CODE AS CARRIER: CONTEXT-EFFICIENT MULTI-TURN INSTRUCTION FOLLOWING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models often fail to preserve instructions across multi-turn conversations, especially when users update constraints without restating them. We present Code as Carrier (CoCa), a programmatic framework for conversation management that dynamically detects topic shifts, identifies user intents, and incrementally updates an executable program that maintains and enforces the topic thread's active constraints. This allows CoCa to reapply constraints through execution rather than recovering them from an ever-growing dialogue history. CoCa provides the largest gains when direct prompting struggles to retain prior constraints: on a short, single-topic benchmark, it improves turn-wise retention by 10.6% to 172.0% for weaker models while leaving an already-stable model nearly unchanged. For longer multi-topic dialogues, it can reduce the turn-level failure rate from 92% to 59%. Finally, by updating programs in place, CoCa maintains approximately constant per-turn context and uses $2.9\times$ fewer context tokens over direct baselines.

1 INTRODUCTION

Instruction following is a core capability expected of large language models (LLMs), yet models often struggle to reliably adhere to user-specified constraints, especially in multi-turn conversational contexts (Wen et al., 2024; Zhou et al., 2023). In standard LLM-based inference workflows, instructions and constraints are typically stored implicitly in the growing dialogue transcript. This means that for each turn, the model must repeatedly recover the active constraints by rereading the entire history, a process that becomes increasingly error-prone and computationally expensive as the dialogue lengthens. Silent constraint violations commonly occur when models fail to retrieve earlier directives or misinterpret them. For example, a user might instruct a model to cap responses at 200 words or avoid modifying pre-existing files across a sequence of actions, but these constraints are often dropped after a few turns. The situation is further complicated when instructions evolve or interact: a user's subsequent turns may revise, relax, or withdraw earlier directives, requiring the model to accurately track and reconcile these changes as the conversation progresses (Laban et al., 2026; He et al., 2024). Multi-topic conversations exacerbate these difficulties, as constraints must be applied selectively to the correct topic thread, and switching between threads increases the likelihood of misapplying or forgetting constraints altogether (Jia et al., 2025).

Simply increasing the length of the context window, while helpful for accommodating longer transcripts, does not reliably solve these issues. These models often exhibit attention decay, struggling to prioritize information in the middle of lengthy context inputs, as shown in Liu et al. (2024). Additionally, larger context windows do not address topic interleaving directly, often leading to poor constraint tracking, misapplication, and loss of standing instructions. Like the example in Figure 1, direct prompting LLMs can forget constraints after topic shifts (e.g., Max 12 words), confuse unrelated topics' requirements, prioritize recent inputs over previously stated instructions. These problems compound over time, resulting in significant degradation of model performance in complex, evolving dialogues.

Existing methods either preserve conversational state as text, requiring repeated model processing at each turn, or store it in predefined structures that lack flexibility for open-ended interactions with interleaved topics and evolving constraints, see §5 for a detailed discussion. We argue that addressing these limitations requires a fundamental shift in how constraints are represented and maintained across dialogue turns. To this end, we propose Code as Carrier (CoCa), a programmatic

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.

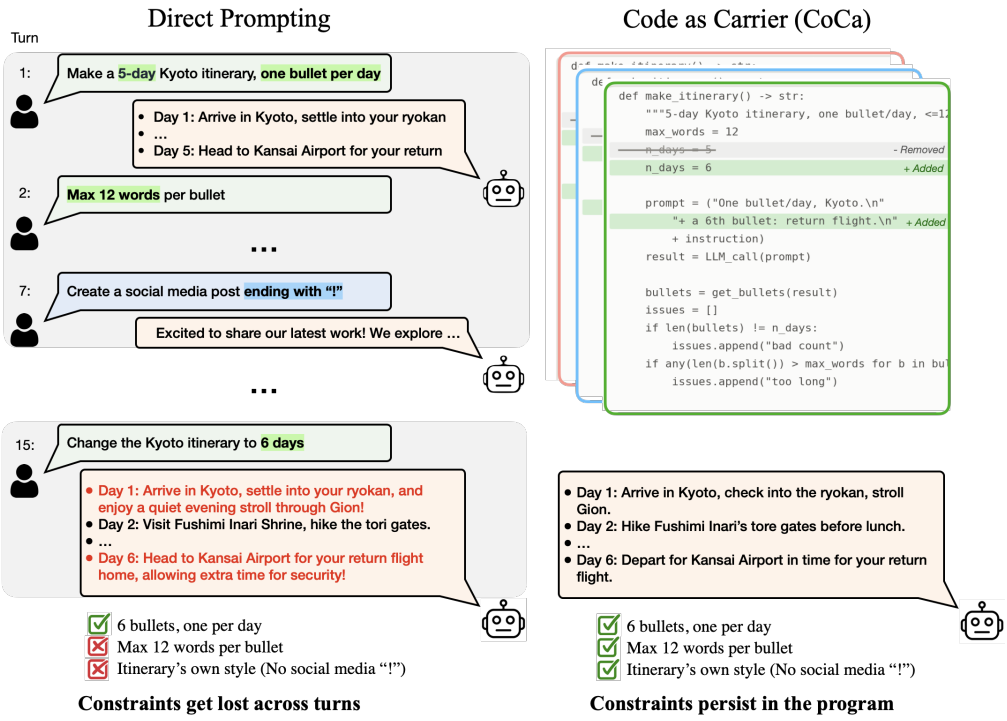

Figure 1: Code as Carrier (CoCa) preserves constraints in an executable program and enforces them at execution. *Left*: Direct prompting relies on dialogue history, causing constraints to be lost or misapplied. *Right*: CoCa retrieves correct topic thread and maintains active constraints in code, ensuring they are satisfied.

framework for conversation management that dynamically detects topic shifts, identifies user intents, and manages conversational context accordingly. Instead of relying solely on re-inference from raw and ever-growing transcripts, CoCa represents standing constraints for each topic as explicit and executable programs. These programs are updated incrementally as instructions are added, modified, or withdrawn, ensuring that all active constraints remain accessible as a structured state rather than being embedded implicitly in the dialogue history. Handling interleaved conversations, however, introduces an additional complexity: determining the appropriate topic thread for each turn. To address this, we augment the code-as-carrier mechanism with two supporting components: (1) a lightweight classifier that evaluates the contextual relevance of prior conversations to the current turn, and (2) a topic router that assigns each turn to the appropriate topic thread. Together, these components allow the model to consistently enforce the correct set of constraints, whether in long, evolving dialogues or when managing multiple interleaved topics. We summarize our main contributions as follows:

- We introduce CoCa, a framework that represents the standing constraints of a multi-turn dialogue as an executable program rather than embedding them in the transcript. Featuring three key modules: user intent classification, topic-aware routing, and code-as-carrier, each turn updates its topic thread's program as a delta, ensuring constraints are maintained by design rather than re-inferred from an expanding history.
- We empirically show that CoCa significantly enhances the multi-turn instruction-following capabilities of six open-weight LLMs, spanning five model families from 3B to 120B parameters, in both short and complex long-form conversations. Notably, with CoCa, a 3B model performs on par with, or even exceeds, much larger 120B+ models.
- Ablations show that CoCa reliably detects new topics, improves turn-wise retention by 10.6–172.0%, and reduces late-turn failure from 92% to 59%. Topic-aware routing is critical: 81% of misrouted turns fail, and disabling it reduces average endurance by 0.71 turns and maximum dialogue survival by around 18 turns. By updating programs in place, CoCa also keeps per-turn context approximately constant and uses 2.9× fewer tokens over 50 turns.

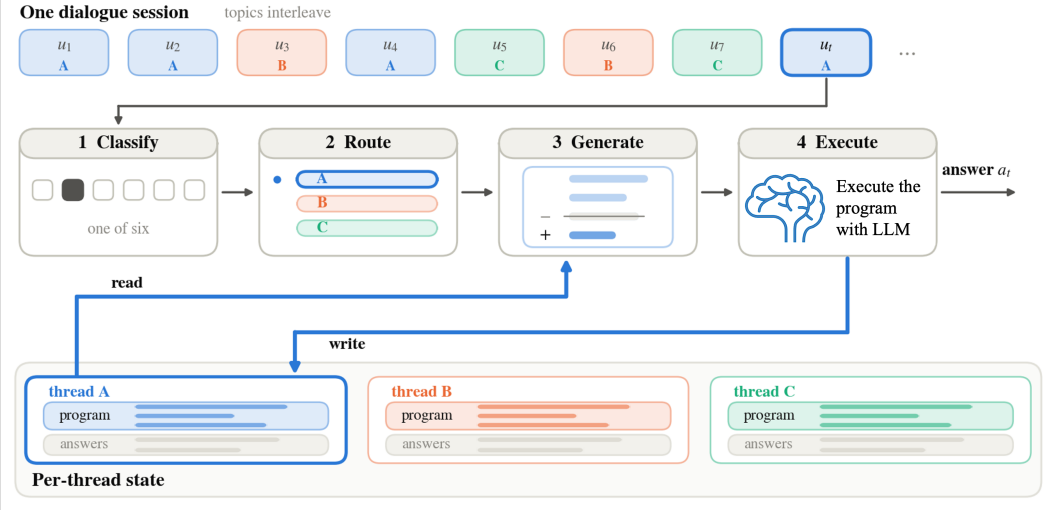

Figure 2: **Overview of CoCa.** *Top*: A dialogue session interleaves multiple topic threads (A, B, and C). *Middle:* Each user turn u_t is processed in four stages: intent classification, topic-aware context routing, program generation, and execution. *Bottom:* Each thread maintains its own state, consisting of an executable program that represents its active constraints and the corresponding answer history.

2 CoCa: Framework Overview

CoCa treats a multi-turn instruction-following conversation as a sequence of edits to an executable program, rather than as an ever-growing transcript. As illustrated in Figure 2, a dialogue session may interleave several topic threads, and each user turn may introduce, revise, or remove constraints associated with one of them. For a user turn u_t, CoCa performs four steps. **1) User intent classification** characterizes how the new turn relates to the existing conversation state. It assigns one of six categories: *unrelated*, *incremental*, *expanded*, *correction*, *repair*, or *related*, inspired by (Moore & Arar, 2018; Budzianowski et al., 2018). Appendix B.1 provides the category definitions and classifier details. Conditioned on the predicted intent, **2) Topic-aware context routing** identifies the topic thread to which u_t belongs and retrieves the corresponding thread state. It then selects the historical context supplied to the program generator, following the policy in Table 1. Note that the current user turn is always included plus the *additional* context made available to the generator. This category-dependent policy ranges from no historical context for a new topic (*unrelated* category) to the carrier program and complete answer history (*related* category) for turns that require broader conversational context. This design follows the general principle of retaining task-relevant conversational state rather than indiscriminately replaying the full transcript. It then by design largely compresses the input context length comparing to direct inference. **3) Program generator** (re)writes the thread's carrier program, based on the selected context (see Appendix B.3 for details), to reflect the new request. Constraints unaffected by the current turn are preserved, revised or retracted constraints are updated or removed, and newly introduced constraints are added. **4) Program execution** delegates semantic content generation to an LLM *executor* while checking or repairing mechanically enforceable constraints in code through a bounded validation-and-revision loop. The updated program replaces the previous program in the selected thread's state, and the new answer is appended to that thread's answer history.

Table 1: **User Intent category → additional generator context.** *Program* denotes the latest carrier program of the selected thread.

Category	Additional context
unrelated	None
incremental	*Program*
expanded	*Program*
correction	*Program*
repair	*Program* + *latest answer*
related	*Program* + *answer history*

Key design choices. Three aspects of CoCa are central to its operation.

- **User intent classification.** This LLM-based classifier determines how a user turn relates to the existing topic thread state and, consequently, how much prior context the LLM generator

requires. It prevents simple updates from unnecessarily reading the full dialogue while preserving additional context for turns that depend on previous responses.

- **Code as a constraint carrier.** Instead of repeatedly recovering active constraints from natural language history, CoCa maintains them in an executable program per topic. This representation supports both deterministic enforcement of mechanically checkable constraints and LLM-based generation for semantic constraints. Rewriting the carrier in place also bounds the context needed to represent the current constraint state.
- **Topic-aware context routing.** When a conversation interleaves multiple topics, CoCa maintains one carrier program and answer history per topic thread. An *unrelated* turn opens a new topic thread with an anchor as the topic identifier. This topic anchor is derived from the subject-bearing terms in the main function name of its initial program, for example, in Figure 1, the topic anchor of turn 1 will be *make itinerary*. For a subsequent turn (not classified as *unrelated*), the router compares the turn's content words (removing stop words) with the anchors of existing topic threads. It switches to an earlier thread only when there is a clear and unique match, otherwise, it retains the most recently active thread. This conservative policy avoids switching topics on ambiguous or tied matches.

3 EXPERIMENTS

In this section we show that CoCa closes most of the distance between a small executor and strong models two orders of magnitude larger. With CoCa, Granite 4.1 3B (IBM Research, 2026) approaches GPT-OSS-120B direct for its overall mean score in Multi-IF (He et al., 2024) benchmark, and reaches 77.93% CSR and 60.06% ISR on EvolIF (Jia et al., 2025), surpassing Kimi-K2[1] (1T total, 32B active) on every EvolIF metric (see Table 15). A 3B executor thus matches or beats systems with roughly 3–10$\times$ its active parameter count on multi-turn instruction following, which suggests that what limits small models here is not capacity but the absence of an explicit carrier for standing constraints and reliable topic routing mechanism.

Models and baselines. We evaluate six executors spanning five model families and sizes from 3B to 120B: Granite 4.1 3B and 8B (IBM Research, 2026), Llama 3.1 8B[2], Gemma 4 26B-A4B-it (Gemma Team, 2026), Qwen 3.5 27B (Qwen Team, 2026) and GPT-OSS-120B (Agarwal et al., 2025). We hold GPT-OSS-120B as the program generator and its prompt fixed across executors. Our baseline is *Direct* inference, in which the same executor receives the dialogue transcript without programmatic constraint carrying, thus, each comparison changes how prior context is represented while holding the executor fixed.

Benchmarks. We evaluate CoCa on two complementary multi-turn instruction-following benchmarks. **Multi-IF** (He et al., 2024) contains three-turn, single-topic dialogues in eight languages. Its constraints accumulate strictly: the response must satisfy every constraint introduced so far. It measures how well a system retains an increasingly demanding constraint set and enables analysis across languages. **EvolIF** (Jia et al., 2025) contains 150 longer (at least 50 turns) dialogues that interleave multiple topics within a conversation session, a harder setting for carrying standing constraints. A turn resumes a topic after several intervening turns, requiring the system to recover the correct topic-specific constraints. Thus, it additionally stresses models' capability of topic tracking and resumption over longer dialogues. See Appendix A.1 for more details.

Metrics. On Multi-IF, we follow IFEval-derived checkers (Zhou et al., 2023) and report strict and loose accuracy at both the prompt and instruction levels, **Overall** is their mean. We also report turn-wise **Retention**, defined as turn-3 divided by turn-1 strict-prompt accuracy, to measure performance decay as constraints accumulate. On EvolIF, we use its default scorer with patience $P_{\max}$=3 and report its four metrics: endurance ($\mathbf{EDR}_{\text{len}}$), cumulative constraint satisfaction ($\mathbf{EDR}_{\text{acc}}$), fully correct turns ($\mathbf{EDR}_{\text{succ}}$), and longest success streak ($\mathbf{EDR}_{\text{lss}}$). We additionally report micro-averaged constraint satisfaction (**CSR**) and turn-level instruction satisfaction (**ISR**), next-turn recovery after failure (**REC**), and macro-averaged per-dialogue success (**ROB**).

[1] https://kimik2.com/

[2] https://ai.meta.com/blog/meta-llama-3-1/

Table 2: CoCa versus Direct inference evaluating via Multi-IF. t1–t3: turns 1–3.

Executor	Setting	Overall by turn (%)			Mean	Strict prompt (%)		Reten-
		t1	t2	t3	overall	t1	t3	tion
Granite 4.1 3B	Direct	77.97	51.37	40.49	56.61	74.03	18.00	24.3%
	CoCa	**87.46**	**78.48**	**71.38**	**79.11**	**85.25**	**56.33**	**66.1%**
Granite 4.1 8B	Direct	80.71	61.17	49.87	63.91	76.05	27.56	36.2%
	CoCa	**89.05**	**80.62**	**73.34**	**81.00**	**86.65**	**58.56**	**67.6%**
Llama 3.1 8B	Direct	73.08	65.75	57.80	65.54	66.92	35.86	53.6%
	CoCa	**87.95**	**78.14**	**70.75**	**78.95**	**85.74**	**55.46**	**64.7%**
Gemma-4 26B-A4B	Direct	90.60	**84.41**	**78.56**	**84.52**	88.09	**64.05**	**72.7%**
	CoCa	**91.27**	83.81	77.13	84.07	**89.40**	63.82	71.4%
Qwen3.5 27B	Direct	89.43	80.00	73.00	80.81	86.14	55.39	64.3%
	CoCa	**91.51**	**84.19**	**76.89**	**84.20**	**89.54**	**63.67**	**71.1%**
GPT-OSS-120B	Direct	87.86	80.17	75.61	81.21	85.14	60.34	70.9%
	CoCa	**91.48**	**84.08**	**77.18**	**84.25**	**89.85**	**64.57**	**71.9%**

Table 3: CoCa versus Direct inference evaluating via EvolIF.

Executor	Setting	Endurance (turns)				CSR	ISR	REC	ROB
		$\mathbf{EDR}_{\text{len}}$	$\mathbf{EDR}_{\text{acc}}$	$\mathbf{EDR}_{\text{succ}}$	$\mathbf{EDR}_{\text{lss}}$	(%)	(%)	(%)	(%)
Granite 4.1 3B	Direct	6.75	4.21	2.87	1.99	62.35	42.45	14.12	32.68
	CoCa	**11.20**	**8.73**	**6.73**	**4.49**	**77.93**	**60.06**	**23.46**	**51.32**
Granite 4.1 8B	Direct	6.59	4.02	2.83	2.08	61.10	42.91	12.61	31.38
	CoCa	**11.25**	**8.86**	**6.71**	**4.39**	**78.74**	**59.69**	**24.10**	**51.04**
Llama 3.1 8B	Direct	6.45	3.88	2.69	2.04	60.24	41.68	14.34	31.95
	CoCa	**11.50**	**9.03**	**6.98**	**4.54**	**78.50**	**60.70**	**24.14**	**51.71**
Gemma 4 26B-A4B	Direct	9.71	7.39	5.49	3.67	76.15	56.52	18.96	46.02
	CoCa	**14.36**	**11.88**	**9.54**	**6.08**	**82.72**	**66.43**	**25.63**	**58.00**
Qwen 3.5 27B	Direct	9.50	6.84	5.19	3.48	72.02	54.67	20.85	42.88
	CoCa	**12.65**	**10.22**	**8.08**	**5.26**	**80.79**	**63.86**	**26.07**	**55.44**
GPT-OSS-120B	Direct	10.10	7.40	5.80	4.04	73.27	57.43	22.17	47.65
	CoCa	**14.79**	**12.45**	**10.03**	**6.35**	**84.21**	**67.81**	**27.74**	**59.61**

Main results. Tables 2 and 3 summarize the main results. On Multi-IF, CoCa substantially improves five of the six executors, with mean-overall gains ranging from 3.04 to 22.50 percentage points (pp). These gains arise primarily at later turns as turn-1 differences are generally small; for example, Granite 4.1 3B's improvement grows from 9.49 pp at turn 1 to 30.89 pp at turn 3. Correspondingly, the cross-executor spread of retention contracts from 48.4 to 7.2 pp, indicating that CoCa mitigates performance decay as constraints accumulate rather than merely improving initial-turn accuracy. Gemma 4 26B-A4B, the strongest direct inference model, is the sole exception, with a small mean-overall regression of 0.45 pp, suggesting limited headroom when direct retention is already high.

On EvolIF, CoCa improves all eight metrics for all six executors. The largest relative gains occur in persistence: average endurance ($\mathbf{EDR}_{\text{len}}$) increases by 54% and the longest success streak ($\mathbf{EDR}_{\text{lss}}$) nearly doubles, compared with gains of $13 - 15$ pp in CSR and ISR. Granite 4.1 3B with CoCa outperforms the direct inference with Gemma 4 26B-A4B and GPT-OSS-120B baselines on every reported metric, showing that programmatic constraint carrying can outweigh substantial differences in executor scale. REC improves least, however, indicating that restoring compliance immediately after a failed turn remains challenging even with programmatic enforcement.

Across both benchmarks, CoCa primarily improves persistence: on Multi-IF, its advantage grows at later turns as constraints accumulate, whereas on EvolIF, it sustains compliance for more turns despite topic switches and evolving constraints. The gains are also consistently larger for weaker executors: the cross-executor spread contracts from 27.91 to 5.30 pp (by 81%) in Multi-IF mean-overall accuracy and from 15.75 to 8.12 pp (by 48%) in EvolIF ISR. Thus, weaker and stronger executors converge toward a higher common performance level under CoCa.

Table 4 reports per-category constraint pass rates for EvolIF, aggregated across six executor models as mean $\pm$ SD. Overall, CoCa achieves its largest gains on Python-verifiable constraints, such as word existence (*existence*) and letter case (*case*). By contrast, performance consistently declines on

Table 4: **Per-category constraint pass rate.** Aggregated over the six executors of Table 3 as mean ± SD. The six Δ columns are shaded by magnitude (green gain, red loss); rows are sorted by mean Δ. Direct and CoCa pass rates are computed based on unmatched **total constraints (N)**: EvolIF's patience rule ($P_{\max}$=3) ends dialogues after three consecutive failures, so Direct contributes fewer per-category occurrences due to shorter surviving dialogues. *Macro-avg.*'s N is the grand total across all twelve categories.

Constraint	**Pass rate (%)**		**Total N**		**Δ per executor (pp)**					
	Direct	CoCa	Direct	CoCa	Gra. 3B	Gra. 8B	Llama	Gem. 4	Qwen	GPT-OSS
existence	22.0 ± 22.6	**61.7 ± 10.3**	1340	2484	**+56.0**	**+52.4**	**+49.1**	**+27.4**	**+29.7**	+23.9
case	49.1 ± 5.7	**84.7 ± 3.3**	1140	2068	**+34.7**	**+34.9**	**+39.5**	**+28.8**	**+35.2**	**+40.1**
format	62.6 ± 11.0	**83.5 ± 2.6**	999	1719	+17.5	**+35.4**	**+30.9**	+8.9	+19.8	+12.9
forbidden	79.7 ± 8.5	**97.0 ± 1.1**	1201	2223	+20.5	**+30.9**	+16.6	+7.9	+15.0	+12.8
punctuation	72.3 ± 8.1	**87.5 ± 1.1**	1061	2035	+16.0	+21.4	**+25.5**	+0.9	+14.3	+13.4
length	60.9 ± 3.7	**74.7 ± 3.9**	1329	2329	+9.8	+15.8	+16.4	+12.1	+12.2	+16.8
endwith	69.0 ± 9.1	**82.2 ± 1.3**	1407	2221	+14.9	+22.5	+22.4	−2.8	+8.3	+14.2
startwith	69.3 ± 7.7	**79.7 ± 2.1**	1237	2038	+14.1	+9.2	**+26.4**	+7.6	−1.3	+6.7
countableItems	65.2 ± 13.4	**75.5 ± 3.3**	1086	2046	+20.7	+16.0	+21.3	−4.2	+1.4	+6.6
emotion	77.2 ± 6.6	**83.8 ± 2.9**	1327	2160	+13.4	+5.5	+2.4	+4.0	−0.6	+14.9
reader_age	**69.8 ± 3.9**	61.7 ± 3.9	1683	2644	−6.3	−6.6	−9.4	**−11.5**	**−13.2**	−1.2
style	**81.8 ± 5.4**	73.7 ± 6.0	1441	2313	−5.1	**−11.5**	**−14.7**	−5.8	−8.4	−3.3
Macro-avg.	64.9 ± 7.2	**78.8 ± 2.5**	15251	26280	+17.2	+18.8	+18.9	+6.1	+9.4	+13.2

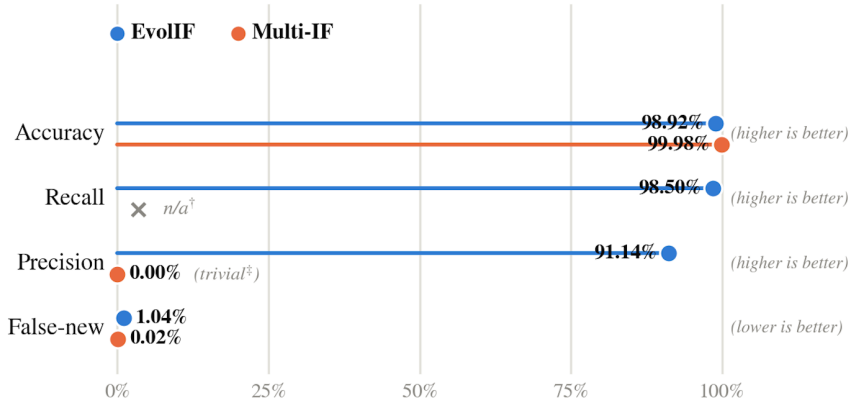

Figure 3: **New-vs-continuation call rates on non-first turns only.** Multi-IF's non-first turn are always continuations of the single topic conversation, thus, its recall is undefined and precision *is* defined to be zero.

non-Python-verifiable constraints, including *style* and *reader_age*, with low variance across executors. This pattern suggests that CoCa is most effective when a constraint can be translated into an assertion that the validate-and-rewrite loop can evaluate. Importantly, the Direct and CoCa pass rates are computed over unmatched populations, as indicated by the **Total N** columns. EvolIF's patience rule $P_{\max}$=3 terminates a dialogue after three consecutive failed turns. Consequently, Direct's shorter surviving dialogues (i.e., lower $\mathbf{EDR}_{\text{len}}$ in Table 3) cease contributing per-category instances earlier than those of CoCa. Across all categories, CoCa is evaluated on 1.57–1.92 × more constraints as Direct (summed across the six executors). Its per-category pass rates therefore incorporate a longer tail of later-turn instances that Direct does not reach.

All these results cannot be achieved simply by using GPT-OSS-120B as a stronger direct executor: CoCa paired with Granite 4.1 3B matches or exceeds direct inference with GPT-OSS-120B on both benchmarks. Section 4.4 further examines sensitivity of the program generator. Per-language Multi-IF results and more analysis are reported in Appendix A.2.

4 Ablation Studies

In this section, we present a comprehensive set of ablation studies to evaluate the key design choices of CoCa and demonstrate that its overall improvements depend on the contribution of each module.

4.1 User-Intent Classification

We first evaluate CoCa's user-intent classifier. As described in Section 2, the classifier assigns each user turn an intent category using a three-turn window of user prompts. A key decision is whether the turn starts a new topic (*unrelated*) or continues an existing one, since a false new-topic prediction creates a fresh program and discards the constraints accumulated in the current thread.

Figure 3 evaluates this decision on both benchmarks. We exclude the first turn of every dialogue because it clearly starts a new topic by construction, and score only turns $t \geq 2$. Multi-IF contains only single-topic dialogues, so every predicted new topic after the first turn is a false positive. The classifier makes this error on only 2 of $8,946$ turns, corresponding to a false-new rate of 0.02% and a new-topic classification accuracy of 99.98%. Although rare, such errors are consequential: incorrectly opening a new thread at turn 2 or 3 drops all previously accumulated constraints. For EvolIF, we derive the ground-truth new topic labels from the dataset's `active_topic` metadata. Among 731 non-initial turns that begin a new topic, the classifier identifies 720, corresponding to 98.50% recall. Its errors occur in both directions: it misses 11 genuine topic openings, and approximately one in eleven predicted openings is instead a continuation.

4.2 Effect of code-as-carrier

To isolate the effect of representing multi-turn constraints in code, we conduct an ablation study on EvolIF with the same generator (GPT-OSS-120B) and a weak executor (Granite 4.1 3B). The two settings differ only in the representation that carries prior constraints across turns: the previous assistant response in natural language (*prose*) or the updated executable program (*program*). As shown in Table 5[3], replacing prose with a program improves ISR by 11.39 percentage points (pp), ROB by 10.90 pp, and dialogue endurance by 2.57 turns. Under EvolIF's patience-based protocol ($P_{\max} = 3$), evaluation of a dialogue stops after the failure budget is exhausted. CoCa delays this cutoff, increasing the number of evaluated turns by 33%. Its aggregate rates are evaluated over later, more constraint-heavy turns than the prose carrier's. The turn-wise results in Table 6 further support the intended mechanism: at turn 1, before any prior constraints need to be carried, the two settings perform comparably; as constraints accumulate, the gap increases monotonically, reaching 33 pp by turn 4. Thus, CoCa's advantage emerges specifically when prior constraints must persist across turns. Further, even on shorter conversation like Multi-IF, where presenting full history of the conversation is not too costly, CoCa still boosts carried-over constraints pass rate for Granite 4.1 3B by up to 32.29 pp, see Table 2.

Table 5: Prose versus program on EvolIF

	Prose	Program	Δ
EDR$_{\text{len}}$	7.82	**10.39**	+2.57
EDR$_{\text{acc}}$	5.20	**7.80**	+2.60
EDR$_{\text{succ}}$	3.69	**6.08**	+2.39
EDR$_{\text{lss}}$	2.54	**4.22**	+1.68
CSR (%)	66.55	**75.12**	+8.57
ISR (%)	47.14	**58.54**	+11.39
REC (%)	18.48	**22.00**	+3.53
ROB (%)	40.37	**51.27**	+10.90

Table 6: Per-turn failure rate for Prose and Program

Turn	1	2	3	4	8	12	14
Prose	15	39	61	76	68	81	92
Program	15	**27**	**37**	**43**	**50**	**62**	**59**
Δ (pp)	0	−12	−24	−33	−18	−19	−33

4.3 Effect of Topic-Aware Routing

As we pointed out in Section 2, CoCa's router allows a new turn to identify, retrieve and update the appropriate state for its topic thread, even when that topic was last active several turns earlier. We evaluate this design choice with two ablations: **Rolling Turn Summary** maintains no carrier program, topic anchors, or persistent thread state, while **−Routing** retains the program carrier but disables topic-aware routing. Summarizing the three most recent user prompts[4] into a self-contained instruction context, **Rolling Turn Summary** tests whether a compact recency-based context already explains CoCa's gains, while **−Routing** isolates the benefit of retrieving the *correct* thread rather than simply carrying the state of the most recent one via code.

From Table 7, Granite 4.1 3B and Gemma 4 26B-A4B show a similar trend: **Rolling Turn Summary** recovers only about one-third of CoCa's ISR gain, in particular, it captures 35% of the gain on Granite

[3]The results in Table 3 are slightly higher because they use an improved topic routing method.

[4]CoCa's user intent classifier uses the same context window size.

Table 7: Isolating topic-aware routing on EvolIF

Model	Setting	Endurance (EDR)				CSR	ISR	REC	ROB
		len	acc	succ	lss	(%)	(%)	(%)	(%)
Granite 4.1 3B	Direct	6.75	4.21	2.87	1.99	62.35	42.45	14.12	32.68
	Rolling Turn Summary	7.91	5.31	3.85	2.67	67.15	48.65	19.08	38.51
	−Routing	9.70	7.19	5.51	4.11	74.17	56.77	20.50	48.95
	CoCa	**11.20**	**8.73**	**6.73**	**4.49**	**77.93**	**60.06**	**23.46**	**51.32**
Gemma-4 26B	Direct	9.71	7.39	5.49	3.67	76.15	56.52	18.96	46.02
	Rolling Turn Summary	10.65	8.08	6.32	4.41	75.87	59.36	24.97	52.15
	−Routing	11.30	8.98	7.09	5.37	79.47	62.71	22.51	57.63
	CoCa	**14.36**	**11.88**	**9.54**	**6.08**	**82.72**	**66.43**	**25.63**	**58.00**

Table 8: Generator sensitivity analysis on EvolIF

Generator	Exec.†	$\mathbf{EDR}_{\text{len}}$	$\mathbf{EDR}_{\text{lss}}$	CSR	ISR	REC	ROB
Direct (no generator)	–	9.71	3.67	76.15	56.52	18.96	46.02
Claude Haiku 4.5	97.52%	**14.48**	**6.31**	**83.44**	**67.17**	**27.00**	**58.88**
GPT-OSS-120B	97.06%	14.36	6.08	82.72	66.43	25.63	58.00
Qwen3-Coder-Next	96.12%	13.62	5.68	81.21	64.95	24.76	56.28

†Exec. is the fraction of turns whose generated program ran without exceptions.

and 29% on Gemma. Thus, across both weaker and stronger executors, a cleaner self-contained prompt explains only a small fraction of the improvement. Given a program carrier, correct topic routing provides a further, separable benefit. On Gemma 4, enabling topic routing accounts for $37-50\%$ of the corresponding Direct-to-CoCa gains. It increases $\mathbf{EDR}_{\text{len}}$ by 3.06 turns, more than the carrier-only increase over Direct (up by 1.59 turns), while for Granite the routing contributes only 1.50 turns against a 2.95 carrier-alone increment for $\mathbf{EDR}_{\text{len}}$. Overall, a compact summary of recent prompts is insufficient, most gains come from maintaining a persistent program carrier, while topic-aware routing provides an additional benefit by retrieving the appropriate thread state.

4.4 Generator sensitivity and cost analysis

Sensitivity of the program generator. To test whether CoCa depends on a particular program generator model, we fix the executor to Gemma 4 26B-A4B and vary the generator among Claude Haiku 4.5 (Anthropic, 2025) (closed-weight), GPT-OSS-120B (open-weight and used in our other experiments), and Qwen3-Coder-Next (Qwen Team) (open-weight and code-specialized). These models differ in size, accessibility, training, and specialization, the downstream executor remains fixed. As shown in Table 8, all three generators substantially outperform direct inference, yielding gains in both conversation durance and instruction following rates. Claude Haiku 4.5 performs best on every metric, followed consistently by GPT-OSS-120B and Qwen3-Coder-Next. Nevertheless, the best-to-worst gaps are only 2.22% in ISR and 2.60% in ROB, approximately one quarter of the weakest generator's gain over direct. This ranking also matches the generators' execution-success rates, suggesting that part of the performance gap arises from program reliability: a program that fails causes the entire turn to score zero, regardless of the quality of successfully executed programs. Overall, CoCa's gains persist across all three generators, indicating that gains are not specific to a single program generator model, although generator reliability affects the magnitude of the improvement.

Cost analysis. Carrying constraints in a program rather than in the transcript changes how context spending scales with dialogue depth. From Figue 4, we can see that direct prompting's per-turn context grows linearly (≈ 489 tokens/turn) and its cumulative spend grows quadratically, 611k input tokens for a single 50-turn dialogue. Instead, CoCa passes the prior *program* instead holds the per-turn context flat at ≈ 4.2k tokens, of which only 27% is carried state (code plus prior answer); the rest is a fixed in-context prompt. Both series count only the call that carries dialogue state, the re-fed transcript for direct, the generator prompt for CoCa, so Figure 4 (b) is a lower bound on CoCa's spend, which additionally issues an executor call per turn, which is merely the current user prompt.

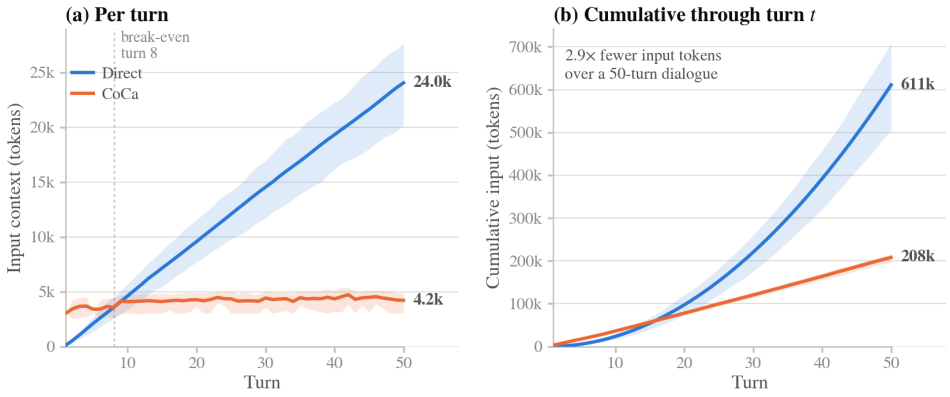

Figure 4: Input context per turn (a) and accumulated over a dialogue (b), for CoCa v.s. direct inference on EvolIF with Llama 3.1 8B.

5 Related Work

Multi-turn instruction following benchmarks. Early multi-turn evaluations such as MT-Bench (Zheng et al., 2023) and single-turn instruction-following suites such as IFEval (Zhou et al., 2023) established foundational methods for evaluating instruction-following capabilities of LLMs. More recent benchmarks He et al. (2024); Jia et al. (2025); Li et al. (2025); Deshpande et al. (2025) have shifted focus to studying dependencies between turns. Specifically, Multi-IF (He et al., 2024) evaluates models on incremental accumulation of constraints over a short, fixed number of turns, EvolIF (Jia et al., 2025) pushes this further by simulating long, evolving dialogues where topics interleave constraints dynamically change. They show a recurring pattern: as conversations lengthen, models consistently struggle to retain earlier constraints, leading to performance degradation.

Long context representation and management. LLMs with long input contexts often attend unevenly to content, favoring the beginning or end of prompts over the middle (Liu et al., 2024; Laban et al., 2026). Methods like retrieval-augmented generation (RAG) (Lewis et al., 2020; Izacard et al., 2023), paging conversational memory (Packer et al., 2023; Jung et al., 2026), or maintaining a persistent prose-form memory (Park et al., 2023) address this by managing historical context, but they still rely on representing constraints as raw text that the model must reprocess at every turn. In contrast, task-oriented dialogue systems utilize structured representations, such as turn-by-turn updates to a dialogue *state* or relation graph, which align more closely with our approach (Budzianowski et al., 2018; Peng et al., 2021; Li et al., 2026). However, these systems rely on rigid, predefined slot schemas and lack mechanisms for handling open-ended, revisable constraints or routing turns across multiple topics and tracking topic states.

Generative computing. Program-aided reasoning (Gao et al., 2023), code-generating embodied-control policies (Liang et al., 2023; IBM, 2025), and agents that build reusable code libraries over time (Wang et al., 2023) use generated code primarily for action or capability development, not for representing active constraints in a conversation. Moreover, these approaches assume a single ongoing task and do not address the challenge of interleaved-topic routing central to this paper. CoCa adapts the concept of code as an intermediate representation, but with a novel goal: serving as a dynamic carrier of standing constraints for distinct topic threads, updated incrementally as a delta at each turn, rather than being executed once and discarded.

6 Conclusion

We presented CoCa, a programmatic framework for multi-turn instruction following. It pairs code as carrier of each topic's active constraints with a lightweight user intent classifier and topic-aware router for context access and thread assignment. Experiments show that CoCa significantly improves small models' instruction following in complex multi-turn conversations, suggesting that explicit constraint carriers and reliable topic routing matter more than model capacity alone. Future work could explore whether specialized fine-tuning can reduce the generator model to the size of the executor.

AI USE STATEMENT

In this work, we used generative AI tools for implementing the proposed framework, generating tables and figures based on our experimental results and polishing the paper writing. We have not used generative AI tools for forming and brain-storming the ideas. We have reviewed all AI-assisted work, for example, all tables and figures and text are proof-read by all authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This study uses only publicly available benchmarks (EvolIF (Jia et al., 2025) and Multi-IF (He et al., 2024)) and involves no new data collection, human participants, or personally identifiable information. All executors and the default program generator are open-weight models; the generator-sensitivity experiment in Section 4.4 additionally uses Claude Haiku 4.5 through a paid, closed-model API. We do not foresee harms specific to this work beyond those already associated with the underlying language models it evaluates.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce our results by reference below. §3 gives the exact benchmarks and released scorers used, the metric definitions, and the executor/generator panel, including which models are open-weight and self-hostable and which are accessed via a closed API. §3 and §4 disclose which experimental variables are not held constant across settings and document properties of the judge-based scoring pipeline used for EvolIF. §4.1 and §4.3 report the measured stability of the upstream user intent labeler that our topic-routing and new topic detection results depend on. Exact per-run configuration files, job identifiers, and the inference, evaluation, scoring, and figure-generation code underlying every table and figure will be released later and cited in the supplementary material, so that each reported number can be regenerated from its own recorded invocation rather than reconstructed from the paper text alone.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Anthropic. Claude haiku 4.5, 2025. URL https://www-cdn.anthropic.com/7aad69bf12627d42234e01ee7c36305dc2f6a970.pdf.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. Multiwoz-a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 5016–5026, 2018.

Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E Primack, Summer Yue, and Chen Xing. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18632–18702, 2025.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pp. 10764–10799. PMLR, 2023.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Yun He, Di Jin, Chaoqi Wang, Chloe Bi, Karishma Mandyam, Hejia Zhang, Chen Zhu, Ning Li, Tengyu Xu, Hongjiang Lv, et al. Multi-if: Benchmarking llms on multi-turn and multilingual instructions following. *arXiv preprint arXiv:2410.15553*, 2024.

IBM. Mellea. https://mellea.ai/, 2025.

IBM Research. Granite 4.1 language models. https://huggingface.co/blog/ibm-granite/granit-4-1, 2026. Accessed: 2026-04-28.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43, 2023.

Qi Jia, Kaiwei Zhang, Xiujie Song, Ye Shen, Xiangyang Zhu, and Guangtao Zhai. One battle after another: Probing llms' limits on multi-turn instruction following with a benchmark evolving framework, 2025. URL https://arxiv.org/abs/2511.03508v2.

Yeongseo Jung, Jaehyeok Kim, Eunseo Jung, Jiachuan Wang, Yongqi Zhang, Ka Chun Cheung, Simon See, and Lei Chen. Context-driven incremental compression for multi-turn dialogue generation. *arXiv preprint arXiv:2606.12411*, 2026.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. In *International Conference on Learning Representations*, volume 2026, pp. 54738–54778, 2026.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Jinnan Li, Jinzhe Li, Yue Wang, Yi Chang, and Yuan Wu. Structflowbench: A structured flow benchmark for multi-turn instruction following. *arXiv preprint arXiv:2502.14494*, 2025.

Zhenhe Li, Can Lin, Ling Zheng, Wen-Da Wei, Junli Liang, and Qi Song. Graphif: Enhancing multi-turn instruction following for large language models with relation graph prompt. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31879–31887, 2026.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Robert J Moore and Raphael Arar. Conversational ux design: an introduction. In *Studies in conversational UX design*, pp. 1–16. Springer, 2018.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Baolin Peng, Chunyuan Li, Jinchao Li, Shahin Shayandeh, Lars Liden, and Jianfeng Gao. Soloist: Building task bots at scale with transfer learning and machine teaching. *Transactions of the Association for Computational Linguistics*, 9:807–824, 2021.

Qwen Team. Qwen3-coder-next technical report. Technical report. URL https://github.com/QwenLM/Qwen3-Coder/blob/main/qwen3_coder_next_tech_report.pdf. Accessed: 2026-02-03.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, et al. Benchmarking complex instruction-following with multiple constraints composition. *Advances in Neural Information Processing Systems*, 37:137610–137645, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

Table 9: Benchmark statistics for EvolIF and Multi-IF

	EvolIF	**Multi-IF**
Dialogs	150	4,501
Turns (labeled / nominal)	7,620 / 7,620	13,447 / 13,503
Turns per dialog	50–70 (avg 50.8)	3 (fixed)
Topic threads per dialog	2–14 (avg 5.87)	1 (single, cumulative)
Languages	1 (English)	8
Total constraints	26,399	32,070
Constraints per turn	avg 3.46 (flat)	1.39 / 2.39 / 3.39 (turns 1/2/3)
Split	eval-only	test-only

Table 10: **Multi-IF headline results**, 4,501 rows (4,445 at turn 3), same six executors and row order as Table 3. Overall is the mean of strict/loose × prompt/instruction accuracy; *retention* is turn-3 / turn-1 strict-prompt accuracy. Δ is CoCa − Direct.

Executor	**Setting**	**Overall by turn (%)**			**Mean**	**Strict prompt (%)**		**Reten-**
		t1	t2	t3	**overall**	t1	t3	**tion**
Granite 4.1 3B	Direct	77.97	51.37	40.49	56.61	74.03	18.00	24.3%
	CoCa	**87.46**	**78.48**	**71.38**	**79.11**	**85.25**	**56.33**	**66.1%**
	Δ	+9.49	+27.11	+30.89	+22.50	+11.22	+38.34	+41.8*pp*
Granite 4.1 8B	Direct	80.71	61.17	49.87	63.91	76.05	27.56	36.2%
	CoCa	**89.05**	**80.62**	**73.34**	**81.00**	**86.65**	**58.56**	**67.6%**
	Δ	+8.35	+19.45	+23.47	+17.09	+10.60	+31.00	+31.3*pp*
Llama 3.1 8B	Direct	73.08	65.75	57.80	65.54	66.92	35.86	53.6%
	CoCa	**87.95**	**78.14**	**70.75**	**78.95**	**85.74**	**55.46**	**64.7%**
	Δ	+14.87	+12.40	+12.94	+13.40	+18.82	+19.60	+11.1*pp*
Gemma-4 26B-A4B	Direct	90.60	84.41	78.56	84.52	88.09	64.05	72.7%
	CoCa	**91.27**	83.81	77.13	84.07	**89.40**	63.82	71.4%
	Δ	+0.67	−0.60	−1.43	−0.45	+1.31	−0.22	−1.3*pp*
Qwen3.5 27B	Direct	89.43	80.00	73.00	80.81	86.14	55.39	64.3%
	CoCa	**91.51**	**84.19**	**76.89**	**84.20**	**89.54**	**63.67**	**71.1%**
	Δ	+2.09	+4.19	+3.90	+3.39	+3.40	+8.28	+6.8*pp*
GPT-OSS-120B†	Direct	87.86	80.17	75.61	81.21	85.14	60.34	70.9%
	CoCa	**91.48**	**84.08**	**77.18**	**84.25**	**89.85**	**64.57**	**71.9%**
	Δ	+3.62	+3.91	+1.58	+3.04	+4.71	+4.23	+1.0*pp*

†Also the program generator for every row above; here it generates AND executes.

A Comprehensive Experimental Setup and Results

A.1 Benchmark Statistics

Table 9 presents dataset statistics of the two benchmarks: EvolIF Jia et al. (2025) and Multi-IF He et al. (2024). EvolIF contains 150 long dialogues, with 50–70 turns (50.8 on average) and 2–14 interleaved topic threads (5.87 on average) per dialogue. Its 26,399 constraints remain roughly constant across dialogue depth, averaging 3.46 constraints per turn. In contrast, Multi-IF contains 4,501 single-topic dialogues of up to three turns in eight languages (English and seven others). Because each turn adds an instruction while retaining all previous ones the average number of active constraints grows from 1.39 at turn 1 to 2.39 at turn 2 and 3.39 at turn 3, yielding 32,070 constraints. Of the nominal 13,503 turns, 13,447 are present because 56 dialogues lack a third-turn prompt.

A.2 Additional Experimental Results

We provide a list of more comprehensive results here. Table 10 and 11 are more detailed results of Table 2 and 3.

Table 2 presented aggregated results across all eight languages. Tables 12, 13 and 14 give the same six-executor, same-column breakdown as Table 2 for the three largest per-language panels: English (n=909), French (n=548), and Hindi (n=542), together 44.4% of the benchmark's rows.

Table 11: **CoCa v.s. Direct evaluating via EvolIF**. Δ is CoCa $-$ Direct.

Executor	Setting	Endurance (turns)				CSR	ISR	REC	ROB
		EDR_{len}	EDR_{acc}	EDR_{succ}	EDR_{lss}	(%)	(%)	(%)	(%)
Granite 4.1 3B	Direct	6.75	4.21	2.87	1.99	62.35	42.45	14.12	32.68
	CoCa	**11.20**	**8.73**	**6.73**	**4.49**	**77.93**	**60.06**	**23.46**	**51.32**
	Δ	+4.45	+4.52	+3.86	+2.50	+15.58	+17.61	+9.34	+18.64
Granite 4.1 8B	Direct	6.59	4.02	2.83	2.08	61.10	42.91	12.61	31.38
	CoCa	**11.25**	**8.86**	**6.71**	**4.39**	**78.74**	**59.69**	**24.10**	**51.04**
	Δ	+4.66	+4.84	+3.88	+2.31	+17.64	+16.78	+11.49	+19.66
Llama 3.1 8B	Direct	6.45	3.88	2.69	2.04	60.24	41.68	14.34	31.95
	CoCa	**11.50**	**9.03**	**6.98**	**4.54**	**78.50**	**60.70**	**24.14**	**51.71**
	Δ	+5.05	+5.15	+4.29	+2.50	+18.26	+19.02	+9.80	+19.76
Gemma-4 26B-A4B	Direct	9.71	7.39	5.49	3.67	76.15	56.52	18.96	46.02
	CoCa	**14.36**	**11.88**	**9.54**	**6.08**	**82.72**	**66.43**	**25.63**	**58.00**
	Δ	+4.65	+4.49	+4.05	+2.41	+6.57	+9.91	+6.67	+11.98
Qwen3.5 27B	Direct	9.50	6.84	5.19	3.48	72.02	54.67	20.85	42.88
	CoCa	**12.65**	**10.22**	**8.08**	**5.26**	**80.79**	**63.86**	**26.07**	**55.44**
	Δ	+3.15	+3.38	+2.89	+1.78	+8.77	+9.19	+5.22	+12.56
GPT-OSS-120B†	Direct	10.10	7.40	5.80	4.04	73.27	57.43	22.17	47.65
	CoCa	**14.79**	**12.45**	**10.03**	**6.35**	**84.21**	**67.81**	**27.74**	**59.61**
	Δ	+4.69	+5.05	+4.23	+2.31	+10.94	+10.38	+5.57	+11.96

†Also the program generator for every row above; here it generates AND executes.

Table 12: **Multi-IF, English only** ($n = 909$ rows). Same columns as Table 2.

Executor	Setting	Overall by turn (%)			Mean	Strict prompt (%)		Reten-
		t1	t2	t3	overall	t1	t3	tion
Granite 4.1 3B	Direct	85.75	65.75	51.54	67.68	82.40	29.13	35.4%
	CoCa	**90.42**	**83.17**	**75.99**	**83.20**	**88.12**	**62.50**	**70.9%**
	Δ	+4.67	+17.43	+24.46	+15.52	+5.72	+33.37	+35.6*pp*
Granite 4.1 8B	Direct	89.54	72.03	62.08	74.55	85.37	41.52	48.6%
	CoCa	**91.72**	**83.31**	**75.58**	**83.54**	**89.44**	**61.05**	**68.3%**
	Δ	+2.18	+11.28	+13.50	+8.99	+4.07	+19.53	+19.6*pp*
Llama 3.1 8B	Direct	83.16	75.79	68.26	75.74	78.22	49.44	63.2%
	CoCa	**91.55**	**81.89**	**75.50**	**82.98**	**89.66**	**62.28**	**69.5%**
	Δ	+8.39	+6.10	+7.24	+7.24	+11.44	+12.83	+6.3*pp*
Gemma-4 26B-A4B	Direct	92.57	86.86	82.68	87.37	90.32	69.64	77.1%
	CoCa	**93.54**	86.11	81.39	87.02	**91.97**	**69.87**	76.0%
	Δ	+0.97	−0.75	−1.29	−0.36	+1.65	+0.22	−1.1*pp*
Qwen3.5 27B	Direct	91.02	83.48	77.75	84.08	87.68	61.61	70.3%
	CoCa	**93.70**	**86.34**	**81.66**	**87.23**	**91.86**	**69.87**	**76.1%**
	Δ	+2.67	+2.86	+3.91	+3.15	+4.18	+8.26	+5.8*pp*
GPT-OSS-120B†	Direct	88.37	81.24	78.52	82.71	84.93	64.29	75.7%
	CoCa	**93.99**	**87.26**	**81.06**	**87.44**	**92.41**	**69.31**	75.0%
	Δ	+5.62	+6.02	+2.54	+4.72	+7.48	+5.02	−0.7*pp*

Direct baseline competence ranks the three languages consistently across most executors (with English >French >Hindi), with Qwen 3.5 showing only a minimal English/French gap of 0.03 points. For four of the six executors, CoCa's mean-overall gains align with this order, with the largest gains on Hindi and the smallest on English. Interestingly, the largest direct baseline language gap for weaker executors is also where the carrier mechanism provides the greatest improvements, supporting the hypothesis that enforcing constraints programmatically (via Python) helps to close instruction-following gaps misinterpreted as language gaps during direct inference. However, two executors break this ordering. GPT-OSS-120B completely inverts it, with gains of +2.33 (Hindi) <+3.35 (French) <+4.72 (English), with its prior exception on Hindi vs. English. Qwen3.5 retains the largest gain on Hindi (+9.69), but French falls slightly behind English (+2.20 vs. +3.15), reflecting a noise-prone regime where total gains are small (2-4 points) and differences are marginal. Notably, French is the only language where Gemma 4 does not regress overall (+0.41 mean-overall changes in Table 13, compared to −0.36 on English and −0.45 overall in Table 2). However, even

Table 13: **Multi-IF, French only** ($n = 548$ rows, $n = 518$ at turn 3). Same columns as Table 2. French is the only one of the three per-language tables where every executor's mean-overall Δ is positive, including Gemma-4's ($+0.41$) — though its turn-3 cell alone is still marginally negative (-0.08).

Executor	Setting	Overall by turn (%)			Mean	Strict prompt (%)		Reten-
		t1	t2	t3	overall	t1	t3	tion
Granite 4.1 3B	Direct	78.05	50.06	39.27	55.80	74.09	16.99	22.9%
	CoCa	**88.34**	**80.29**	**71.73**	**80.12**	**85.95**	**55.98**	**65.1%**
	Δ	+10.29	+30.23	+32.46	+24.32	+11.86	+39.00	+42.2*pp*
Granite 4.1 8B	Direct	81.14	59.54	47.32	62.67	76.28	24.32	31.9%
	CoCa	**88.15**	**82.92**	**74.39**	**81.82**	**85.22**	**59.85**	**70.2%**
	Δ	+7.01	+23.38	+27.07	+19.15	+8.94	+35.52	+38.3*pp*
Llama 3.1 8B	Direct	72.87	69.35	63.44	68.55	66.79	41.12	61.6%
	CoCa	**88.27**	**80.24**	**70.95**	**79.82**	**86.13**	**54.83**	**63.7%**
	Δ	+15.40	+10.89	+7.51	+11.27	+19.34	+13.71	+2.1*pp*
Gemma-4 26B-A4B	Direct	90.51	86.49	78.99	85.33	88.14	64.48	73.2%
	CoCa	**91.53**	**86.79**	78.91	**85.74**	**89.60**	**66.41**	**74.1%**
	Δ	+1.02	+0.30	−0.08	+0.41	+1.46	+1.93	+1.0*pp*
Qwen3.5 27B	Direct	90.71	84.41	77.03	84.05	87.96	61.20	69.6%
	CoCa	**92.08**	**87.87**	**78.80**	**86.25**	**90.33**	**66.99**	**74.2%**
	Δ	+1.38	+3.46	+1.77	+2.20	+2.37	+5.79	+4.6*pp*
GPT-OSS-120B†	Direct	88.01	82.01	74.87	81.63	85.77	59.27	69.1%
	CoCa	**91.82**	**86.61**	**76.52**	**84.98**	**90.15**	**64.09**	**71.1%**
	Δ	+3.80	+4.60	+1.65	+3.35	+4.38	+4.83	+2.0*pp*

Table 14: **Multi-IF, Hindi only** ($n = 542$ rows, $n = 529$ at turn 3). Same columns as Table 2. This is CoCa's single best language: the largest mean-overall gain of any (executor, language) cell in Tables 12, 13 and this one is Granite 4.1 3B's $+29.00$ here.

Executor	Setting	Overall by turn (%)			Mean	Strict prompt (%)		Reten-
		t1	t2	t3	overall	t1	t3	tion
Granite 4.1 3B	Direct	66.66	43.58	34.01	48.09	61.99	11.53	18.6%
	CoCa	**85.09**	**75.62**	**70.54**	**77.08**	**82.84**	**55.01**	**66.4%**
	Δ	+18.43	+32.03	+36.53	+29.00	+20.85	+43.48	+47.8*pp*
Granite 4.1 8B	Direct	66.02	53.25	42.85	54.04	59.59	19.66	33.0%
	CoCa	**87.08**	**80.66**	**72.66**	**80.13**	**84.32**	**56.14**	**66.6%**
	Δ	+21.06	+27.41	+29.82	+26.09	+24.72	+36.48	+33.6*pp*
Llama 3.1 8B	Direct	62.58	55.95	48.13	55.56	54.98	23.82	43.3%
	CoCa	**84.76**	**76.48**	**68.12**	**76.45**	**82.29**	**51.80**	**62.9%**
	Δ	+22.18	+20.53	+19.99	+20.90	+27.31	+27.98	+19.6*pp*
Gemma-4 26B-A4B	Direct	88.44	82.90	76.03	82.45	84.69	58.79	69.4%
	CoCa	**89.49**	**84.70**	76.01	**83.40**	**87.27**	**60.11**	68.9%
	Δ	+1.05	+1.80	−0.01	+0.95	+2.58	+1.32	−0.5*pp*
Qwen3.5 27B	Direct	83.39	70.51	62.67	72.19	78.41	39.70	50.6%
	CoCa	**88.16**	**82.24**	**75.23**	**81.88**	**85.24**	**59.55**	**69.9%**
	Δ	+4.77	+11.73	+12.55	+9.69	+6.83	+19.85	+19.2*pp*
GPT-OSS-120B†	Direct	87.33	78.90	73.53	79.92	84.32	56.14	66.6%
	CoCa	**88.67**	**83.55**	**74.52**	**82.25**	**86.90**	**60.11**	**69.2%**
	Δ	+1.34	+4.66	+0.99	+2.33	+2.58	+3.97	+2.6*pp*

†Also the program generator for every row above; here it generates AND executes.

for French, Gemma 4's performance is tenuous, with a turn 3 score still mildly negative (-0.08), sitting just one turn away from regression.

Overall, Hindi exhibits the largest gains with CoCa, making it the framework's best-performing language, while English consistently shows the smallest improvements. French consistently ranks between the two, matching its expected position based on both its competence and gain.

Table 15 compares CoCa with ten frontier models from Table 2 of Jia et al. (2025). CoCa carries GPT-OSS-120B to rank 3 overall, above every open-weight model in the source table (best: MiniMax-M2,

Table 15: **Our CoCa runs merged into the EvolIF leaderboard.** The ten frontier models are from Table 2 of Jia et al. (2025); the six shaded rows are ours from Table 3, with 3B–120B open-weight executors. All rows are evaluated with patience $P_{\max}$=3. Rows are ranked by ROB; **bold** and underline mark the best and second-best value per column. ΔROB is the gain over the same executor prompted directly.

#	Model	Endurance (turns)				CSR	ISR	REC	ROB	ΔROB
		$\mathbf{EDR}_{\text{len}}$	$\mathbf{EDR}_{\text{acc}}$	$\mathbf{EDR}_{\text{succ}}$	$\mathbf{EDR}_{\text{lss}}$	(%)	(%)	(%)	(%)	(pp)
1	GPT-5	**19.32**	**17.11**	**14.09**	**8.80**	**88.57**	**72.91**	**29.09**	**66.40**	–
2	Gemini-3.0-Pro	16.36	14.11	11.41	7.17	86.22	69.72	27.50	60.81	–
3	CoCa + GPT-OSS-120B	14.79	12.45	10.03	6.35	84.21	67.81	27.74	59.61	+11.96
4	CoCa + Gemma-4 26B-A4B	14.36	11.88	9.54	6.08	82.72	66.43	25.63	58.00	+11.98
5	CoCa + Qwen3.5 27B	12.65	10.22	8.08	5.26	80.79	63.86	26.07	55.44	+12.56
6	MiniMax-M2	11.75	9.19	7.17	4.77	78.22	60.98	24.29	54.54	–
7	CoCa + Llama 3.1 8B	11.50	9.03	6.98	4.54	78.50	60.70	24.14	51.71	+19.76
8	CoCa + Granite 4.1 3B	11.20	8.73	6.73	4.49	77.93	60.06	23.46	51.32	+18.64
9	CoCa + Granite 4.1 8B	11.25	8.86	6.71	4.39	78.74	59.69	24.10	51.04	+19.66
10	Kimi-K2	10.16	7.82	5.99	4.13	76.92	58.99	19.52	48.43	–
11	Qwen3-235B	10.02	7.66	5.80	3.97	76.43	57.88	21.15	47.47	–
12	Grok-4-Fast	9.52	7.29	5.50	4.13	76.58	57.77	16.01	46.03	–
13	DeepSeek-V3.2	8.64	6.32	4.62	3.38	73.15	53.47	15.87	44.42	–
14	Seed-1.6	8.21	5.78	4.20	2.95	70.44	51.18	15.90	39.43	–
15	Llama-4-Maverick	8.10	5.21	3.90	2.76	64.37	48.15	19.05	39.37	–
16	Mistral-Large-3	7.86	5.37	3.91	2.83	68.34	49.70	15.79	38.56	–

54.54% ROB), and all six CoCa runs outrank 7 of the ten, among them there are LLMs as large as hundred billion models.

B EXAMPLE INPUT OF COCA

B.1 USER INTENT CLASSIFIER

Each user turn is first passed to a classifier that assigns exactly one of seven discourse-act labels. These seven labels are mapped to six intent categories used in the main text; the mapping is indicated in parentheses below and summarized in Section B.2.

- FULL INQUIRY (Unrelated): A self-contained request on a NEW subject that does not build on the content of any earlier turn.
- RELATED INQUIRY (Related): A context-dependent request that reaches back BEYOND the immediately-prior turn.
- INCREMENTAL INQUIRY (Incremental): Asks for NEW content on the SAME ongoing subject as the immediately-prior turn.
- SELF-CORRECTION (Correction): Corrects a mistake in the user's OWN previous turn (e.g., "Oh wait, I meant Wednesday, not Thursday.").
- REPAIR INITIATOR (Repair): Does NOT ask for new content; it acts on EXISTING material. Either reworks the assistant's PRIOR (unseen) RESPONSE in place or changes a STANDING rule for FUTURE responses.
- DETAIL (Expanded): Supplies specific information the assistant asked for, OR adds a ONE-OFF specification/constraint to the CURRENT request (a length, a format, a missing value).
- OTHER (Related): None of the above.

classifier prompt: task framing + label definitions

```
You are a conversation analyst. Your task is to classify a single user turn.

Classify the target user turn into EXACTLY ONE of the labels below. The categories are mutually
exclusive | each definition states how it differs from the ones it is most easily confused with.

IMPORTANT: This conversation shows USER turns only | the assistant's replies happened but are
not included. Two consecutive user turns may have had an assistant response between them that
you cannot see. Judge the target from its own wording and the surrounding user turns; when the
target reacts to the assistant (agrees, asks for a revision, etc.), infer that from how the
user phrases it, not from visible assistant text.

TOPIC RULE: decide NEW SUBJECT vs CONTINUATION by the SUBJECT MATTER, never by formatting.
Reusing the same format/style requirements as earlier turns | "exactly N bullet points",
"use * or -", a word/character count, an emoji, a particular tone | is NOT topical continuity;
the same boilerplate recurs across unrelated subjects in this data. A turn that introduces a new
subject is a FULL INQUIRY even when it wraps that subject in the previous turns' formatting. A
turn is a CONTINUATION only when it depends on earlier CONTENT (it reuses, refers to, or resumes
a prior answer, task, or subject).

- FULL INQUIRY: A self-contained request on a NEW subject that does not build on the content of
  any earlier turn - it makes full sense on its own. New subject -> FULL INQUIRY, regardless of
  how much formatting it shares with earlier turns. A new subject is required: strip every
  requirement (length, format, wording, tone, what must appear or be avoided) and if what
  remains does not say what to write ABOUT, the turn names no subject of its own and is a
  continuation, however little it shares with earlier turns.
- RELATED INQUIRY: A context-dependent request that reaches back BEYOND the immediately-prior
  turn | either (a) an explicit call-back to an EARLIER, non-adjacent subject ("back to the ...
  we discussed", "the X you mentioned earlier"), or (b) a summary that consolidates MULTIPLE
  prior turns ("summarize what we've covered"). The distinction from INCREMENTAL is REACH: a
  turn that merely continues the immediately-prior subject is INCREMENTAL, not RELATED.
- INCREMENTAL INQUIRY: Asks for NEW content on the SAME ongoing subject as the immediately-prior
  turn, shifted to a related sub-area, scope, or angle (e.g., after minimalist MORNING routines:
  "now cover the EVENING routine instead"). The shift stays INSIDE the established subject. It
  differs from FULL (same subject, not a new one), from RELATED (continues the immediate turn,
  does not reach back to an earlier one), and from DETAIL (asks for new content, not just an
  added constraint).
- SELF-CORRECTION: Corrects a mistake in the user's OWN previous turn (e.g., "Oh wait, I meant
  Wednesday, not Thursday."). It fixes the user's own wording | not the assistant's answer (that
  is REPAIR) and not a standing rule.
- REPAIR INITIATOR: Does NOT ask for new content; it acts on EXISTING material. Either
    (a) reworks the assistant's PRIOR (unseen) RESPONSE in place | "revise", "rewrite",
        "rephrase", "shorten it", "change the tone of", "fix that", or a clarification of that
        answer ("can you explain that more simply?", "what did you mean by X?"); or
    (b) changes a STANDING rule for FUTURE responses | ADD ("from now on...", "always...", "every
        response must..."), REMOVE/relax ("forget the rule about...", "no more commas", "let that
        rest"), or SWAP ("instead of X, use Y", "switch the colon rule to no commas").
  A turn that asks for NEW content is NOT repair | see INCREMENTAL (new sub-area) or DETAIL
  (added constraint).
- DETAIL: Supplies specific information the assistant asked for, OR adds a ONE-OFF
  specification/constraint to the CURRENT request (a length, a format, a missing value) |
  constraint-stacking on the SAME subject. It differs from REPAIR because it does not rework the
  prior answer and does not set a standing "from now on" rule, and from INCREMENTAL because it
  adds a constraint rather than asking for a new sub-area of content.
```

```
- OTHER: You genuinely cannot tell which label applies. First check whether a filler, greeting,
  or persona-voiced turn actually conveys an instruction and label THAT; use OTHER only as a
  last resort, never as a convenient default.
```

Twelve worked examples follow the definitions, each a short (context, target) pair with its gold label.

classifier prompt: worked examples (ICL)

```
---

### Examples (context blocks show user turns only; assistant replies are not shown):

Example 1 (RELATED INQUIRY | call-back to an EARLIER, non-adjacent subject: the immediately-prior turn (turn
  ↪  2) is a different subject, so the target reaches back PAST it to the turn-1 thread. That reach is
  ↪  what makes it RELATED | had turn 2 also been about the French Revolution, the same question would be
  ↪  INCREMENTAL):
  Context:
    1: "user: Give a 150-word overview of the causes of the French Revolution, in a neutral encyclopedic
      ↪  tone."
    2: "user: Separately, list five beginner tips for indoor rock climbing, as a numbered list."
  Target:
    3: "user: Going back to the French Revolution from earlier | who were its three most influential figures,
      ↪  and why? Answer in under 90 words."
  {"label": "RELATED INQUIRY"}

Example 2 (REPAIR INITIATOR | asks to redo/revise the assistant's PRIOR (unseen) RESPONSE; "the previous
  ↪  draft" gives it away):
  Context:
    1: "user: Please draft a short professional email to my team announcing next week's offsite agenda."
  Target:
    2: "user: Could you adjust the previous draft to suit a more professional tone suitable for our
      ↪  workplace standards? Also, keep it under 150 words and include the keywords 'insightful' and
      ↪  'networking opportunities'."
  {"label": "REPAIR INITIATOR"}

Example 3 (REPAIR INITIATOR | "revise / rewrite the previous X" pattern, even with added constraints):
  Context:
    1: "user: Compose a poem about the wisdom of nature using vivid imagery."
  Target:
    2: "user: Please revise the previous poem to adopt a more reflective and introspective tone. Use a
      ↪  template starting with 'In solitude...' and ending with 'Thus, nature's wisdom bestowed upon
      ↪  all.'"
  {"label": "REPAIR INITIATOR"}

Example 4 (REPAIR INITIATOR | manages the standing instruction set by ADDING / REMOVING / SWAPPING a global
  ↪  rule; a confirmation tag does not change this):
  Context:
    1: "user: Write a project proposal on using AI to improve education in developing countries."
    2: "user: From now on, please make sure every response includes the '%' punctuation mark."
  Target:
    3: "user: Let's switch that up | instead of needing the '%' mark, the response must now be completely
      ↪  free of any ',' commas. Is that clear?"
  {"label": "REPAIR INITIATOR"}

Example 5 (INCREMENTAL INQUIRY | SAME subject (minimalism), shifted to a new sub-area (evening vs morning);
  ↪  NOT a new topic and NOT a redo):
  Context:
    1: "user: Building on the earlier discussion on minimalism, I would like to delve into specific
      ↪  minimalist morning routines. Could you illustrate these routines in a Markdown table?"
  Target:
    2: "user: In revisiting our conversation about minimalism, I am now interested in simplifying my evening
      ↪  routines rather than the morning ones. Please provide insightful examples of minimalist evening
      ↪  practices in one paragraph."
  {"label": "INCREMENTAL INQUIRY"}

Example 6 (SELF-CORRECTION | fixes the user's OWN prior turn, NOT the assistant's response):
  Context:
    1: "user: Book me a table for four on Thursday evening at an Italian place downtown."
  Target:
    2: "user: Oh wait, I meant Wednesday, not Thursday."
  {"label": "SELF-CORRECTION"}

Example 7 (RELATED INQUIRY | summary request that consolidates the prior conversation):
  Context:
    1: "user: Explain the first law of thermodynamics."
    2: "user: Now explain heat versus work."
  Target:
    3: "user: To solidify my understanding, could you provide a concise summary that consolidates our
      ↪  discussions on the first law of thermodynamics, work, heat, and internal energy? Keep it under
      ↪  100 words."
  {"label": "RELATED INQUIRY"}

Example 8 (FULL INQUIRY | completely new topic with no reference to prior conversation):
  Context:
    1: "user: got it, how does erosion change these mountains? use a template and keep it like, 3
      ↪  sentences."
  Target:
    2: "user: hey, what's different with intel and amd processors? like, just the basics."
  {"label": "FULL INQUIRY"}

Example 9 (DETAIL | supplies specifics in response to the assistant's (unseen) request for more info):
  Context:
    1: "user: Draft a cover letter for a software engineering role."
  Target:
```

```
    2: "user: It's for a senior backend position at a fintech startup, 5 years of Go experience, and keep it
      ↪ under 250 words."
  {"label": "DETAIL"}

Example 10 (FULL INQUIRY | the Target switches to a NEW subject (a banana-bread recipe) even though it
  ↪ reuses the prior turns' "exactly N bullet points, using -" formatting; the shared formatting is
  ↪ boilerplate, not topical continuity. NOTE: only the Target turn is classified | the context turns are
  ↪ shown unlabeled, and turn 2 here would itself be a REPAIR INITIATOR):
  Context:
    1: "user: Summarize the rules of chess for beginners in exactly 6 bullet points, using - for each
      ↪ bullet."
    2: "user: Make it exactly 4 bullet points instead, still using -."
  Target:
    3: "user: Now give me a recipe for banana bread in exactly 4 bullet points, using - for each bullet."
  {"label": "FULL INQUIRY"}

Example 11 (INCREMENTAL INQUIRY | the Target stays on the SAME subject (the rules of chess), shifted to a
  ↪ sub-area (castling) and reusing the earlier formatting; it is a continuation because the SUBJECT is
  ↪ unchanged. Contrast with Example 10, where the subject changed):
  Context:
    1: "user: Summarize the rules of chess for beginners in exactly 6 bullet points, using - for each
      ↪ bullet."
  Target:
    2: "user: Now cover the rules specific to castling instead, keeping exactly 6 bullet points using -."
  {"label": "INCREMENTAL INQUIRY"}

Example 12 (DETAIL | the Target states ONLY requirements and names no subject of its own: strip them and
  ↪ nothing is left saying what to write ABOUT, so it cannot be a FULL INQUIRY however little it shares
  ↪ with turn 1. It attaches to the standing subject as an added one-off requirement; it would be REPAIR
  ↪ INITIATOR only if it set a standing "from now on" rule or reworked the prior answer):
  Context:
    1: "user: Write a short guide to composting kitchen scraps for apartment dwellers."
  Target:
    2: "user: Keep it under 200 words and include the word 'balcony' at least twice."
  {"label": "DETAIL"}
```

The definitions and examples above are then filled into the template below to classify every turn in a conversation.

classifier prompt: per-turn context/target and output format

```
---

### Conversation context (user turns only; assistant replies are not shown):
<context turns, most recent 3 user turns, one per line, numbered>

### Target turn to classify:
<the turn being labelled, same numbered format>

### Output format:
Respond with only a JSON object: {"label": "<LABEL>"}
```

B.2 DISCOURSE-ACT LABEL DISTRIBUTION ON EVOLIF AND MULTI-IF

Table 16 counts what the classifier prompt in Appendix B.1 actually outputs on every user turn of both benchmarks. They are later mapped onto the six intent categories of Figure 2 with DETAIL → *expanded*, FULL INQUIRY → *unrelated*, REPAIR INITIATOR → *repair*, INCREMENTAL INQUIRY → *incremental* and RELATED INQUIRY, SELF CORRECTION → *correction*, and OTHER both → *related*. The two distributions look almost nothing alike, and the reason is structural rather than a difference in classifier behaviour: EvolIF uses all seven labels, dominated by REPAIR INITIATOR (55.2%) and DETAIL (28.2%), reflecting 150 dialogs of 50+ turns each in which a topic's constraint set is revised and re-revised many times before the dialogue moves on. Multi-IF uses effectively three labels (FULL INQUIRY, DETAIL, REPAIR INITIATOR account for 99.98% of turns) and never emits RELATED INQUIRY or SELF-CORRECTION at all, because a 3-turn, single-topic, cumulative-by-construction dialogue never presents a turn with an earlier, non-adjacent topic to call back to, nor a turn correcting the user's own prior wording.

Multi-IF's near-collapse to three labels is itself an artefact of turn position, not topic content: Table 17 breaks its distribution down by turn. Turn 1 is FULL INQUIRY 98.78% of the time, turn 2 is DETAIL 95.11% of the time, and turn 3 is DETAIL 90.71%/REPAIR INITIATOR 9.29% — so on Multi-IF the turn INDEX alone predicts the label almost as well as the classifier does, which is the qualitative shape behind why Multi-IF's labels carry little independent routing signal beyond "which turn number is this".

Table 16: **User intent classifier label distribution**, counted over every labeled user turn (EvolIF: 7,620 turns, 150 dialogs; Multi-IF: 13,447 turns, 4,501 dialogs)). Rows follow the label definitions' decision order.

	EvolIF		Multi-IF	
Label	n	(%)	n	(%)
Full Inquiry	930	12.20	4,448	33.08
Related Inquiry	178	2.34	0	0.00
Incremental Inquiry	85	1.12	1	0.01
Self-Correction	50	0.66	0	0.00
Repair Initiator	4,209	55.24	676	5.03
Detail	2,153	28.25	8,321	61.88
Other	15	0.20	1	0.01

Table 17: **Multi-IF label distribution by turn position**. Blank cells are exactly zero.

Label	**Turn 1**	**Turn 2**	**Turn 3**
Full Inquiry	98.78%	0.04%	–
Detail	0.18%	95.11%	90.71%
Repair Initiator	1.02%	4.82%	9.29%
Incremental Inquiry	0.02%	–	–
Other	–	0.02%	–

B.3 Example input of Program generation module

A `related` turn builds on, or calls back to, a topic already discussed in the current session. It is the user intent type category with the widest prior-state grant, i.e., it reads both the thread's standing program and that thread's all previous assistant answers, so its input shows every moving part of the carrier at once. The generator receives a single user prompt,

$$\underbrace{\text{instructions} + 3 \times \text{worked example}}_{\text{fixed ICL block, 21,380 chars}} + \underbrace{\texttt{<Program>} + \texttt{<PriorAnswers>} + \texttt{<TurnType>} + \texttt{<UserTurn>}}_{\text{live block, 4,848 chars}}$$

and continues it at a trailing `<Pseudocode>` tag. Only the live block varies from turn to turn: the ICL block is identical for every `related` turn in every dialogue, and at 82% of the 26,229-char prompt it is both the bulk of the input and a constant one. What grows with dialogue depth is only the live block, and the two components that could grow without bound is the program and the carried answers, which are each bounded by construction, which is why the per-turn context in Figure 4 stays flat instead of accumulating.

B.4 Instructions

The preamble names the three inputs and states the central rule: read the standing constraints *off the program* and carry them forward unless the turn changes them.

ICL instructions

```
You are provided the prior Program for the referenced topic (<Program>), the prior assistant replies for
   ↪  that topic (<PriorAnswers>), and the user's latest message (<UserTurn>), categorised as <TurnType> =
   ↪  "Related". Your task is to generate a Python program that uses LLM_call() and standard Python to
   ↪  handle the new user request.

WHAT EACH CONTEXT BLOCK IS FOR:
- <Program> is the program generated for the most recent turn of the topic this message refers to. It is the
   ↪  FAITHFUL, LOSSLESS record of the STANDING CONSTRAINTS in force on that topic (validators and
   ↪  constraint literals | required/forbidden keywords, markers, casing, counts, length, audience, format).
   ↪  Read the standing constraints off the <Program> and carry them forward unless the <UserTurn> changes
   ↪  them.
- <PriorAnswers> holds the CONTENT (facts and conclusions) the new request builds on or calls back to.

CRITICAL RULES:
```

```
1. The <PriorAnswers> block contains the referenced topic's assistant responses, each labelled "[Assistant
   ↪  answer N]" in turn order. It holds the facts and conclusions the user's new request builds on. A
   ↪  "Related" turn may build on the most recent answer OR call back to an EARLIER answer | read the
   ↪  <UserTurn> to decide which answer(s) it references, and pull facts from the right one(s).
2. The <UserTurn> is the user's latest message, categorised as <TurnType> = "Related". Carry the standing
   ↪  constraints from <Program> and apply the turn's delta: ADD a validator for a new constraint, REPLACE
   ↪  one whose parameter changed, REMOVE one the user relaxed | keep every other validator from <Program>
   ↪  unchanged. (When no prior <Program> is available the block is omitted; rely on <PriorAnswers> alone.)
3. The generated function MUST take NO parameters and MUST NOT rely on any pre-defined runtime variables.
   ↪  Extract the relevant content from the correct <PriorAnswers> entry/entries AND the carried
   ↪  constraints from <Program> at code-generation time and bake them into your generated code as Python
   ↪  literals (variables, lists, short summary strings, validators).
4. Keep total LLM_call() invocations low (ideally 1-3, at most 5). Do NOT use while loops around LLM_call().
   ↪  Batch-validate and do a single rewrite if needed. Prefer Python-based mechanical fixes over
   ↪  additional LLM_call() invocations.
5. Treat every LLM_call() result as untrusted: it may be empty, truncated, or malformed. After each
   ↪  LLM_call(), check the result; if empty/malformed, retry ONCE with a simpler prompt. If retries are
   ↪  exhausted, fall back to the LAST non-empty draft from a prior LLM_call() | do NOT hardcode answer
   ↪  text. Python literals are reserved for STRUCTURE and CONSTRAINTS (word/line counts, required keywords,
   ↪  forbidden terms, headers, validators, persona/audience, content extracted verbatim from the prior
   ↪  turns) and must never contain the answer content itself.
6. Read the new turn's intent before drafting the prompt. A "Related" turn is sometimes a build-on (new
   ↪  angle, format, or topic expansion), sometimes a transform of the prior (rephrase, reframe, change
   ↪  voice on the same topic), and sometimes a call-back that re-engages an EARLIER topic (e.g. "remember
   ↪  those X you mentioned?"). For transform-style and call-back turns, the referenced prior topic anchors
   ↪  must be preserved and the response must not pivot to a different subject.

Per-category guidance for <TurnType> = "Related":
- Use <PriorAnswers> at code-generation time as the source of facts/conclusions to reference. Identify which
   ↪  prior answer(s) the turn refers to (the latest, or an earlier one for a call-back), then hard-code
   ↪  the relevant facts (entity names, conclusions, key numbers) from THAT answer as Python literals in
   ↪  your generated code.
- Do NOT call LLM_call() to re-derive prior context | you already have it in <PriorAnswers>; bake it in.
- Validators should focus on the new turn's specific constraints (format, persona, keyword frequency,
   ↪  headers, length) AND on continuity with the referenced prior answer when the turn is a
   ↪  transform/reframe/call-back.

Examples are enclosed within <Example> and </Example> tags. Each example has <PriorAnswers>, <TurnType>,
   ↪  <UserTurn>, and <Pseudocode> sections.
```

B.5 WORKED EXAMPLES

Three worked examples follow, each demonstrating the same discipline: carry every validator out of `<Program>` unchanged, apply the turn's delta, and bake facts taken from `<PriorAnswers>` into the new program as literals rather than re-deriving them with a model call. The remaining 2 are structurally identical and differ only in which constraint is carried, so they are elided here; the build script's `--all-examples` flag emits them in full.

ICL worked example 1 of 3

```
<Example>
<Program>
def recommend_workout_music() -> str:
    """Workout-music recommendations. Standing constraint:
    - must include the keyword 'rhythm'   (existence/required keyword)"""
    required = ["rhythm"]
    prompt = (
        "Recommend electronic music subgenres for workouts and explain why each fits. "
        "Make sure the word 'rhythm' appears naturally."
    )
    result = LLM_call(prompt)
    if not result or not result.strip():
        result = LLM_call("Recommend workout music subgenres; include the word 'rhythm'.")
    result = result.strip()
    for k in required:
        if k.lower() not in result.lower():
            result = LLM_call(f"Rewrite to naturally include the word '{k}':\n" + result) or result
    return result.strip()

# program
{
    "method": "recommend_workout_music",
    "return": "str",
    "execute": "recommend_workout_music()"
}
</Program>
<PriorAnswers>
[Assistant answer 1]
For workouts, House, Drum & Bass, and Techno tend to be the top picks. House keeps a consistent 120-130 BPM
   ↪  that matches a steady jogging or cycling pace. Drum & Bass runs at 160-180 BPM, perfect for
   ↪  high-intensity intervals and sprints. Techno provides relentless, driving energy at 130-150 BPM that
   ↪  works great for sustained cardio sessions. All three maintain a strong, predictable beat that helps
   ↪  you lock into a rhythm and push through fatigue.

[Assistant answer 2]
As for streaming services, Spotify has the largest curated workout-playlist catalogue and good cross-device
   ↪  sync, while Apple Music offers lossless audio and tight integration with the Fitness app. Both let
   ↪  you download tracks for offline gym sessions.
</PriorAnswers>
```

```
<TurnType>Related</TurnType>
<UserTurn>
Cool, now pretend you're a DJ and give me the scoop on those three workout genres you mentioned earlier. Use this format for each: '[Genre Name]... [Vibe]... [Workout Benefit]'. Summarize what we discussed so far from your DJ perspective.
</UserTurn>
<Pseudocode>
import re

def generate_dj_summary_response() -> str:
    """DJ-perspective recap of the three workout subgenres in
    '[Genre Name]... [Vibe]... [Workout Benefit]' format, plus a DJ-style summary.
    Carries the standing constraint from <Program> (must include the keyword 'rhythm')
    and adds this turn's new constraints (DJ persona + per-genre template + summary)."""
    # Carried constraint (from <Program>), unchanged:
    required = ["rhythm"]
    # Related call-back: the turn says "those three workout genres you mentioned EARLIER",
    # so the facts come from [Assistant answer 1] (the genres), NOT [Assistant answer 2]
    # (streaming services). Extract them at code-generation time and bake them in.
    workout_subgenres = [
        ("House",        "120-130 BPM", "steady jogging or cycling pace"),
        ("Drum & Bass",  "160-180 BPM", "high-intensity intervals and sprints"),
        ("Techno",       "130-150 BPM", "sustained cardio sessions"),
    ]
    # matches a line that contains at least one [bracketed] field
    template_pattern = re.compile(r"\[.+?\]", re.IGNORECASE)

    prior_summary_lines = [
        f"- {name}: {bpm}, suits {use}." for name, bpm, use in workout_subgenres
    ]
    prompt = (
        "You are a DJ sharing your insider perspective. From the previous discussion, "
        "the three best electronic music subgenres for workouts are:\n"
        + "\n".join(prior_summary_lines) + "\n\n"
        "Write in first person with an energetic, casual DJ voice "
        "(e.g., 'I always spin...', 'In my experience...', 'Let me tell you...'). "
        "For each of the three subgenres, output one line in this exact format:\n"
        "[Genre Name]... [Vibe]... [Workout Benefit]\n\n"
        "Then add a brief DJ-perspective summary (2-3 sentences) of why these three work "
        "across different workout styles."
    )
    result = LLM_call(prompt)

    # Empty-response defence: drop the formatting scaffolding and ask for the core ask.
    if not result or not result.strip():
        result = LLM_call(
            "As a DJ, write a first-person recap of three workout music subgenres: "
            + ", ".join(name for name, _, _ in workout_subgenres) + "."
        )

    # Persona check: first-person voice in the opening.
    first_person_markers = ["i ", "i'm", "i've", "my ", "me "]
    has_first_person = any(marker in result.lower()[:200] for marker in first_person_markers)

    # Format check: each subgenre appears in a templated line.
    sub_names = [name for name, _, _ in workout_subgenres]
    all_covered = all(
        any(sub.lower() in line.lower() for line in result.split("\n") if template_pattern.search(line))
        for sub in sub_names
    )

    # Carried constraint (from <Program>): the required keyword must still be present.
    has_required = all(k.lower() in result.lower() for k in required)

    # ONE combined rewrite if any check fails.
    if not has_first_person or not all_covered or not has_required:
        missing = [s for s in sub_names if s.lower() not in result.lower()]
        missing_kw = [k for k in required if k.lower() not in result.lower()]
        rewritten = LLM_call(
            "Rewrite the following response to fix these issues:\n"
            "1. Write as a DJ in first person (use 'I', 'my', etc.).\n"
            "2. Ensure ALL three workout subgenres (House, Drum & Bass, Techno) each appear in "
            "a line with the format '[Genre Name]... [Vibe]... [Workout Benefit]'.\n"
            + ("3. Missing subgenres: " + ", ".join(missing) + ".\n" if missing else "")
            + ("4. Include the keyword(s): " + ", ".join(missing_kw) + ".\n" if missing_kw else "")
            + "\nOriginal:\n" + result
        )
        if rewritten and rewritten.strip():
            result = rewritten

    return result.strip()

# program
{
    "method": "generate_dj_summary_response",
    "return": "str",
    "execute": "generate_dj_summary_response()"
}
</Pseudocode>
</Example>
```

B.6 THE LIVE BLOCK

Three things in the listing are worth reading closely. First, the <Program>'s docstring *is* the standing constraint set: subject, five forbidden words, required mood, and each constraint that admits a mechanical test appears twice over: once as a literal and once as a check that repairs its own violation. Second, <PriorAnswers> shows the length bound in both of its branches: the 4 most recent answers appear in full, each capped at $2,000$ characters so that one pathological answer cannot crowd out the rest, while older ones are reduced to a 150-character head that states the full length. Older answers are shortened, never dropped, so a call-back to an early turn still finds that content represented while the block stays bounded. Third, this turn's delta *removes* a standing constraint rather than adding one. That is the case a transcript carries worst, whereas in the carrier it is a deletion, and the constraints that remain are still stated in full.

live block (real run: dialog_65, turn 48)

```
<Example>
<Program>
import re

def riddle_for_façade() -> str:
    """Generate a riddle whose answer is the word 'façade', ensuring the words
    'exterior', 'riddle', 'structure', 'deceive', and 'mask' do not appear in the text,
    and convey a sad, rainy mood."""
    answer = "façade"
    forbidden = ["exterior", "riddle", "structure", "deceive", "mask"]
    sad_keywords = ["sad", "melancholy", "gloomy", "rain", "grey", "tears", "dreary"]

    # Initial prompt asking for a sad-tone riddle
    prompt = (
        f"Create a short, clever riddle whose answer is the word '{answer}'. "
        "The riddle should hint at the meaning of a decorative front or outward appearance, "
        "and it must convey a sad, rainy, gloomy mood. "
        f"Do NOT use any of the words {', '.join(repr(w) for w in forbidden)}."
    )
    result = LLM_call(prompt)
    if not result or not result.strip():
        result = LLM_call(
            f"Write a short sad-tone riddle whose answer is '{answer}'. "
            f"Avoid the words {', '.join(repr(w) for w in forbidden)}."
        )
    result = result.strip()

    # Mechanical forbidden-word repair
    def contains_forbidden(text: str) -> list:
        hits = []
        for w in forbidden:
            if re.search(rf"\b{re.escape(w)}\w*\b", text, re.I):
                hits.append(w)
        return hits

    hits = contains_forbidden(result)
    if hits:
        rewrite_prompt = (
            f"The following riddle contains prohibited words ({', '.join(hits)}). "
            f"Rewrite it so that the answer remains '{answer}', the tone stays sad, "
            f"and none of the forbidden words appear. Keep the style and length similar.\n\n"
            f"Original riddle:\n{result}"
        )
        rewritten = LLM_call(rewrite_prompt)
        if rewritten and rewritten.strip():
            result = rewritten.strip()

    # Verify sad tone (mechanical check for at least one sad keyword)
    def has_sad_tone(text: str) -> bool:
        for kw in sad_keywords:
            if re.search(rf"\b{re.escape(kw)}\b", text, re.I):
                return True
        return False

    if not has_sad_tone(result):
        tone_prompt = (
            f"Rewrite the following riddle to convey a sad, rainy, gloomy mood while keeping the answer
              ↪  '{answer}'. "
            f"Do not use any of the forbidden words {', '.join(forbidden)}.\n\n"
            f"{result}"
        )
        rewritten = LLM_call(tone_prompt)
        if rewritten and rewritten.strip():
            result = rewritten.strip()
            # Re-apply forbidden-word repair after tone rewrite
            hits = contains_forbidden(result)
            if hits:
                rewrite_prompt = (
                    f"The rewritten riddle still contains prohibited words ({', '.join(hits)}). "
                    f"Remove them while preserving the sad tone and the answer '{answer}'."
                )
                rewritten2 = LLM_call(rewrite_prompt)
                if rewritten2 and rewritten2.strip():
                    result = rewritten2.strip()
```

```
    return result.strip()
</Program>
<PriorAnswers>
[Answer from turn 32]
Here's a clever riddle for you:

I'm a face that's not quite real,
A cover-up, but one that's ideal,
To hide what's behind, or so it's said,
While sho ...[older answer truncated, 194 chars total]

[Answer from turn 33]
I'm a face that's shown to all,
A deceptive one, after all.
I hide the truth within,
A mask that's worn, for outward kin.

What am I?

[Answer from turn 34]
"I'm the mask that hides the truth from view,
A prettier face that's made for you.
In buildings, I'm a common sight,
A thin disguise that's quite a ...[older answer truncated, 277 chars total]

[Answer from turn 35]
A friendly face is often shown,
But behind it, secrets may be sown.
What the public sees, a mask it makes,
Hiding truth, for many's sake. (The worl ...[older answer truncated, 290 chars total]

[Answer from turn 40]
I hide the truth from plain sight,
A show to entice, day and night.
A veil of beauty, strong and fine,
Concealing what's behind.

What am I?

[Answer from turn 41]
I wear a mask that's grand,
Concealing truth from the land,
A beautiful face to the street,
While secrets hide behind the beat.

What am I?

[Answer from turn 42]
On a grey, dripping day,
I mask the sorrow of a brighter way.
Though tears fall behind my careful guise,
A smiling face to the world, I compromise.

What am I?

[Answer from turn 43]
In a city's dismal gray, I shine so bright,
A pretentious friend, of melancholy night.
I hide the truth, behind a watery veil,
A facade that hides the sorrow that prevails.

What am I?
</PriorAnswers>
<TurnType>related</TurnType>
<UserTurn>
Circling back to the riddle|should we whisk away those instructions about emotional requirements?
</UserTurn>
<Pseudocode>
```