Fully-Specified Instruction (original)

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?

(a) Original GSM8K instruction.

Sharded Instruction (based on original)

Shard 1: How long before Jay's ready for the snowball fight?

Shard 2: He's preparing for a snowball fight with his sister.

Shard 3: He can make 20 snowballs per hour.

Shard 4: He's trying to get to 60 total.

Shard 5: The problem is that 2 melt every 15 minutes.

(b) Equivalent Sharded Instruction.

Figure 2: Paired instructions: (a) a fully-specified instruction used in single-turn conversation simulation, and (b) a sharded instruction used to simulate underspecified, multi-turn conversation.

As part of our work, we developed a semi-automatic sharding process to scale the creation of sharded instructions. This process, described in depth in Appendix C, ensured that the experiments we carried out used sharded instructions that adhered to the properties we defined.

3.2 Simulating Sharded Conversations

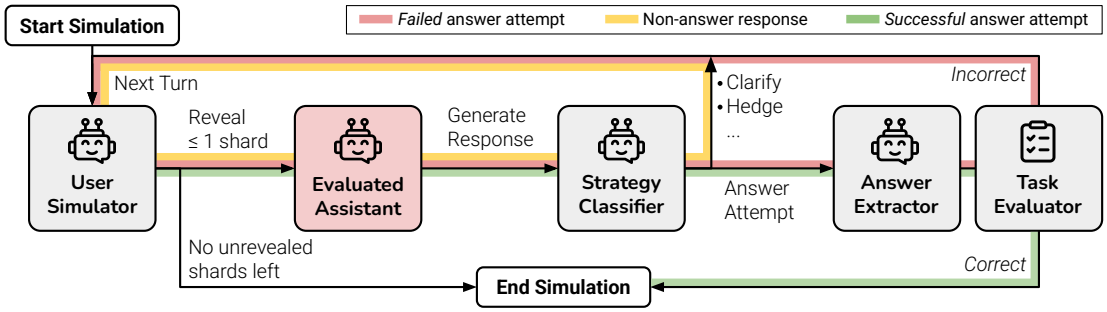

Figure 3: Sharded Conversation Simulation Diagram. The subject for the simulation is highlighted in red.

Figure 3 depicts the process of simulating a multi-turn, underspecified conversation based on a sharded instruction. At a high-level, the conversation involves three parties: the **assistant** is the LLM being evaluated in the simulation, the **user** (simulated by an LLM) who has access to the entirety of the sharded instruction and is in charge of revealing shards during turns of the conversation, and the **system** which categorizes and evaluates assistant responses.

On the first turn, the user simulator reveals the first shard of the instruction (*i.e.*, Shard 1) to the assistant, which then generates a free text response. The system processes the assistant's response into one of seven possible response strategies: *clarification*, *refusal*, *hedging*, *interrogation*, *discussion*, *missing*, or *answer attempt*,[2] based on Herlihy et al. [27]'s LLM response categorization. If the assistant generates an answer attempt (*i.e.*, proposing an explicit, full-form solution), then the answer extractor component determines the span that corresponds to the answer within the assistant's free-form response (*e.g.*, code snippet, number). This step is required because LLMs often pad answer attempts with additional information, such as a natural-language explanation or a follow-up question, which could hinder evaluation. Finally, the extracted answer is scored by a task-specific evaluator function. Subsequent turns follow a similar pattern: at each turn, the user simulator reveals at most one shard of information, the assistant responds freely, which gets evaluated if the response is classified as an answer attempt. The conversation ends if one of two conditions is met: (1) the task-evaluator assesses that an assistant answer attempt is correct, or (2) if at the start of a new turn, the user simulator has run out of shards to reveal in the conversation.

Preliminary experiments revealed that during simulation, evaluated assistants often asked clarification questions that related to specific shards of the instruction. As such, deciding which shard to reveal next in the conversation (the role of the user simulator) is non-trivial, as it should take into account the state of the conversation so far. We instantiate the user simulator as a low-cost LLM (specifically, GPT-4o-mini) that has access to the entire sharded instruction and the state of the conversation so far, tasking it with deciding the next shard to reveal that fits most naturally in the ongoing simulated

[2]See Appendix G for the definition and the example for each strategy.

conversation. The user simulator is also tasked with rephrasing the shard to fit naturally within the conversation without modifying its informational content. See Appendix J for an example simulated sharded conversation.

Besides user messages, the assistant receives a minimal system instruction (before the first turn) that provides the necessary context to accomplish the task (such as a database schema or a list of available API tools). Importantly, the assistant is not explicitly informed that it is participating in a multi-turn, underspecified conversation and is not encouraged to pursue specific conversational strategies. Although such additional instructions would likely alter model behavior, we argue that such changes are not realistic, as such information is not available a priori in practical settings. In summary, we provide no information about the setting to the evaluated assistant model during simulation, aiming to assess default model behavior.

Apart from the user simulator, the strategy classifier and answer extractor components are also implemented with prompt-based GPT-4o-mini. While the choice of LLM-based components in the simulator allows for dynamic choices that provide a more realistic simulation, they also unavoidably lead to simulation errors, which can affect the validity of experiments. To understand the scope of simulation errors and their effect on simulation validity, we conducted an in-depth manual annotation of several hundred simulatesouthworth2023developingd conversations. The annotation effort and its findings are detailed in Appendix D. In summary, we found that errors introduced by the user simulator, strategy classifier, or answer extraction occurred in less than 5% of inspected conversations and that these errors disfavored the assistant model in less than 2% of the conversations. We believe the process described above can accurately simulate multi-turn, underspecified conversations based on sharded instructions, and we rely on it to simulate conversations for our experiments.

3.3 Simulation Types

We leverage sharded instructions to simulate five types of single- or multi-turn conversations, as illustrated in Figure 4. We now introduce each one and explain its purpose in our experiments.

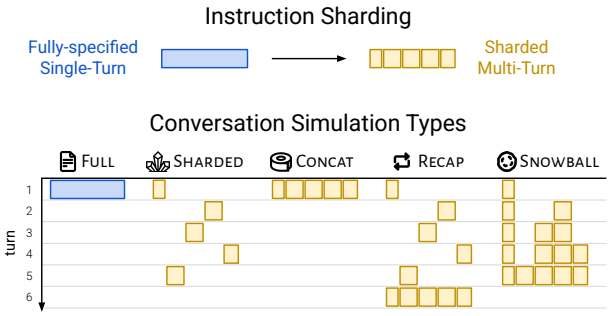

Figure 4: Conversation simulation types based on sharded instructions. Once an original fully-specified instruction (blue block) is sharded (set of yellow blocks), the "shards" can be used to simulate single-turn (FULL, CONCAT) or multi-turn (SHARDED, RECAP, SNOWBALL) conversations, affecting the pace of information disclosure.

FULLY-SPECIFIED (short-form: FULL) simulates single-turn, fully-specified conversations in which the original instruction is provided to the LLM in the first turn. This simulation type evaluates baseline model performance on the tasks.

SHARDED simulates multi-turn, underspecified conversations as outlined above. SHARDED simulations are our primary tool to evaluate model performance in underspecified, multi-turn conversations.

CONCAT simulates single-turn, fully-specified conversation based on the sharded instruction. The shards are concatenated into a single instruction in bullet-point form (with one shard per line), preceded by an instruction to complete the task taking into account all bullet-points. The CONCAT simulation is a logical mid-point between full and sharded, in which underspecification is removed (like FULL) but the rephrasing that occurred during instruction sharding is preserved (like SHARDED). CONCAT is intended as a verification baseline: a model that succeeds at both FULL and CONCAT, but not at SHARDED, struggles specifically because of underspecification and the multi-turn nature of the conversation, and not due to the rephrasing that occurred during the sharding process, which may have led to information loss.

RECAP simulates a SHARDED conversation, and adds a final *recapitulation turn* which restates all the shards of the instruction in a single turn, giving the LLM one final attempt at responding. RECAP is a combination of the SHARDED simulation followed by a CONCAT turn, and is explored as a method in Section 7.1 to evaluate whether such a conceptually simple agent-like intervention can mitigate the loss in performance observed in SHARDED conversations.

SNOWBALL takes the RECAP simulation a step further, implementing turn-level recapitulation. At each turn, the user simulator introduces a new shard, but also restates all the shards that have been revealed so far in the conversation, producing a *snowball* effect as each turn reveals all the information from the previous turn, plus one additional shard. The redundancy implemented in the SNOWBALL simulation is also explored as a method in Section 7.1 to study whether turn-level reminders help alleviate the need for LLMs to recall information across multiple turns of context.

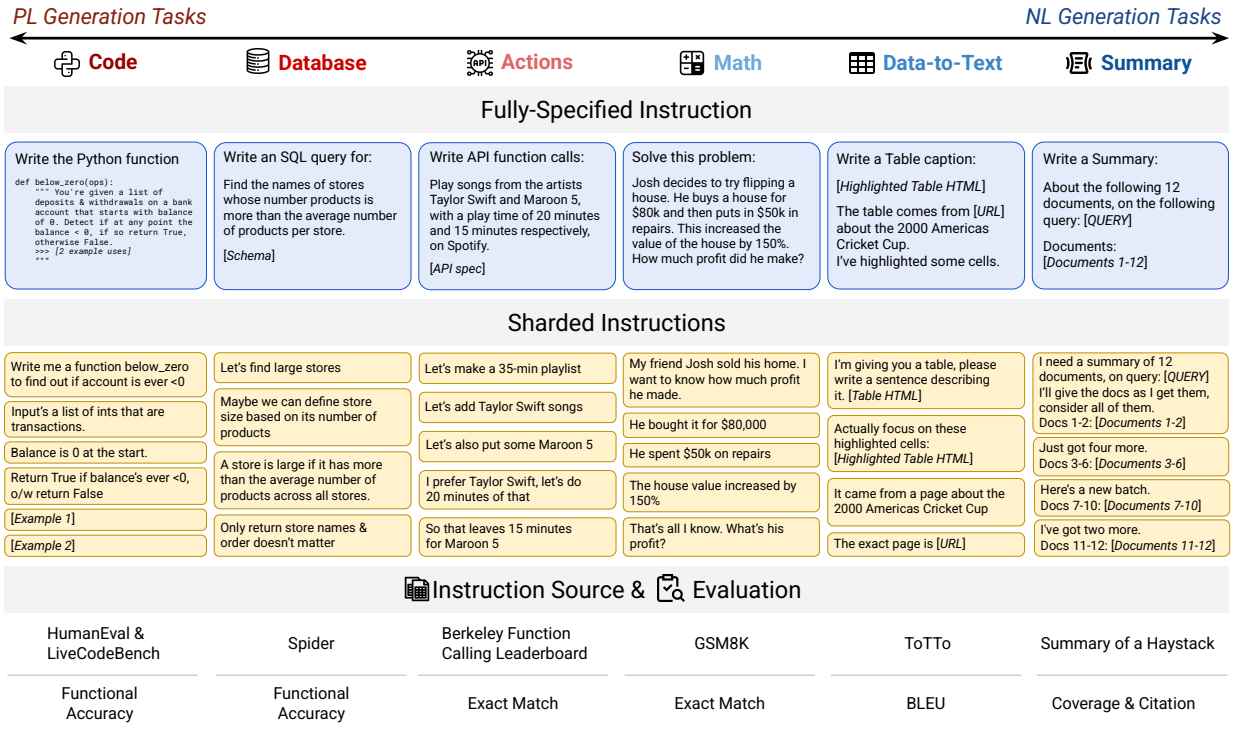

Figure 5: Six sharded tasks included in our experiments. We purposefully include tasks that involve generating programming and natural language. For each task, an illustrative fully-specified instruction and its sharded counterpart. We sharded 90-120 instructions based on high-quality datasets (Instruction Origin), re-purposing existing evaluation.

4 Task and Metric Selection

4.1 Task Selection

We constructed sharded instructions for six tasks that we use in a large-scale simulation experiment. For each task, we selected instructions from one or two high-quality single-turn, fully-specified benchmarks, and implemented a semi-automatic sharding process. The process relied first on an LLM (GPT-4o) to propose and verify sharding candidates, which were then reviewed and edited (when necessary) by the authors of the work. The sharding process (outlined in detail in Appendix C) allowed us to scale the construction of sharded instruction corpora while ensuring validity of the underlying instructions. For each task, we prepared 90-120 sharded instructions (each paired with the original single-turn instructions), which required between 1-4 hours of manual inspection and annotation.

We carefully selected popular and diverse generation tasks across programming and non-programming use cases. Figure 5 provides an example of an original and sharded instruction for each task, which we now introduce.

Code The assistant must help the user write a function in the Python programming language. The original instructions were sourced from the HumanEval [10] and LiveCodeBench [31] datasets, two popular benchmarks used to evaluate LLM programming aptitude.

Database The assistant is provided with the schema of an SQL database and a user query in natural language, and must produce an SQL query that retrieves the requested information from the database (a.k.a., text-to-SQL). The original instructions and databases were sourced from the popular Spider dataset [86].

Actions The assistant is provided with a set of API (Application Programming Interface) schemas, and a user instruction that requires API use, and must generate the programmatic API commands that match the user request. We sourced API schemas and user instructions from the Berkeley Function Calling Leaderboard (BFCL) [85], a popular benchmark used to measure LLM ability at API function calling.

Math The assistant is provided with an elementary math word problem, and must perform a series of calculations using basic arithmetic operations to reach a numerical answer. We sourced problems from the GSM8K dataset [14].