LLM'S GET LOST IN MULTI-TURN CONVERSATION

My summary of paper from: Laban, Philippe, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. *arXiv preprint arXiv:2505.06120* (2025).

INTRODUCTION

CORE QUESTION

Do LLMs perform worse when:

- Instructions are **revealed gradually**
    - Like in continuous conversation
- Requirements are **clarified over multiple turns**
- The task starts **underspecified**

EXAMPLE

Single-Turn (Benchmark Style): Model sees all constraints at once

- User: Write a Python function below_zero (ops) that returns True if cumulative balance ever drops below zero. Balance starts at 0. Example inputs included.

Multi-Turn (Realistic Style): Now the model must: (a) Integrate previous context (b) Avoid premature assumptions (c) Update its reasoning

- User: I need help writing a function.
- LLM: Sure, what should it do?
- User: It processes deposits and withdrawals.
- LLM: Okay.
- User: The balance starts at zero.
- LLM: Got it.
- User: I need to know if it ever drops below zero.

So the question is to investigate whether performance degrades in this second scenario.

INVESTIGATION STEPS

STEP 1: SHARDING INSTRUCTIONS

Take a fully-specified task and break it into **shards** (small pieces).

EXAMPLE

Original (Math Problem):

- Jay makes 20 snowballs per hour. 2 melt every 15 minutes. He wants 60 snowballs. How long will it take?

Sharded:

- Turn 1: How long until Jay is ready?
- Turn 2: He's preparing for a snowball fight.
- Turn 3: He can make 20 per hour.
- Turn 4: He wants 60 total.
- Turn 5: 2 melt every 15 minutes.

STEP 2: SIMULATED CONVERSATION

At each turn:

- User reveals at most 1 shard
- Model responds
- If model attempts final answer → evaluated

Repeat each task **10 times** to measure variability.

STEP 3: MEASURE 3 THINGS

Instead of just accuracy, they measure:

- **Average Performance:** How often does it succeed overall?
- **Aptitude (Best Case):** In the best 10% of runs, how well does it perform?
- **Unreliability:** Difference between best and worst runs.

EXAMPLE

Suppose 10 runs produce scores: [100, 100, 100, 80, 60, 40, 20, 0, 0, 0]

- Aptitude ≈ 100
- Worst case ≈ 0
- Unreliability = 100

Meaning: The model *can* solve it — but often fails badly. That's instability.

OTHER DETAILS

DATASET

Used 600 total instructions across 6 different task (Code, SQL (Text-to-SQL), API calling, Math, Data-to-text, Multi-document summarization)

Each instruction was divided into two forms:

- Fully-specified (original benchmark form)
- Sharded (multi-turn form)

MODELS TESTED

Evaluated **15 LLMs** from 8 families:

- OpenAI (GPT-4o-mini, GPT-4o, o3, GPT-4.1)
- Anthropic (Claude 3 Haiku, Claude 3.7 Sonnet)
- Google (Gemini 2.5 Flash, Gemini 2.5 Pro)
- Meta (Llama 3.1 8B, Llama 3.3 70B, Llama 4 Scout)
- AI2 OLMo-2-13B
- Microsoft Phi-4
- DeepSeek-R1
- Cohere Command-A

Variety of models tested, like:

- Small models (~8B)
- Very large models (300B+)
- Open-weight and closed-weight
- Two reasoning models (o3, DeepSeek-R1)

Purpose was to also test whether stronger models or reasoning models handle multi-turn better.

REPEATED SIMULATIONS (VARIABILITY)

For each: (Model × Instruction × Simulation Type), ran 10 independent simulations for each and computed (a) average performance (b) Aptitude (A90 — best-case) (c) Unreliability (U90–10 — variability gap)

DIFFERENT TASKS

TYPE – 1: Code (Python Function Writing)

- Source: HumanEval, LiveCodeBench
- Example:
    - Original (Fully-Specified): Write a Python function below_zero(ops) that returns True if balance ever drops below zero. Balance starts at 0. Include examples.
    - Sharded:
        - I need help writing a function.
        - It processes deposits and withdrawals.
        - Balance starts at 0.
        - Return True if it ever drops below zero.
- This tests structured code generation under gradual clarification.

TYPE – 2: Database (Text-to-SQL)

- Source: Spider dataset
- Example:
    - Original: Write SQL to find stores whose number of products is greater than the average number of products per store.
    - Sharded:

        - Find stores.
        - Large stores.
        - Large means above average product count.
        - Only return store names.
- This tests structured query generation.

TYPE – 3: Actions (API Function Calling)

- Source: Berkeley Function Calling Leaderboard (BFCL)
- Example:
    - Original: Play Taylor Swift songs for 20 minutes and Maroon 5 for 15 minutes on Spotify.
    - Sharded:
        - Create a playlist.
        - Include Taylor Swift.
        - Include Maroon 5.
        - 20 minutes of Taylor Swift.
        - 15 minutes of Maroon 5.
- Tests structured tool/API call generation.

TYPE – 4: Math (Word Problems)

- Source: GSM8K
- Example:
    - Original: Josh buys a house for $80k, spends $50k on repairs, value increases by 150%. What profit did he make?
    - Sharded:
        - Josh flipped a house.
        - Bought it for $80k.
        - Spent $50k on repairs.
        - Value increased by 150%.
        - What's his profit?
- Tests arithmetic reasoning under incremental disclosure.

TYPE – 5: Data-to-Text

- Source: ToTTo
- Task: Generate a caption describing highlighted table cells.
- Tests structured natural language generation from tables.

TYPE – 6: Summary (Multi-Document Summarization)

- Source: Summary of a Haystack
- Task: Given ~20 documents and a query, generate a summary with citations.
- Tests long-context reasoning (tens of thousands of tokens).

TOTAL SCALE

Roughly: 600 instructions × 15 models × 3 simulation types × 10 repetitions

≈ 270,000 conversations (they report >200,000)

TEMPERATURE SETTING

All simulations were run at: Temperature = 1 (default stochastic behavior)

Ran a supplementary experiment varying temperature to test whether lowering randomness improves reliability.

SIMULATION TYPES

To isolate why performance drops in conversations, **five controlled simulation types** were designed. Each type changes only how information is delivered — not the total information.

TYPE - 1: FULL — Single-Turn, Fully-Specified

- **What it simulates:** All task requirements are provided in a single prompt.
- **Example:**
    - User:
        - Write SQL to find stores whose number of products is above the average number of products per store. Only return store names. Here is the schema: [...]
    - Everything is specified upfront.
- **Purpose:**
    - Baseline model capability (aptitude under ideal conditions).

TYPE - 2: CONCAT — Single-Turn, Sharded Wording

- **What it simulates:** The instruction is broken into shards (like in multi-turn), but then concatenated back into a single prompt.
- **Example:**
    - User:
        - Complete the task considering:
        - Find stores
        - Large means above average product count
        - Only return store names
        - Use this schema: [...]

    Still single-turn. Just structured differently.

- **Purpose:**
    - Control for rephrasing effects.
    - If performance drops here, it could be due to wording changes during sharding.

TYPE - 3: SHARDED — Multi-Turn, Incremental Disclosure

- **What it simulates:** Information is revealed gradually, one shard per turn.
- **Example:**
    - Turn 1: Find stores.
    - Turn 2: Large stores.
    - Turn 3: Large means above average product count.

- Turn 4: Only return store names.

The assistant must accumulate information across turns.

- **Purpose:**
  - Primary condition to test multi-turn underspecification.

TYPE – 4: RECAP — Multi-Turn + Final Summary

- **What it simulates:** Same as SHARDED, but after all shards are revealed, a final turn restates everything in one message.
- **Example:**
  - After multi-turn clarification: Let me summarize:
    - Find stores
    - Large means above average product count
    - Only return store name
    - Now write the final SQL.
- **Purpose:**
  - Tests whether simply summarizing all constraints at the end restores performance.

TYPE – 5: SNOWBALL — Turn-Level Repetition

- **What it simulates:** At every turn, the user repeats all previously revealed shards plus one new shard.
- **Example:**
  - Turn 1: Find stores.
  - Turn 2: Find stores. Large stores.
  - Turn 3: Find stores. Large stores. Large means above average product count.
  - Turn 4: [All previous info repeated] + Only return store names.
- **Purpose:**
  - Tests whether memory decay (forgetting earlier turns) is the main cause of failure.

Note: TYPE – 5 is used in continuous conversation in our designed system

FINDINGS

" Our findings highlight a gap between how LLMs are used in practice and how the models are being evaluated. Ubiquitous performance degradation over multi-turn interactions is likely a reason for low uptake of AI systems, particularly with novice users who are less skilled at providing complete, detailed instructions from the onset of conversation "

MAIN RESULT

FINDING-1

- **All Models Degrade in Multi-Turn:** Performance in: SHARDED << FULL
  - **Average degradation ≈ 39%**
  - This happened: Across all 15 models and across all 6 tasks

    - Even though: The total information is identical. The task is identical
    - The only difference: Information was revealed gradually

---

FINDING-2

- Performance in CONCAT was ~95% of FULL.
- Meaning: Degradation is NOT due to wording differences. It is due to incremental disclosure.

FINDING-3

- Weaker models showed more degradation even in CONCAT.
- Meaning: They are less robust to paraphrasing. Stronger models are more robust to wording changes.

FINDING-4

- Models like Claude 3.7, Gemini 2.5, GPT-4.1 - all degraded by ~30–40% in SHARDED even though their FULL performance was very high.

FINDING-5

Degradation varies by tasks, like:

- Cohere Command-A performs relatively better on API calls.
- Claude 3.7 and GPT-4.1 maintain Code performance better.
- Gemini 2.5 Pro holds up better in Data-to-Text.

Meaning: Multi-turn robustness is domain-dependent. It is not a uniform property.

FINDING-6

- Reasoning Models Do Not Help
- Hypothesis: More test-time compute → better multi-turn reasoning.
- Result: They degrade similarly.

FINDING-7

- Recap Helps, But Doesn't Fully Fix It
- RECAP improves performance compared to SHARDED.
    - However: It does not restore FULL-level performance.
- Meaning: The problem is not just that models need a final consolidated view

---

FINDING-8

- SNOWBALL (repeating everything each turn) improves performance slightly. But performance still remains below FULL.
- Therefore:
    - Memory Alone Is Not the Main Issue. The failure is not just forgetting earlier turns.
    - It is deeper — related to:
        - Early assumption lock-in

- Premature answer attempts
- Inability to reset reasoning cleanly

DEEP DIVE INTO FINDINGS 7 & 8

The intent for this part of additional experiment was to asssess whether modern LLM systems that use agent frameworks like: LangChain, AutoGen not require multi-turn stability (we can just manage state outside the model) as these systems themselves:

- Break tasks into steps
- Manage memory
- Retrieve documents
- Call tools
- Reconstruct prompts

RECAP: This mimics what an orchestration layer might do:

- Collect all constraints
- Build consolidated prompt
- Send final structured instruction

SNOWBALL: Memory buffer injection**,** Prompt rebuilding each step**,** Explicit state reinforcement **-** Which many frameworks already do.

While both RECAP and SNOWBALL Improve Over SHARDED but even with repetition, performance still significantly below FULL and CONCAT.

Example Intuition: If the model forgot that, say, Large = above average product count, repeating it every turn helps.

Meaning: Simply managing prompt state externally does not eliminate instability.

SNOWBALL Is More Realistic

- It simulates continuous state reinforcement.
- It reduces degradation by ~15–20% (however, original degradation was ~39%, so roughly half the instability remains)

KEY TAKEAWAYS

Single-turn benchmarks measure: "Can the model solve the task?"

Multi-turn measures: "Can the model stay stable while solving the task under evolving information?"

The second is much harder — and currently unsolved.

- **Multi-turn clarification is fundamentally harder**
    - Even 2-turn conversations cause instability.
    - Example: Adding just one clarification causes major performance drop.
    - Furthermore, “Multi-turn is an intrinsic reasoning stability problem.”
        - Because:
            - SNOWBALL repeats all prior shards every turn.
            - So memory is explicitly injected.
            - Yet performance still degrades significantly.
        - If the problem were purely orchestration, SNOWBALL should nearly match FULL. But it doesn’t. So the instability must be deeper.

- **Stronger models are not immune**
    - GPT-4.1 and Gemini degrade similarly to smaller models.
    - Example: High single-turn 95% → Multi-turn ~60–70%.

- **The problem is instability, not intelligence**
    - The model can solve the task It just doesn’t do it consistently.
    - Example:
        - Run 1: Perfect answer
        - Run 2: Totally wrong answer
        - Same prompt history.

- **Early mistakes cascade**
    - One wrong assumption early poisons the rest.
    - Example: Assume wrong schema field → entire SQL chain wrong.

- **Lowering temperature does NOT fix it**
    - Even at T=0: Unreliability remains ~30%.
    - Because early token divergence cascades across turns.

- **Agent-style recap/snowball helps but doesn’t solve it**
    - Repeating all requirements each turn reduces degradation ~15–20%. But still far below single-turn performance.
    - Example: System re-sends full spec before final answer → helps, but not enough.

- **Best practical strategy today:**
    - Provide all requirements upfront.
    - Example: Instead of:
        - Let’s build a query… oh also filter this… oh and also…
        - Better: Here is full specification in one instruction.
        - Reliability increases dramatically.