemented accurately.

Step 4: Inspect and Edit Even though the first three steps define the sharding process and implement some level of quality assurance, they do not guarantee the level of quality required for precise and large-scale experiments due to relying on LLM outputs. To obtain high-quality shards, we reserve step 4 for manual inspection and validation. To facilitate the procedure, we developed a web-based annotation interface. In the interface, an annotator can review a pair of fully-specified and sharded instructions, edit, add, or remove individual shards, and decide to accept or reject sharded instructions. Sharded instructions included in our experiments were all manually reviewed by two authors of the work. The amount of editing and filtering required in this final stage varied by task.

Inspecting and editing an auto-generated instruction typically requires 1-3 minutes per instruction, an order of magnitude less than it would require for authors to write the sharded instructions de-novo from a given fully-specified instruction. As part of our open-source release, we provide all the prompts used during sharding, which we hope can facilitate the sharding of additional tasks.

Appendix D Inspection of Simulated Sharded Conversation

Inspection	All Tasks	Actions	Code	Math	Db
Shard Fully Revealed	96.0	98.3	94.9	93.4	100.0
Shard Contextualized	98.4	98.3	98.3	98.3	98.6
Strategy Accuracy	95.2	94.7	95.5	95.6	94.7
Extraction Success	97.0	100.0	93.4	98.4	100.0
Overall Success	97.8	100.0	96.0	96.0	100.0

Table 5: Results of the inspection of 100 simulated sharded conversations across four tasks: Actions, Code, Math, and Database. The first column aggregates annotation results on the four tasks.

The sharding simulation environment (described in Section 3) relies on LLM components to simulate the user, classify assistant responses, and extract answers from free-text responses. LLM-based components are likely to fail, and we performed an inspection of 200 simulated SHARDED conversations to understand the level of simulation error and the potential effect on estimating the performance of the assistant LLMs due to the error.

For each inspected conversation, we annotated user turns, assistant turns, and the overall conversation with five specific elements.

For user utterances, we annotated whether the utterance revealed exactly the information from one shard in the sharded instruction (`Shard Fully Revealed`). Specifically, we flagged turns that revealed more than one shard, and turns that revealed a shard only partially. We also annotated each user's turn for whether it is appropriately contextualized in the conversation (`Shard Contextualized`). For example, if the previous assistant's turn asked a binary clarification question (yes/no), then proper contextualization would require a Yes/No response to directly address the assistant's response.

For assistant utterances, we annotated whether the classified strategy was accurate (`Strategy Accuracy`). For example, if the response is labeled as a clarification, we confirm if it poses a clarification question to the user. When assistant utterances were labeled as answer attempts, we further labeled whether the answer extraction step was successful (`Extraction Success`).

Upon completing the inspection of each user and assistance utterance, we assigned a global label to the entire conversation on whether or not the errors that occurred during simulation (if any) affected the overall validity of the simulation. If not, the simulation was marked as successful (`Overall Success`).

We inspected conversations for four tasks: Actions, Code, Math and Database. The other two (Summary and Data-to-text) are refining tasks that require an answer attempt at each turn, and do not rely on an LLM-based user simulator. As such, they have limited scope for simulation error.

Table 5 summarizes the results of the inspection annotation. Overall, the simulation environment is highly reliable, with roughly 98% of inspected conversations labeled as successful. Some errors occur in each component. With user simulation, a single shard is fully revealed around 96% of the time, and properly contextualized 98% of the time. The processing of assistant responses also leads to errors: the turn strategy classification is only 95% accurate, and extraction of answer attempts has an accuracy of 97%.

Utterance-level errors did not always affect the validity of the overall simulation. In some cases, we observed that the user simulator would correct an error in an early turn, subsequently in the conversation, or that an error in answer extraction on the wrong answer attempt would occur at a turn, but the extraction would be successful later on. In summary, we empirically find that the simulation environment is largely accurate: though some errors occur, large drops of performance in the SHARDED setting (beyond 2%) are not due to errors caused by the simulator.

Appendix E Concrete Example of Loss in Aptitude vs. Reliability

Let's imagine we are provided with ten instructions ($N = 10$), each FULL and SHARDED. We run simulations with an LLM, simulating 10 conversations per instruction and setting ($M = 10$). Let's assume the LLM achieves an averaged performance ($\overline{P}$) of 90% in the FULL, and 60% in the SHARDED setting.

Finally, let's assume that the FULL performance is achieved by having perfect performance (*i.e.*, success in 10/10 randomly sampled runs) on 9 instructions, and failing on all the sampled simulations of the last, tenth instruction. In other words:

$$S_{ij}^{\text{FULL}} = \begin{cases} 100, & \text{if } i \in \{1, \ldots, 9\} \\ 0, & \text{if } i = 10 \end{cases},$$

where S_{ij}^{FULL} represents the score for i-th instruction at j-th simulation run. The aptitude (A) and unreliability (U) of the LLM for the FULL setting is $A = 90\%$ and $U = 0\%$ (*i.e.*, for each instruction, the 10th and 90th percentile scores are equal).

Let's now consider three conditions for the SHARDED setting that all achieve an averaged performance of $\overline{P} = 60\%$. We illustrate the conditions in Figure 8.

Situation 1: Drop in Aptitude. The LLM achieves perfect performance on six of the ten instructions:

$$S_{ij}^{\text{SHARDED}} = \begin{cases} 100, & \text{if } i \in \{1, \ldots, 6\} \\ 0, & \text{if } i \in \{7, \ldots, 10\} \end{cases}.$$

In situation 1, $\overline{P} = 60\%$, $A = 60\%$, and $U = 0\%$. The degradation in performance is entirely explained by a decrease in aptitude, while the reliability remains the same.

Situation 2: Drop in Reliability. The LLM achieves mixed performance (6-7 perfect scores per instruction) on nine of the 10 instructions:

$$S_{ij}^{\text{SHARDED}} = \begin{cases} 100, & \text{if } 1 \le i \le 3, 1 \le j \le 6 \\ 100, & \text{if } 4 \le i \le 9, 1 \le j \le 7 \\ 0, & \text{otherwise} \end{cases}.$$

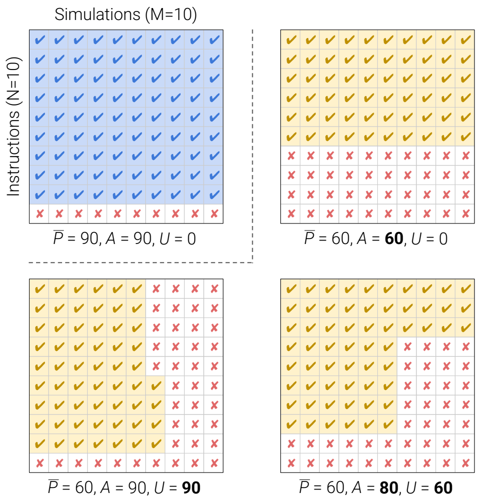
Simulations (M=10)
Instructions (N=10)
$\overline{P}$ = 90, A = 90, U = 0
$\overline{P}$ = 60, A = **60**, U = 0
$\overline{P}$ = 60, A = 90, U = **90**
$\overline{P}$ = 60, A = **80**, U = **60**

Figure 8: Illustrations for different situations. Green and red fills in each grid indicate sample-level score (*e.g.,* pass / exact match). Compared to FULL (top left), three situations in SHARDED achieve the same $\overline{P} = 60$ while varying in aptitude A and unreliability U.

In situation 2, $\overline{P} = 60\%$, with an aptitude of $A = 90\%$, and a unreliability of $U = 90\%$. The degradation in performance is entirely explained by a large drop in reliability, while sharded and fully-specified aptitude are equal.

Situations 1 and 2 illustrate extreme scenarios where the average drop in performance is entirely explained by a drop in aptitude or reliability, but in practice a combination is more likely to occur, as in situation 3.

Situation 3: Combined drop in Aptitude and Reliability. The LLM achieves perfect performance on three instructions, and mixed performance (6 perfect scores per instruction) on five of the 10 instructions: