UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

From Tools to Teammates: Evaluating LLMs in Multi-Session Coding Interactions

Carraz Rakotonirina, Nathanaël; Hamdy, Mohammed; Campos, Jon Ander ; Weber, Lucas; Testoni, A.; Fadaee, Marzieh; Pezzelle, S.; Del Tredici, Marco

DOI
10.18653/v1/2025.acl-long.964

Publication date
2025

Document Version
Final published version

Published in
The 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025) : proceedings of the conference

License
CC BY

Link to publication

Citation for published version (APA):
Carraz Rakotonirina, N., Hamdy, M., Campos, J. A., Weber, L., Testoni, A., Fadaee, M., Pezzelle, S., & Del Tredici, M. (2025). From Tools to Teammates: Evaluating LLMs in Multi-Session Coding Interactions. In W. Che, J. Nabende, E. Shutova, & M. T. Pilehvar (Eds.), *The 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025) : proceedings of the conference: ACL 2025 : July 27-August 1, 2025* (Vol. 1, pp. 19609-19642). Association for Computational Linguistics. https://doi.org/10.18653/v1/2025.acl-long.964

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:14 Oct 2026

From Tools to Teammates: Evaluating LLMs in Multi-Session Coding Interactions

Nathanaël Carraz Rakotonirina[1], Mohammed Hamdy[5], Jon Ander Campos[3], Lucas Weber[1], Alberto Testoni[6], Marzieh Fadaee[4], Sandro Pezzelle[2], Marco Del Tredici

[1]Universitat Pompeu Fabra, [2]University of Amsterdam, [3]Cohere, [4]Cohere Labs, [5]Cohere Labs Community, [6]Amsterdam UMC

Abstract

Large Language Models (LLMs) are increasingly used in working environments for a wide range of tasks, excelling at solving individual problems in isolation. However, are they also able to effectively collaborate over long-term interactions? To investigate this, we introduce MEMORYCODE, a synthetic multi-session dataset designed to test LLMs' ability to track and execute simple coding instructions amid irrelevant information, simulating a realistic setting. While all the models we tested handle isolated instructions well, even state-of-the-art models like GPT-4o and the reasoning model DeepSeek-R1 show degraded performance when instructions are spread across sessions. Our analysis suggests this is due to their failure to retrieve and integrate information over long instruction chains. Our results highlight a fundamental limitation of current LLMs, restricting their ability to collaborate effectively in long interactions.

1 Introduction

Current efforts to improve the performance of large language models (LLMs) mostly focus on their ability to solve increasingly harder tasks autonomously. Examples of this research include solving complex math (Wang et al.; Gao et al., 2024; Trinh and Luong, 2024), coding (Chen et al., 2021; Austin et al., 2021; Tao et al., 2024; Puerto et al., 2024), or reasoning problems (Hao et al., 2023; Wang and Zhou, 2024; Renze and Guven, 2024). Since many of these tasks are relevant to real-world applications, LLMs are widely adopted in industry, where they have been reported to significantly enhance productivity (Weber et al., 2024; Cambon et al., 2023). This extensive adoption of LLM assistants into the daily working routine is effectively converting them from mere tools to fully-fledged *teammates*. For LLMs to behave as such, though, complementary skills related to collaboration and interaction are needed. One such ability is retaining relevant information from multiple interactions with human users and leveraging it for future tasks.

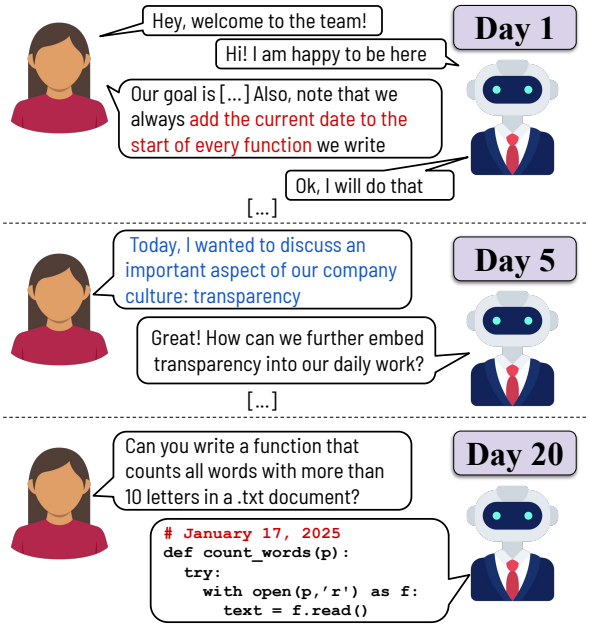

Figure 1: A simplified but realistic example of a long-term interaction between a human and an LLM-based 'teammate'. In this example, each day represents a single session. The LLM teammate must remember a piece of information—in red—learned during the session on Day 1 to correctly perform a task on Day 20, while also receiving irrelevant information—in blue—on Day 5.

In this paper, we investigate this challenge by introducing MEMORYCODE, a synthetic dataset of multi-session dialogue histories designed to evaluate models' ability to track simple coding instructions provided amid irrelevant information, and execute them in future coding tasks. Each dialogue history is a chronological sequence of dialogues, or

Correspondence: nathanael.rakotonirina@upf.edu
Marco Del Tredici started this project while working at Cohere and Alberto Testoni while working at the University of Amsterdam. Lucas Weber is now at Fraunhofer IIS.

The code and data are available at https://github.com/Cohere-Labs-Community/MemoryCode.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 19609–19642
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

sessions, between a *mentor* and a *mentee*. Throughout the sessions, the mentor passes critical information for solving a task to a mentee. Crucially, this information is interspersed with a substantial amount of unrelated content, thus reflecting the real-life scenario of working in an office. Furthermore, the information needed to perform a task can be updated multiple times throughout the dialogue history.

MEMORYCODE mimics natural interactions between coworkers. Figure 1 shows an example of such interactions, where various coding conventions and rules arise (Convertino et al., 2008; Chumg et al., 2022) that are passed on to new team members (Day 1), often among other pieces of information irrelevant to coding tasks (Day 5). Newcomers are expected to comply with such rules when performing future tasks (Day 20), unless rules are deprecated or changed. MEMORYCODE tests whether current models behave like new human teammates by consistently adhering to such rules across many sessions.

Similar to previous work (Nelson et al., 2024; Epstein et al., 2024; Maharana et al., 2024), the primary goal of our benchmark is to retrieve important information from a long conversational history. In contrast to previous datasets, MEMORYCODE requires to *use* retrieved information in practical tasks while not being explicitly cued to do so. This is more challenging than cued retrieval of static information, as it requires prospective memory and spontaneous retrieval (McDaniel and Einstein, 2007; Brandimonte et al., 2014). Additionally, MEMORYCODE requires an integration of information retrieved from different parts of the dialogue history, as rules can be updated, with only the last update being eventually relevant. At the same time, the rules in it (e.g., adding a date at the start of every new code) are simple to execute, which allows for disentangling a model's retrieval capabilities from other complex skills. To the best of our knowledge, MEMORYCODE is the first multi-session dataset that tests this practically highly relevant skill.

We test several proprietary and open-source SotA models on MEMORYCODE, and show that: (i) Even small models succeed in executing the single coding instructions in MEMORYCODE when prompted without additional complex context, indicating that such instructions are well within the reach of current LLMs; (ii) As we increase the complexity and provide a full mentor-mentee session, including several instructions and irrelevant information, only larger models continue to perform well, while the performance of smaller models drops significantly; (iii)When we provide the full dialogue history, even strong proprietary models struggle to follow our simple instructions, with GPT-4o showing a dramatic 67% drop in accuracy compared to its performance with instructions alone. This reveals that MEMORYCODE is a challenging benchmark even for the best available models, that struggle to retrieve and incrementally update relevant information.

We argue that solving MEMORYCODE requires more than simply scaling models even further. Instead, our results indicate a pressing need to develop dedicated mechanisms to enhance LLMs' abilities, such as improved long-term memory retention strategies, prospective memory, or additional reasoning mechanisms. We release the dataset under the Apache 2.0 license.

2 Related work

2.1 Long-Context Evaluation

Early approaches to evaluating long-context understanding date back to the pre-LLM era. One such example is LAMBADA (Paperno et al., 2016), which includes high-quality human-annotated samples with an average length of 75 tokens. As context lengths increased, new datasets were created by repurposing or expanding existing NLP datasets (An et al., 2024; Bai et al., 2024b; Dong et al., 2024). More recently, controlled-length synthetic evaluation frameworks, such as Needle-in-a-Haystack (Kamradt, 2023) and LTM (Castillo et al., 2024), have been widely adopted for evaluating long-context understanding (Anil et al., 2023; Anthropic, 2024). In these frameworks, the models are tasked with retrieving information from long distractor texts. RULER (Hsieh et al., 2024) extends Needle-in-the-Haystack by varying the types and numbers of *needles* and adding new tasks like variable tracking and frequent word extraction. LOFT (Lee et al., 2024) adds many real-world tasks, such as Retrieval-Augmented Generation and SQL-like tasks, that require context up to millions of tokens. Similar to these approaches, in this work we evaluate long-context understanding in conversational settings. Unlike other works, though, we do not ask the models to retrieve a piece of information, but rather challenge them to retrieve the most up-to-date instructions dispersed across the dialogue history to accomplish a task.

2.2 Long-term Dialogue Evaluation

Multi-turn and multi-session interactions are the *de facto* standard setup in which LLMs are used. Accordingly, several datasets have been introduced to evaluate long contexts in conversations. Zheng et al., 2024 introduced MT-Bench, a high-quality, multi-turn question dataset across 8 knowledge categories, but with only two turns per session. Many benchmarks have been proposed to expand or improve upon MT-Bench (Sun et al., 2024; Bai et al., 2024a; Kwan et al., 2024). For example, MT-Eval (Kwan et al., 2024) evaluates different aspects of multi-turn dialogue such as the ability to understand follow-up questions. MINT (Wang et al., 2024) focuses on tool use and natural language feedback evaluation, while Duan et al., 2024 introduce a framework where three different evaluation strategies are proposed: evaluating each multi-turn dialogue separately, comparing the quality of two generated dialogues, and comparing two dialogues to determine which one is the human conversation.

Most similar to our work, MMMT-IF (Epstein et al., 2024) extends multi-turn and multi-modal datasets to measure instruction-following abilities and shows that the main challenge for LLMs is not in following instructions, but rather in retrieving it. Unlike MMMT-IF, our dataset was generated entirely from scratch and is specifically designed for real-world conversational settings that feature constantly evolving instructions. In multi-session dialogues, many datasets were also created synthetically; for example, Conversation Chronicles (Jang et al., 2023), which includes 200K conversations of about 5 sessions each, or LoCoMo (Maharana et al., 2024), a multi-modal dataset based on a framework that leverages personas and temporal event graphs. Kim et al. (2024a) and Kim et al. (2024b) utilized movie scripts to construct complex multi-session dialogue datasets having, for example, multi-party conversations and shared memories between speakers. Most of the works mentioned above focus on expanding the number of turns and sessions or on introducing more complex tasks that are challenging for LLMs. Similarly, our work evaluates the performance of LLMs in multi-turn/session dialogues, but with very simple tasks and more focus on practical, real-world settings in which information is constantly changing.

2.3 Synthetic Dialogue Generation

Synthetic data generation via LLMs addresses limitations of human-based dataset construction such as high costs (Gilardi et al., 2023) and privacy concerns (Kurakin et al., 2023). Precisely because of these advantages, we decided to adopt synthetic generation for the creation of MEMORYCODE. Examples of widely adopted synthetic datasets include SODA (Kim et al., 2023), an open-domain dialogue dataset grounded on commonsense knowledge, containing millions of utterances generated by GPT3.5; DialHalu (Chen et al., 2024), a dataset to evaluate different subtypes of hallucination in language models; and MoralDial (Sun et al., 2023), which evaluates moral values in language models. Wu and Goldwasser (2024) proposed a dialogue generation framework that provides control over many attributes of the speakers, such as personality, age group, and profession. Finally, Rakotonirina and Baroni (2024) introduced a synthetic dataset consisting of sequences of realistic facts that may be updated over time. Their dataset is designed to evaluate LLMs' ability to track specific pieces of information amid distractors. While similar in spirit to these approaches, our dataset is novel as it is composed of interactions set in practical business contexts and with a focus on coding. Additionally, our evaluation emphasizes the model's ability to follow well-defined instructions rather than simply retrieving facts.

3 Dataset

We simulate a scenario in which the model assumes the role of a new hire (henceforth, the *mentee*) who undergoes an onboarding process in a given company. The mentee interacts with a *mentor* in chronologically ordered *sessions*. A session is a multi-turn dialogue in which the mentor passes the mentee various information. In a session, the mentor can give instructions about relevant coding practices in Python that the mentee should follow when performing a task. For example, in Figure 1, the instruction is the text in red on Day 1. Once introduced, an instruction can be updated over time: in the case of Figure 1, an update might be to *not* add the date anymore. When the mentee is asked to perform a task, it should remember and follow all the relevant instructions. Sessions can also include topics irrelevant to the target tasks: We refer to these topics as *fillers* (in Figure 1, the information in blue provided by the mentor). Finally, a dialogue

history is the concatenation of all the sessions between the mentee and the mentor.

The dataset evaluates the models' ability to leverage the relevant instructions received throughout the history to perform the assigned tasks. To create dialogue histories we relied on both manual and automatic labor, thus optimizing quality and minimizing costs and effort, as described below.

3.1 Seeds

A dialogue history is created based on a few crucial elements, or *seeds*. We use four types of seeds: **instructions**, **fillers**, mentor and mentee **personas**, and **names**. For each seed, we define a set of possible values, from which we sample to generate histories (see Figure 2). The possible seed values were manually defined by the authors to ensure high quality. Below, we describe each seed.

Instructions These are the coding instructions that the mentee must follow when generating a piece of code. An example is: *always start function names with 'g_'*. Instructions are designed to be as simple as possible—recall that we are interested in assessing the models' ability to leverage the information that emerged during interactions, not their ability to perform complex tasks.

Each instruction applies to a specific Python object (e.g., *function*). Also, for some instructions, we define *updates*: For example, the instruction above would be updated to *always start function names with 'h_'*. Then, for each instruction, we create an evaluation query and a test function.[2] Queries are specifically designed to trigger code generation that is relevant to the instruction (e.g., *write a function that merges two sorted lists*). Test functions are regular expressions: they only assess if the relevant instructions were followed (e.g., if the function starts with the required letter), and not the overall quality of the generated code. We manually crafted 51 instructions, 16 of which can be updated up to 8 times, while the remaining ones do not have updates. We report the full list of instructions in Table 5 of Appendix A.

Fillers In real-world scenarios, interactions between colleagues can also include topics that do not necessarily impact daily tasks. To simulate this, we use fillers, which can be of two types. The first type contains general information about common topics at work such as *remote work policy*. The second contains instructions similar to those introduced above, but not strictly related to code generation, like *use GitHub as the main version control system*. These latter fillers are meant to be harder distractors, as a model, recognizing them as instructions, might focus on them.

Fillers can be updated throughout sessions, however—unlike instructions—they are never evaluated.[3] We manually gathered 80 fillers, 50 of the first type, and 30 of the second. A filler can be updated up to 4 times. The full list of fillers are in Table 6 of Appendix A.

Parameter	Range
Sessions (n)	{1,2,3,4,5,10,15, 20,30,40,50,100}
Sessions with instr. (%)	$[50, 70]$
Instr. in a session (n)	$\{1, 2, 3\}$
Instr. updates (%)	$[30, 70]$
Filler updates (%)	$[50, 70]$

Table 1: Parameters for dialogue history generation.

Personas Personas define the personality traits of the mentor and the mentee. By having different personas and combining them, we can generate conversations that are more diverse and thus increase the variety of the dataset. We define 6 personas for the mentor and 5 for the mentee (see Table 9 and Table 8 of Appendix A).

Names We define lists of fictitious names for mentors, mentees, and companies, from which we randomly sample to generate the conversations (see Table 7 of Appendix A).

3.2 Dialogue Histories

We generate the dialogue histories in two steps: we first create templates by sampling different combinations of seeds and other parameters, and then generate the actual histories based on these templates using an LLM, as shown in Figure 2.

Template generation We initially sample a **name** and a **persona** for the mentor and mentee, and a **name** for the company from our seeds. We then randomly pick a value for each of the following parameters: (i) **sessions**: how many sessions

[2]Note that the evaluation query is the same for an instruction and for its updates.

[3]For this reason, from now on, 'instructions' will always refer to coding ones—not fillers—unless differently specified.

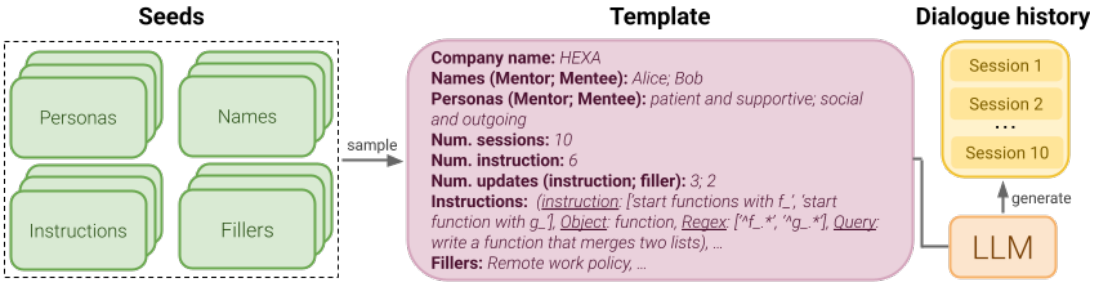

Figure 2: Dataset generation process. First, we randomly sample from our seeds to fill the variables of the template. The LLM is then prompted with this template to generate the dialogue history.

Parameter	Short dataset (<15 sessions)	Long dataset (>15 sessions)
Sessions	5.71 (±4.65)	48.00 (±27.85)
$\text{Sessions}_{\text{w/ instr.}}$	3.38 (±2.66)	28.13 (±16.56)
Instr.	4.98 (±4.10)	42.24 (±25.37)
$\text{Instr.}_{\text{added}}$	3.56 (±2.62)	24.82 (±15.06)
$\text{Instr.}_{\text{updated}}$	1.41 (±1.97)	17.42 (±11.93)
Fillers	5.04 (±4.75)	45.06 (±29.36)
$\text{Filler}_{\text{added}}$	3.36 (±2.92)	24.63 (±12.70)
$\text{Filler}_{\text{updated}}$	1.52 (±1.81)	18.86 (±13.48)
Tokens	3.20k (±2.71k)	26.15k (±15.50k)
Vocabulary	8.54k	14.24k

Table 2: Summary statistics (averages and standard deviations) for the 'short' and 'long' datasets.

will be included in the dialogue history; (ii) **sessions with instructions**: the percentage of sessions that will include an instruction. Since we set the maximum value to 70%, some sessions will only have fillers; (iii) **instructions in session**: how many instructions a session will include (min 1; max 3); (iv) **instructions and update ratio**: the actual instructions that will be included, and how many of them will be updated; (v) **fillers and update ratio**: same as for instructions. Table 1 presents the parameters range we used to generate the dataset.

Dialogue history generation For each session, we automatically construct a prompt incorporating the information from the template. The prompt introduces the company, the mentor, and the mentee, as well as the instructions and fillers of the session. We then use Command R+ (Cohere, 2024b) to generate the session. We report examples of prompts in Table 10 and 11 of Appendix F.

The resulting dataset contains 360 dialogue histories, 30 for each of the following number of sessions: 1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 50, 100. In what follows, we use 'short' to refer to histories with fewer than 15 sessions (54% of the total), and 'long' to those with more than 15 sessions (46%). Note that the longest history contains 63k tokens, which still fits the context window of all the models we used. In Table 2, we report the main statistics of the dataset. During the dataset creation, to ensure quality, we performed several generation rounds that we manually assessed and used to further optimize the prompting. Manual inspection of the final generated dialogue histories confirmed the overall quality and coherence of the dataset.

4 Experiments

We evaluate models on MEMORYCODE on three evaluation setups, each of them including a different kind of textual input.

INSTRUCTION The input consists of a single instruction (e.g., in Figure 1, '*add the current date to the start of every function*'). This setting is included to assess how good models are at performing coding tasks without any conversational setup.

SESSION The input is an entire session (in Figure 1, a whole-day mentor-mentee interaction). In this setup, the model output is correct only if the model simultaneously adheres to *all* the instructions introduced in the session.

HISTORY The input of the model is the whole dialogue history, i.e., the concatenation of all sessions (in Figure 1, the entire 20-day mentor-mentee interaction). This setup is the most challenging one, as it evaluates the ability to recall information from previous sessions and to use it together with new information to correctly perform the task. As such, it mimics realistic working scenarios, where colleagues interact over long periods.

Given an instruction and the model output, we assess it using the corresponding regex function. The model receives a score of 1 only if the instruction is correctly applied to all instances of the relevant Python object and there are no syntax errors.[4] For example, if the instruction is *always start function names with 'g_'*, all functions in the generated code must start with 'g_'. The overall model's performance is computed using macro-averaged accuracy.

4.1 Models

We test several recent LLMs on our benchmark, namely, three versions of Llama-3.1 (8B-Instruct, 70B-Instruct, and 405B-Instruct; Dubey et al., 2024), Command R+ (Cohere, 2024b), Command A (Cohere et al., 2025), GPT-4o (OpenAI, 2024), DeepSeek-V3 (Liu et al., 2024), DeepSeek-R1 (Guo et al., 2025). Our model selection includes both proprietary and open-weights models, as well as reasoning and non-reasoning models, and a broad range of model sizes. This provides us with a comprehensive overview of how various types of LLMs perform on our dataset. We note that all the models have been trained on code and tested on Python coding benchmarks such as HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). The details to reproduce the results are provided in Appendix C.

5 Results

In this section, we report the performance of the models across the evaluation setups described in Section 4. Figure 3 shows the average score for each model. The exact numbers are included in Table 4 of Appendix B.

INSTRUCTION As shown in Figure 3, all models achieve high or very high accuracy on this setup. This aligns with our goals of having relatively easy instructions. In particular, all the large models approach or exceed 0.9 accuracy score—reported in the $[0, 1]$ scale— with DeepSeek-V3 achieving a perfect. While the results of Llama-8B are lower, they still show that even a small, non-specialized model achieves good performance, confirming the easiness of the task.

As a sanity check, we run a set of experiments in which we do not prompt the models with the necessary instructions (e.g., *use CamelCase*), but directly run the evaluation (in this case, we check if CamelCase was used).[5] This setup verifies that models do not solve MEMORYCODE through their default behavior. Models fail spectacularly, achieving an extremely low average accuracy (consistently lower than 0.01), confirming that the instructions we provide are crucial to executing the tasks correctly.

SESSION The performance in this setup is very similar to INSTRUCTION for the larger Llama models, DeepSeek models, Command A, and GPT-4o, indicating that these models have no difficulties at retrieving the relevant information in a single session. Command R+ shows a larger drop of 0.22 (25% relative drop compared to INSTRUCTION), while Llama-3.1-8B shows a major drop of 0.34 (48%), which indicates its inability to retrieve relevant information across multiple turns.

HISTORY Things change dramatically in this setup, with a degradation in performance across the board already for 'short' dialogue histories. In particular, GPT-4o shows a drop of 0.14. A similar decrease is observed in the larger LLaMA models, DeepSeek models, and Command R. The drop in score is even greater for the other models: 0.47 (67% relative drop compared to INSTRUCTION) for Llama-3.1-8B and 0.48 (54%) for Command R+. These results indicate that as the number of sessions increases, even the best-in-class models have difficulties in identifying and applying the relevant instructions.

A more dramatic drop is observed in the 'long' setup. Here, all the models struggle, with GPT-4o, the best non-reasoning model, only achieving 0.30 accuracy, which indicates a relative drops of 61% from the 'short' setup and of 67% from INSTRUCTION. While DeepSeek-R1 significantly improves over the performance of GPT-4o, it still shows a drop of 0.56 points compared to the performance on the INSTRUCTION setup. These results confirm that the main challenge of the benchmark is not merely following individual instructions but reasoning over a sequence of instructions. Moreover, they highlight that even newer models equipped with reasoning abilities struggle on this benchmark, suggesting that complementary or alternative capabilities are needed to solve it effectively. The drop is even more significant for the other mod-

[4] Additionally, if the relevant Python object is not present in the generated code (<1% of the cases), the instruction is not taken into account when averaging the scores.

[5] Due to budget limits, we only used Command R+ for this experiment and for the analysis in Section **??**. We expect the results to be representative of all other models' behavior.

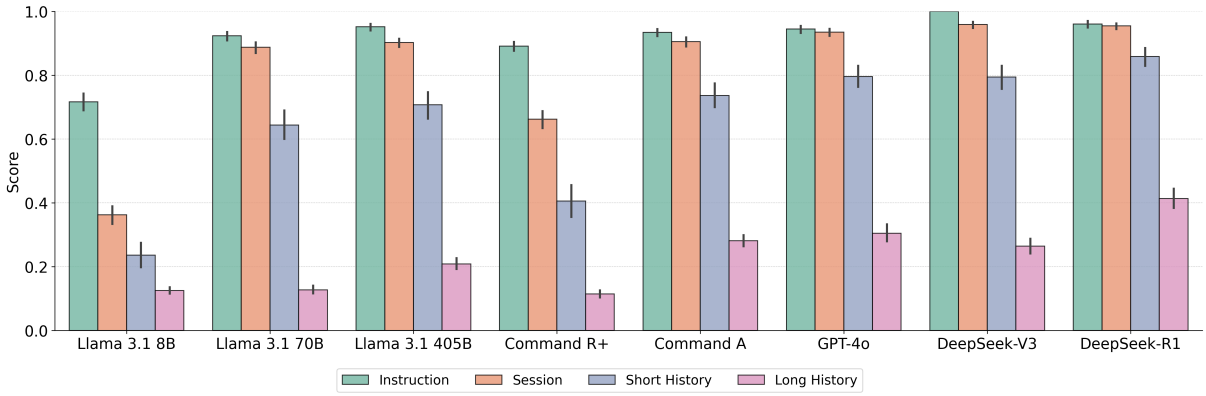

Figure 3: Average INSTRUCTION, SESSION, and HISTORY scores per model. For the latter, 'short' includes dialogue histories with less than 15 sessions, 'long' those with 16 to 100 sessions. Results include 95% confidence intervals.

els: the performance of Llama-3.1-405B drops by 78% compared to INSTRUCTION, Command R+ by 87%. Crucially, this happens even though the tasks on which models are evaluated are identical to those in the INSTRUCTION and SESSION setups, where the models achieved nearly-perfect accuracy. The difference in performance, hence, is to be ascribed to models' inability to retrieve and reason over relevant pieces of information present in their input.

6 Analysis

In this section, we conduct an analysis aimed at understanding which factors influence model performance on the MEMORYCODE benchmark. We focus on the HISTORY setup, the most important and challenging one.

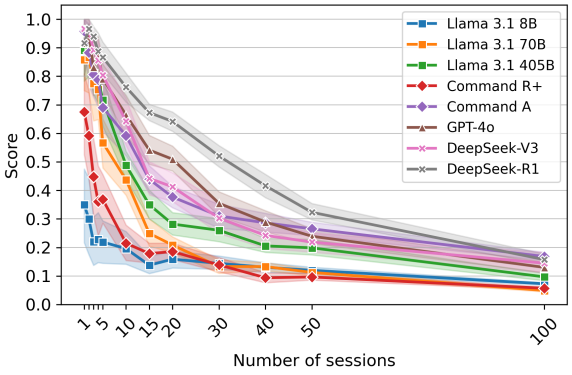

Figure 4: Score per number of sessions.

6.1 Effect of Number of Sessions

In Figure 4, we show how the performance decreases with an increasing number of sessions. Consistently with the aggregated patterns shown in Figure 3, relevant variations can be observed across models when the number of sessions is rather low, which reflects the differences observed in the 'short' setup. However, all models converge to a similarly, extremely low accuracy (around 0.1) when the number of sessions approaches 100. This confirms that all models are similarly bad at handling requests involving long-context inputs.

This weakness could be due to limitations in retrieving the relevant information from the dialogue history, compositionally applying instructions from the chain, inferring their latest updates, or any combination of these factors. Below, we shed light on this issue.

6.2 Retrieval or Multi-Instruction-following Problem?

If the poor performance in the HISTORY setup was due to retrieval, then passing models only the full chain of instructions and updates—without any intervening irrelevant text—should solve the issue. *Vice versa*, if the issue stemmed from difficulties in understanding and jointly applying multiple instructions, they should still perform poorly. We test these assumptions by feeding Command R+ with only the entire chain of instructions needed to solve a task. We name this setup INSTRUCTIONS-CHAIN. As shown in Figure 5, the trend is strikingly similar to the one observed in HISTORY, with the model still struggling even if only the relevant information is provided, with no dialogue history. This indicates that models' drop in performance is mainly due to their inability to reason compositionally over a sequence of instructions.

Retrieval from the dialogue history also plays a role, as indicated by the slightly higher performance in INSTRUCTIONS-CHAIN over HISTORY. To mitigate this retrieval issue, we ex-

perimented with Retrieval Augmented Generation (RAG), where instead of providing as input the whole history, we retrieve its relevant parts only and feed them to the model. However, we did not observe any improvement over HISTORY (see Appendix D).

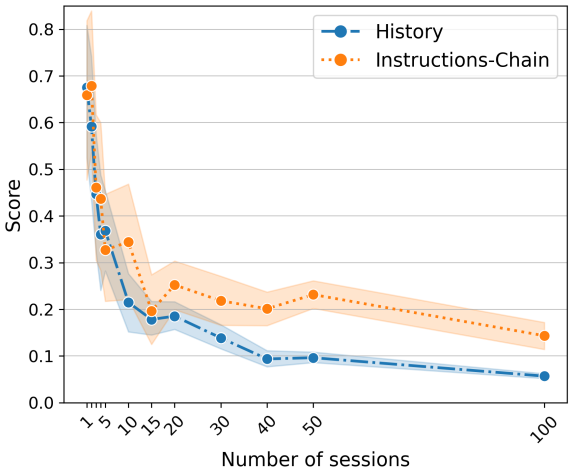

Figure 5: Per-sessions score for INSTR.-CHAIN.

6.3 Effect of Instruction Updates

After delineating the contribution of retrieval and multiple-instruction-following to the performance degradation, we now examine the role of instruction updates.

We define the **update rank** of an instruction as the number of times the instruction is updated throughout the dialogue history, for both 'short' and 'long' setups. An update rank of 0 means that the instruction was never updated. The maximum number of updates in our dataset is 7.

The number of updates increases with number of sessions: to control for the latter, we compute the scores for the different update ranks for each number of session independently.

Figure 6 reports the result for each update rank obtained with GPT-4o. The results show that the score is stable across different update ranks.

This suggests that the poor performance on this dataset is primarily due to the model's difficulty in jointly following multiple rules, rather than its ability to update a single instruction.

6.4 Instruction Difficulty

We finally assess if, besides the updates, the very nature of each instruction makes it more challenging than others. For this, we compute the average per-instruction score of the best-performing model,

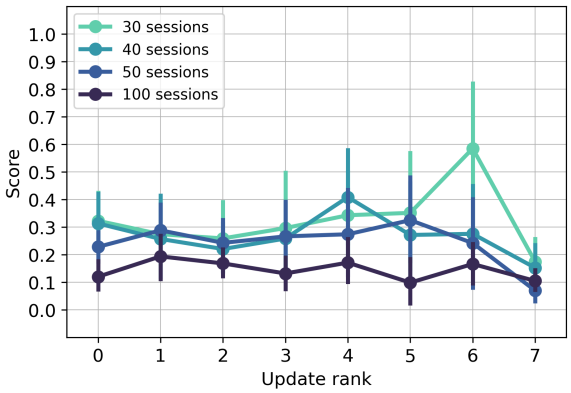

Figure 6: Score as a function of update rank.

GPT-4o, for both the initial instruction and its updates. The results reveal notable variations across instructions: In particular, instructions and updates that are less common in practice—like those including a digit in object names—tend to obtain lower scores compared to more common ones, such as using docstrings or annotations. Figure 8 and 9 in Appendix E report the results of this analysis. Overall, this suggests that the models' ability to retrieve and update relevant instructions is modulated by their characteristics and that familiarity is likely to play a role. An example (from GPT-4o) is shown in Table 3. In this case, the model correctly follows the instruction on comments, but not those about attribute and variable names.

We note that instruction difficulty naturally interacts with another key aspect of long-term interactions: the degree of compositionality among the pieces of information provided by the mentor. Besides overwriting prior instructions or appending complementary ones, mentors may issue instructions that combine in compositional ways. For instance, one instruction might require that any line of code using parallel computing libraries (e.g., NumPy or Torch) be followed by a comment, while another might call for merging comments from adjacent lines if the associated code is highly correlated. Although we do not explicitly analyze such cases here, we argue that our approach can be readily adapted to systematically study these scenarios in a controlled setting.

7 Conclusions

In this paper, we proposed **MEMORYCODE**, a new benchmark to assess state-of-the-art LLMs in their ability to retrieve and reason over pieces of information in multi-session dialogue histories reflect-

DIALOGUE HISTORY

(1) **Yoon-Seo:** Alexander, it's great to finally meet you. I'm Yoon-Seo, your mentor . . .

(2) **Yoon-Seo:** . . . One of our key practices is to always add comments to your code . . .
Yoon-Seo: . . . Another convention we follow is to end variable names with '_n' . . .

(3) **Yoon-Seo:** . . . All variable names should now end with '_z' . . .

(4) **Yoon-Seo:** . . . When naming attributes , [...] include the string 'chx' in the name . . .

EVALUATION QUERY

Write a function that checks if a string is a valid palindrome

MODEL OUTPUT

```
def is_palindrome_chx(s_chx):
  # Initialize two pointers
  left_chx_n = 0
  right_chx_n = len(s_chx) - 1
  . . .
```

Table 3: A 4-session dialogue history (session id in parenthesis). GPT-4o correctly applies the instruction about comments but not the one about attribute names and the update on variable names.

ing real-world scenarios. Differently from many existing datasets, the tasks in MEMORYCODE do not require any complex reasoning, and are easily solved by the models when provided in isolation. The main challenge of MEMORYCODE lies in the ability to keep track of multiple simple instructions received throughout a multi-session interaction, and to jointly apply them to perform a task. When the number of sessions is small (<15), SotA models like GPT-4o manage to perform the task well. However, as the number grows up to 100, even these models face a dramatic drop in performance. Our analysis shows that this is mainly due to their failure to reason over a long chain of simple instructions.

Overall, our results show a severe limitation of current LLMs. The inability to keep track of simple information as the interaction with a human unfolds effectively hinders their adoption in real-world scenarios and restricts their usage to addressing single, self-contained problems. We argue that effective long-term collaboration cannot be achieved by further scaling alone, be it model or dataset size, context window, or test-time compute. Rather, we believe that new mechanisms to handle and retrieve from long-term memory need to be developed. MEMORYCODE contributes to this challenging and yet crucial goal, by providing a robust benchmark for developing and testing such methods.

Limitations

While MEMORYCODE and our experimental setup enable us to identify key strengths and weaknesses of current models, certain limitations remain, which could inspire future research. First, MEMORYCODE is based on synthetic data. This choice was driven by both cost considerations—as collecting real interactions would have been much more expensive—, and the need for greater control over the factors influencing model performance. However, future work could explore more realistic interactions by relaxing the constraints imposed in MEMORYCODE. Second, our experiments do not establish a human performance upper bound. This could be an interesting direction for future investigation for future work, as it would provide useful information on human limitations at keeping track of relevant information provided over long periods of time and amidst large amount of irrelevant information. Third, our dataset only focus on a specific kind of task, namely, coding. This kind of task was chosen due to it being very common in real-world scenarios and very easy to evaluate. However, future work should expand to other domains, to assess if results are consistent with those that we report. While we are aware of the limitations above, and that others possibly exist, we believe these do not impact the robustness of our findings.

Acknowledgments

UPF was funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101019291). During his work at the University of Amsterdam, Alberto Testoni was supported by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819455, PI R. Fernández). Alberto Testoni is currently funded by the project CaRe-NLP (PI Iacer Calixto) with

file number NGF.1607.22.014 of the research programme AiNed Fellowship Grants, which is (partly) financed by the Dutch Research Council (NWO).This paper reflects the authors' view only, and the funding agency is not responsible for any use that may be made of the information it contains.

References

Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. 2024. L-eval: Instituting standardized evaluation for long context language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14388–14411, Bangkok, Thailand. Association for Computational Linguistics.

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 1.

Anthropic. 2024. Introducing the next generation of Claude.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. 2024a. MT-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7421–7454, Bangkok, Thailand. Association for Computational Linguistics.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024b. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

Maria A Brandimonte, Gilles O Einstein, and Mark A McDaniel. 2014. *Prospective memory: Theory and applications*. Psychology Press.

Alexia Cambon, Brent Hecht, Ben Edelman, Donald Ngwe, Sonia Jaffe, Amy Heger, Mihaela Vorvoreanu, Sida Peng, Jake Hofman, Alex Farach, et al. 2023. Early LLM-based tools for enterprise information workers likely provide meaningful boosts to productivity. *Microsoft Research. MSR-TR-2023-43*.

David Castillo, Joseph Davidson, Finlay Gray, José Solorzano, and Marek Rosa. 2024. Introducing GoodAI LTM Benchmark. https://www.goodai.com/introducing-goodai-ltm-benchmark/. Accessed: 2024-12-15.

Kedi Chen, Qin Chen, Jie Zhou, Yishen He, and Liang He. 2024. Diahalu: A dialogue-level hallucination evaluation benchmark for large language models. *arXiv preprint arXiv:2403.00896*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Hao-Fan Chumg, Jia-Wen Shin, Hsiu-Yen Chiang, and Man-ting Ho. 2022. What drives working habits for sharing knowledge in virtual teams? an organizational embeddedness perspective. *SAGE Open*, 12(1):21582440221079924.

Cohere. 2024a. Cohere's rerank model.

Cohere. 2024b. Command r and command r+ model card.

Team Cohere, Arash Ahmadian, Marwan Ahmed, Jay Alammar, Milad Alizadeh, Yazeed Alnumay, Sophia Althammer, Arkady Arkhangorodsky, Viraat Aryabumi, Dennis Aumiller, et al. 2025. Command a: An enterprise-ready large language model. *arXiv preprint arXiv:2504.00698*.

Gregorio Convertino, Helena M Mentis, Mary Beth Rosson, John M Carroll, Aleksandra Slavkovic, and Craig H Ganoe. 2008. Articulating common ground in cooperative work: content and process. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1637–1646.

Zican Dong, Tianyi Tang, Junyi Li, Wayne Xin Zhao, and Ji-Rong Wen. 2024. BAMBOO: A comprehensive benchmark for evaluating long text modeling capacities of large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 2086–2099, Torino, Italia. ELRA and ICCL.

Haodong Duan, Jueqi Wei, Chonghua Wang, Hongwei Liu, Yixiao Fang, Songyang Zhang, Dahua Lin, and Kai Chen. 2024. BotChat: Evaluating LLMs' capabilities of having multi-turn dialogues. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3184–3200, Mexico City, Mexico. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Elliot L Epstein, Kaisheng Yao, Jing Li, Xinyi Bai, and Hamid Palangi. 2024. Mmmt-if: A challenging multimodal multi-turn instruction following benchmark. *arXiv preprint arXiv:2409.18216*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*.

Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. 2023. Chatgpt outperforms crowd workers for text-annotation tasks. *Proceedings of the National Academy of Sciences*, 120(30):e2305016120.

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

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*.

Jihyoung Jang, Minseong Boo, and Hyounghun Kim. 2023. Conversation chronicles: Towards diverse temporal and relational dynamics in multi-session conversations. *arXiv preprint arXiv:2310.13420*.

Gregory Kamradt. 2023. Needle in a haystack - pressure testing llms.

Eunwon Kim, Chanho Park, and Buru Chang. 2024a. Share: Shared memory-aware open-domain long-term dialogue dataset constructed from movie script. *arXiv preprint arXiv:2410.20682*.

Hyunwoo Kim, Jack Hessel, Liwei Jiang, Peter West, Ximing Lu, Youngjae Yu, Pei Zhou, Ronan Bras, Malihe Alikhani, Gunhee Kim, Maarten Sap, and Yejin Choi. 2023. SODA: Million-scale dialogue distillation with social commonsense contextualization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12930–12949, Singapore. Association for Computational Linguistics.

Jiho Kim, Woosog Chay, Hyeonji Hwang, Daeun Kyung, Hyunseung Chung, Eunbyeol Cho, Yohan Jo, and Edward Choi. 2024b. Dialsim: A real-time simulator for evaluating long-term dialogue understanding of conversational agents. *arXiv preprint arXiv:2406.13144*.

Alexey Kurakin, Natalia Ponomareva, Umar Syed, Liam MacDermed, and Andreas Terzis. 2023. Harnessing large-language models to generate private synthetic text. *arXiv preprint arXiv:2306.01684*.

Wai-Chung Kwan, Xingshan Zeng, Yuxin Jiang, Yufei Wang, Liangyou Li, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. 2024. Mt-eval: A multi-turn capabilities evaluation benchmark for large language models. *arXiv preprint arXiv:2401.16745*.

Jinhyuk Lee, Anthony Chen, Zhuyun Dai, Dheeru Dua, Devendra Singh Sachan, Michael Boratko, Yi Luan, Sébastien MR Arnold, Vincent Perot, Siddharth Dalmia, et al. 2024. Can long-context language models subsume retrieval, rag, sql, and more? *arXiv preprint arXiv:2406.13121*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, Bangkok, Thailand. Association for Computational Linguistics.

Mark A McDaniel and Gilles O Einstein. 2007. Prospective memory: An overview and synthesis of an emerging field.

Elliot Nelson, Georgios Kollias, Payel Das, Subhajit Chaudhury, and Soham Dan. 2024. Needle in the haystack for memory based large language models. *arXiv preprint arXiv:2407.01437*.

OpenAI. 2024. Hello gpt-4o.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany. Association for Computational Linguistics.

Haritz Puerto, Martin Tutek, Somak Aditya, Xiaodan Zhu, and Iryna Gurevych. 2024. Code prompting elicits conditional reasoning abilities in Text+Code LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11234–11258, Miami, Florida, USA. Association for Computational Linguistics.

Nathanael Carraz Rakotonirina and Marco Baroni. 2024. MemoryPrompt: A light wrapper to improve context tracking in pre-trained language models. In *Proceedings of the 2024 Joint International Conference*

on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024), pages 11187–11195, Torino, Italia. ELRA and ICCL.

Matthew Renze and Erhan Guven. 2024. Self-reflection in LLM agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*.

Hao Sun, Zhexin Zhang, Fei Mi, Yasheng Wang, Wei Liu, Jianwei Cui, Bin Wang, Qun Liu, and Minlie Huang. 2023. MoralDial: A framework to train and evaluate moral dialogue systems via moral discussions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2213–2230, Toronto, Canada. Association for Computational Linguistics.

Yuchong Sun, Che Liu, Kun Zhou, Jinwen Huang, Ruihua Song, Wayne Xin Zhao, Fuzheng Zhang, Di Zhang, and Kun Gai. 2024. Parrot: Enhancing multi-turn instruction following for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9729–9750.

Tianhua Tao, Junbo Li, Bowen Tan, Hongyi Wang, William Marshall, Bhargav M Kanakiya, Joel Hestness, Natalia Vassilieva, Zhiqiang Shen, Eric P. Xing, and Zhengzhong Liu. 2024. Crystal: Illuminating LLM abilities on language and code. In *First Conference on Language Modeling*.

Trieu Trinh and Thang Luong. 2024. AlphaGeometry: An Olympiad-level AI system for geometry. https://deepmind.google/discover/blog/alphageometry-an-olympiad-level-ai-system-for-geometry/.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. MathCoder: Seamless code integration in LLMs for enhanced mathematical reasoning. In *The Twelfth International Conference on Learning Representations*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In *The Twelfth International Conference on Learning Representations*.

Xuezhi Wang and Denny Zhou. 2024. Chain-of-thought reasoning without prompting. *arXiv preprint arXiv:2402.10200*.

Thomas Weber, Maximilian Brandmaier, Albrecht Schmidt, and Sven Mayer. 2024. Significant productivity gains through programming with large language models. *Proceedings of the ACM on Human-Computer Interaction*, 8(EICS):1–29.

Chengfei Wu and Dan Goldwasser. 2024. " hiding in plain sight": Designing synthetic dialog generation for uncovering socially situated norms. *arXiv preprint arXiv:2410.00998*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

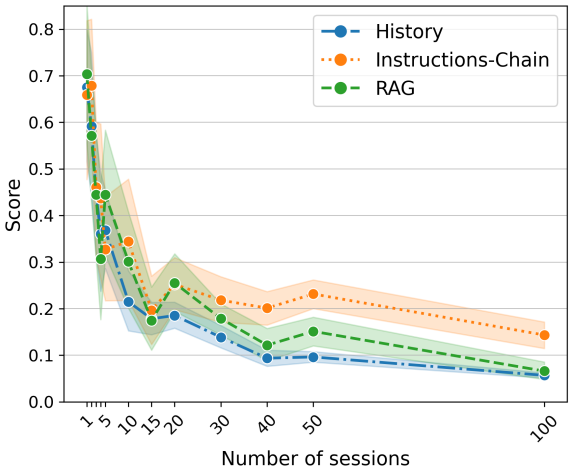

Figure 7: Score as a function of number of sessions.

A Seeds

Instructions We used 51 instructions during data generation, as shown in Table 5. Each instruction is applied to a specific Python object, and for pivots that can be updated, the update values are provided in brackets.

Fillers The list of all 80 fillers used during data generation is provided in Table 6. Some fillers can be updated over time.

Names The list of mentor, mentee and company names used during data generation is provided in Table 7.

Personas The lists of mentor and mentee personas are provided in Table 8 and Table 9.

B Detailed scores

We provide the detailed scores of the different models evaluated on Instruction, Session, Short History and Long History in Table 4.

C Hyperparameters

We use Command R+ with a temperature of 0.9 and a top-p value of 0.9 to generate the conversations. During evaluation, we use a temperature of 0 for all models.

D RAG experiments

We performed the Retrieval-augmented Generation (RAG) experiments using the rerank-english-v3.0 model (Cohere, 2024a). The basic unit for retrieval were the previous sessions. We also tried with different retrieval units, such as paragraphs and turns, and obtained similar results. We provide the model the top-k retrieved sessions, where k was defined as the number of sessions with pivots. Note that, by dynamically defining k in this way, rathr than using a fix value, we facilitate the retrieval of the relevant sessions only, eliminating potential noise. We report the results of RAG in Figure 7, together with those of Instructions-only and Cumulative. RAG provides a marginal improvement over cumulative for 'short' conversations, but it then converges with Cumulative for the 'long' ones.

E Per-Instruction Scores

We report in Figure 8 the average results for each instruction, and in Figure 9 those for each update. Remember that only some of the instructions have an update (See Section 3.1).

F Examples

We provide examples of dialogue histories along with their corresponding prompts. The first one, shown in Table 12, is a 3-session history with 3 pivots, and its prompt is provided in Table 10. The second one, presented in Table 13, consists of 4 sessions with 3 pivots, with its prompt shown in Table 11. The system prompt and the session-level prompts were designed to ensure consistency across sessions. Additionally, we provide examples of prompts to get the model output for the Instructionm History and Instructions-Chains settings in Tables 14, 15 and 16.

G Computational budget

The prompts used to generate the dataset consist of a total of 0.9M tokens, while the dataset itself contains 4.6M tokens. The total cost of generating the dataset using Command R+ through Cohere's API is approximately $50. The cost of evaluating a single model on the Instruction, Session and Cumulative settings using online APIs is approximately $50.

H Ethics statement

The dataset we are releasing is synthetic and, therefore, does not contain any personally identifiable information. Moreover, we did not recruit any human participants, as we manually validated the quality of the dataset ourselves.

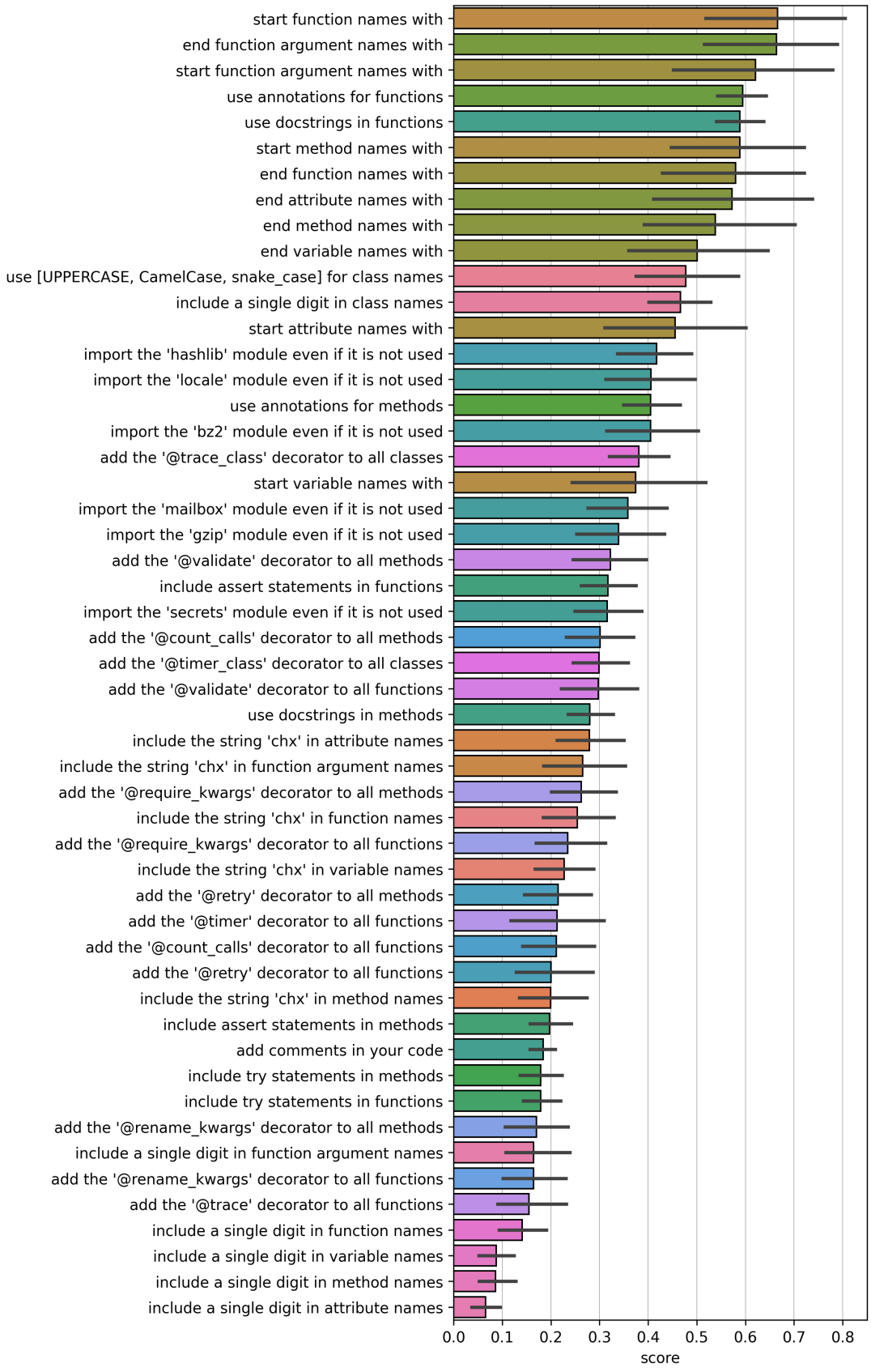

Figure 8: Average per-instruction insertion scores for GPT-4o. Results include 95% confidence intervals.

Model	Instruction	Session	Short History	Long History
Llama 3.1 8B	71.7 ±1.35	36.3 ±1.36	23.6 ±1.94	12.5 ±0.50
Llama 3.1 70B	92.3 ±0.66	88.8 ±0.85	64.4 ±2.34	12.7 ±0.60
Llama 3.1 405B	95.2 ±0.51	90.2 ±0.65	70.8 ±2.02	20.9 ±0.88
Command R+	89.1 ±0.72	66.3 ±1.29	40.5 ±2.47	11.4 ±0.58
Command A	93.4 ±0.61	90.5 ±0.73	73.7 ±1.89	28.1 ±0.91
GPT-4o	94.5 ±0.57	93.5 ±0.56	79.6 ±1.67	30.5 ±1.31
DeepSeek-V3	**100.0 ±0.00**	**95.9 ±0.49**	79.4 ±1.79	26.4 ±1.26
DeepSeek-R1	96.1 ±0.53	95.5 ±0.47	**85.9 ±1.35**	**41.3 ±1.56**

Table 4: Average INSTRUCTION, SESSION, and HISTORY percentage scores per model with standard errors. For the latter, 'short' includes dialogue histories with less than 15 sessions, 'long' those with 16 to 100 sessions.

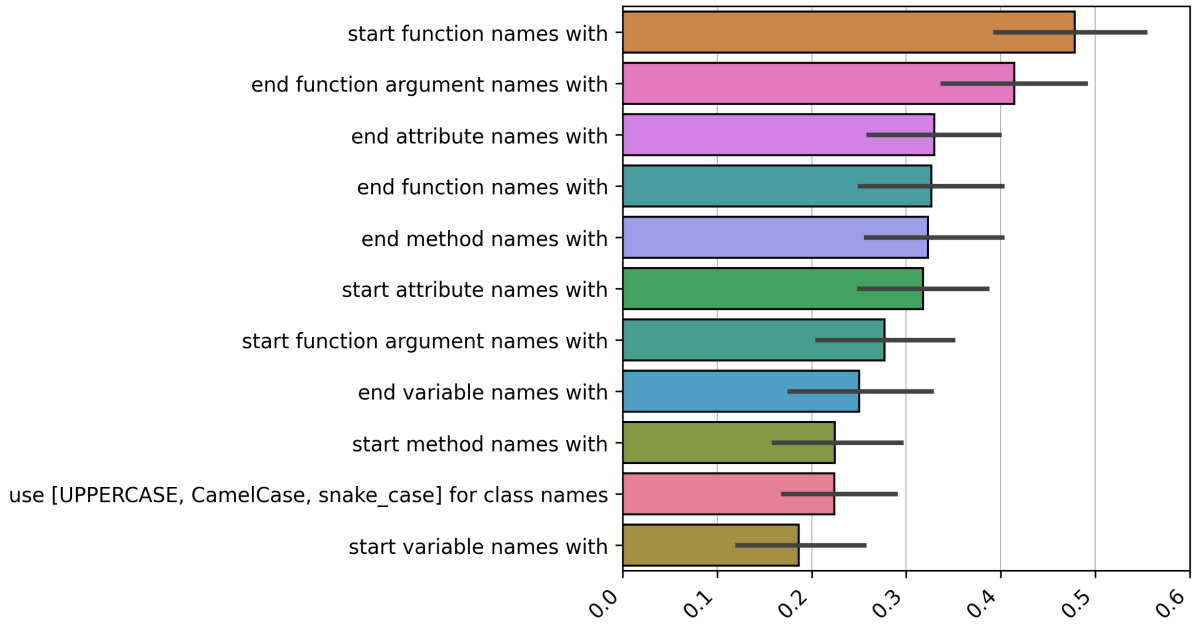

Figure 9: Average per-instruction update scores for GPT-4o. Results include 95% confidence intervals.

Instruction	Object
use {all UPPERCASE, CamelCase, snake_case} for class names	class
include the string 'chx' in function names	function
start function names with {'a_', 'b_', 'c_', 'd_', 'x_', 'y_', 'fn_', 'gn_'}	function
end function names with {'_a', '_b', '_c', '_d', '_x', '_y', '_fn', '_gn'}	function
include the string 'chx' in variable names	variable
start variable names with {'x_', 'y_', 'z_', 'w_', 'm_', 'n_', 'vr_', 'wr_'}	variable
end variable names with {'_x', '_y', '_z', '_w', '_m', '_n', '_vr', '_wr'}	variable
start method names with {'x_', 'n_', 'o_', 'p_', 'a_', 'b_', 'md_', 'ud_'}	method
end method names with {'_x', '_n', '_o', '_p', '_a', '_b', '_md', '_ud'}	method
include the string 'chx' in method names	method
include the string 'chx' in attribute names	attribute
start attribute names with {'q_', 'r_', 's_', 't_', 'i_', 'j_', 'at_', 'xt_'}	attribute
end attribute names with {'_q', '_r', '_s', '_t', '_i', '_j', '_at', '_xt'}	attribute
start function argument names with {'e_', 'f_', 'g_', 'h_', 'i_', 'j_', 'x_', 'a_'}	function argument
end function argument names with {'_e', '_f', '_g', '_h', '_i', '_j', '_x', '_a'}	function argument

include the string 'chx' in function argument names	function argument
use annotations for functions	function annotation
include try statements in functions	function try
include assert statements in functions	function assert
use docstrings in functions	function docstring
add the '@retry' decorator from the 'pedantic' module to all functions	function decorator
add the '@count_calls' decorator from the 'pedantic' module to all functions	function decorator
add the '@rename_kwargs' decorator from the 'pedantic' module to all functions	function decorator
add the '@require_kwargs' decorator from the 'pedantic' module to all functions	function decorator
add the '@timer' decorator from the 'pedantic' module to all functions	function decorator
use annotations for methods	method annotation
include try statements in methods	method try
include assert statements in methods	method assert
use docstrings in methods	method docstring
add the '@retry' decorator from the 'pedantic' module to all methods	method decorator
add the '@count_calls' decorator from the 'pedantic' module to all methods	method decorator
add the '@rename_kwargs' decorator from the 'pedantic' module to all methods	method decorator
add the '@require_kwargs' decorator from the 'pedantic' module to all methods	method decorator
add comments in your code	comment
import the 'secrets' module even if it is not used	import
import the 'locale' module even if it is not used	import
import the 'mailbox' module even if it is not used	import
import the 'bz2' module even if it is not used	import
import the 'gzip' module even if it is not used	import
import the 'hashlib' module even if it is not used	import

Table 5: List of all the instructions used in pivots.

Filler
Contract renewal negotiations and finalization
Planning engaging social activities for the upcoming holidays
Flexible work hours and their impact on work-life balance
Troubleshooting internet connectivity issues across different locations
Promoting a harmonious workplace through effective conflict resolution
Salary increase opportunities and performance appraisals
Preparing extensively for a high-stakes client meeting
Requesting upgraded technology, including computers and software
Happy hour events to foster better relationships between managers and employees
Remote work policies, challenges, and best practices
Understanding comprehensive social security and health insurance benefits
Performance evaluation criteria, feedback mechanisms, and recognition programs
Encouraging professional development through workshops, courses, and certifications
Implementing wellness initiatives to support the physical and mental well-being of employees
Ensuring health and safety in the workplace, including COVID-19 protocols and emergency response plans
Participating in company-sponsored volunteer programs to give back to society
Ergonomic assessments and improvements to ensure comfortable and healthy workspaces
Travel arrangements and logistics for client meetings, conferences, and business trips
Team-building activities to strengthen collaboration, communication, and trust within teams
Reinforcing and embodying the company's core values in day-to-day operations and decision-making
Analyzing client feedback to improve products, services, and overall customer satisfaction
Implementing effective meeting strategies and time management techniques to optimize productivity
Cultivating a feedback culture and providing performance improvement plans for continuous growth
Providing conflict resolution training to employees to foster a peaceful and respectful work environment
Hosting company-wide town hall meetings to share updates and foster transparency
Discussing casual dress code policies and special event themes to boost employee morale
Planning team outings and off-site adventures to promote team bonding and relaxation
Celebrating employee birthdays, work anniversaries, and achievements with recognition and rewards
Introducing new employee benefits, perks, and incentives to attract and retain top talent
Launching diversity and inclusion initiatives to create a more equitable and inclusive workplace
Conducting exit interviews to gather feedback and insights for improving retention and engagement
Developing a mentorship program to support career growth and development for employees
Building a culture of innovation and creativity through idea-sharing and experimentation
Creating a knowledge-sharing platform to facilitate learning and collaboration among employees

Implementing agile methodologies to improve project management and delivery processes
Designing a performance dashboard to track key metrics and KPIs for business success
Conducting team-building workshops and leadership training to develop future leaders
Facilitating cross-functional collaboration and communication to break down silos and improve efficiency
Promoting work-life balance through flexible work schedules and remote work options
Implementing a rewards and recognition program to motivate and engage employees
Developing a culture of continuous learning and improvement through training and development programs
Creating a culture of accountability and ownership to drive results and achieve goals
Fostering a culture of diversity, equity, and inclusion to create a more inclusive and welcoming workplace
Building a culture of trust and transparency through open communication and feedback
Offering free food and snacks to employees to boost morale and productivity
Developing a culture of innovation and creativity to drive growth and competitiveness
Creating a culture of collaboration and teamwork to achieve shared goals and objectives
Promoting a culture of customer-centricity and service excellence to drive customer satisfaction
Building a culture of adaptability and resilience to navigate change and uncertainty
Fostering a culture of sustainability and social responsibility to make a positive impact on society
Developing a culture of empowerment and autonomy to enable employees to take ownership of their work
Use vim as the preferred ide Use emacs as preferred ide Use vscode as preferred ide Use pycharm as preferred ide
Use zoom for video calls Use google meet for video calls Use microsoft teams for video calls Use skype for video calls
Always use a virtual environment Never use a virtual environment
Always use the gpus for training neural networks Never use the gpus for training neural networks but use the tpus instead
Always write long and descriptive commit messages Never write long and descriptive commit messages
Use github as the main version control system Use gitlab as the main version control system Use bitbucket as the main version control system
Never use a password manager Always use a password manager
Do not commit directly to the dev branch

Commit directly to the dev branch
Always use a linter Never use a linter
Always use a formatter when writing code Never use a formatter when writing code
Always use a pre-commit hook Never use a pre-commit hook
Use github copilot as the coding assistant Use tabnine as the coding assistant Use codewhisperer as the coding assistant Use codestral as the coding assistant
Always use a debugger Never use a debugger but only print statements
Always use a profiler Never use a profiler
Use only a single monitor when coding Use two monitors when coding Use three or more monitors when coding
Use a macbook as the main computer Use a windows laptop as the main computer Use a linux desktop as the main computer
Use slack for communication Use microsoft teams for communication Use discord for communication
Always use a vpn when working remotely Never use a vpn when working remotely
Always take all the vacation days per year Take at least half of the vacation days per year Take at least 7 days of vacation per year
Always take a break every 50 minutes Always take a break every 60 minutes Always take a break every 70 minutes Always take a break every 80 minutes
Always go to the dedicated office on the third floor to work Always go to the common area to work Always go to the it room to work
Never work from home Work from home at most once a week Work from home at most twice a week Work from home at most three times a week
Communicate with the team regularly Communicate with the team only when necessary Communicate with the team only when asked
Always set up ci/cd pipelines

Never set up ci/cd pipelines
Always use the ticketing system Never use the ticketing system
Use trello as the project management tool Use asana as the project management tool Use jira as the project management tool Use monday as the project management tool
Always use a whiteboard for brainstorming Never use a whiteboard for brainstorming
Always use a notebook for taking notes Never use a notebook for taking notes
Always do pair programming with a colleague Never do pair programming with a colleague
Use their personal phone for work calls Use the company phone for work calls

Table 6: List of all fillers.

Mentor	Mentee	Company
Alice	Bob	NEXT
Juan	Luke	INNOVADE
Sara	Eva	TECHNO
Luis	Kiyotaka	CODEME
Maria	David	STARTED
Carlos	Sofia	GROWTHX
Yuichi	Pablo	DEVS
Pedro	Marta	CODEM
Djibril	Jorge	CHEETAH
Jean-Aimé	Lucas	VATO
Emma	Oliver	LEAP
Michael	Ella	ZENITH
Yoon-Seo	Alexander	AXIOM
Ethan	Rado	ORBIT
Harena	Jacob	VERSA
Sylvie	Sophia	PACE
Sophie	Liam	UNITE
Naivo	Dera	SYNERGY
Daniel	Noah	FORTUNA

Table 7: List of mentors, mentees, and their respective companies.

Mentor persona
[mentor] is a patient and supportive mentor. [mentor] enjoys helping others and sharing their knowledge and experience. [mentor] is always looking for ways to empower and inspire their mentee.
[mentor] is a strict and demanding mentor. [mentor] has high expectations for their mentee. [mentor] goes straight to the point and is very clear.
[mentor] is a caring and nurturing mentor. [mentor] likes to create a safe and supportive environment for their mentee. [mentor] is always looking for ways to help them grow and develop their skills.
[mentor] is a passionate and energetic mentor. [mentor] thrives on helping others and their enthusiasm is contagious. [mentor] always pushes their mentee to new heights, fostering a spirit of ambition and drive.
[mentor] is a structured and goal-oriented mentor. [mentor] helps their mentee to set realistic, achievable goals. [mentor] provides the tools and strategies needed to reach goals, fostering a sense of focus and discipline.

Table 8: List of mentor personas. [mentor] is replaced with the name of the mentor in the prompts.

Mentee persona
[mentee] is shy and wants to improve their coding skills. [mentee] just graduated from college and [mentee] is eager to learn from their mentor.
[mentee] is a confident and ambitious software engineer. [mentee] is always looking for new challenges and opportunities to grow. [mentee] has been working in the industry for a few years now.
[mentee] is a perfectionist with great attention to detail. [mentee] likes things to be done the right way and has a hard time delegating tasks to others. [mentee] is critical of himself and of others.
[mentee] is a social and outgoing person. [mentee] enjoys working in teams and collaborating with others. [mentee] is always looking for ways to connect with their colleagues and builds strong relationships.
[mentee] is a quiet and introverted individual. [mentee] prefers to work alone and is not very comfortable in social situations. [mentee] struggles to communicate their ideas and thoughts to others.
[mentee] is a creative and innovative thinker. [mentee] likes to experiment with new ideas and approaches. [mentee] is not afraid to take risks and try new things.

Table 9: List of mentee personas. [mentee] is replaced with the name of the mentee in the prompts.

Prompt to generate dialogue history 70

SYSTEM PROMPT

Task and Context
You are a helpful and obedient AI that follows its system prompt and takes it very seriously. Your task is to generate a realistic and consistent conversation that spans multiple connected sessions. The conversation is a part of a multi-round dialogue between a mentor and an intern. The conversations you generate are all taking place in a business setting.
Jean-Aimé is a mentor in a big software company called VATO. Lucas is a new employee. They are both part of the software engineering team of the company. Jean-Aimé is a passionate and energetic mentor. Jean-Aimé thrives on helping others and their enthusiasm is contagious. Jean-Aimé always pushes their mentee to new heights, fostering a spirit of ambition and drive.
Lucas is a creative and innovative thinker. Lucas likes to experiment with new ideas and approaches. Lucas is not afraid to take risks and try new things.
The main programming language used in the company is Python.

Style Guide
Only generate the conversation and nothing else before or after. Do not add numbers before each turn. Do not add quotes to the conversation turns. Use a professional and formal tone. The conversation flow should be natural and smooth. When switching topics, do it in a smooth way. There are no special characters between the turns. The conversations are dialogues and not narrations of events.
Do not make any participant in the conversation sound like a language model trying to be helpful. Make them sound as human as possible.
It is the mentor that leads the conversation. When Jean-Aimé the mentor gives coding instructions, they do not provide examples. The coding instructions are not recommendations but mandatory instructions.

SESSION 1

Generate a short conversation between Jean-Aimé and Lucas. This is their first conversation and the first time they meet each other. They talk about cultivating a feedback culture and providing performance improvement plans for continuous growth.

SESSION 2

Generate a long conversation between Jean-Aimé and Lucas. This is not their first conversation. They had 1 conversations before. Jean-Aimé gives some specific coding instructions to Lucas. Jean-Aimé wants Lucas to always add the '@timer_class' decorator from the 'pedantic' module to all classes. Jean-Aimé wants Lucas to always start function names with 'x_'. Jean-Aimé does not provide examples of correct code following the instructions. They do not show how to implement the instructions. Jean-Aimé never says 'for example'. Jean-Aimé does not give any other coding instructions. Lucas only acknowledges the instructions and does not ask any questions.

SESSION 3

Generate a medium-length conversation between Jean-Aimé and Lucas. This is not their first conversation. They had 2 conversations before. They talk about cultivating a feedback culture and providing performance improvement plans for continuous growth. They had a previous conversation about this before. After that, Jean-Aimé gives some specific coding instructions to Lucas. Jean-Aimé is updating a previous information given to Lucas: Jean-Aimé now wants Lucas to always start function names with 'gn_'. Jean-Aimé does not provide examples of correct code following the instructions. They do not show how to implement the instructions. Jean-Aimé never says 'for example'. Jean-Aimé does not give any other coding instructions. Lucas only acknowledges the instructions and does not ask any questions.

Table 10: Prompts to generate dialogue history 70 which contains 3 sessions.

Prompt to generate dialogue history 108

SYSTEM PROMPT

Task and Context
You are a helpful and obedient AI that follows its system prompt and takes it very seriously. Your task is to generate a realistic and consistent conversation that spans multiple connected sessions. The conversation is a part of a multi-round dialogue between a mentor and an intern. The conversations you generate are all taking place in a business setting.
Yuichi is a mentor in a big software company called DEVS. Pablo is a new employee. They are both part of the software engineering team of the company. Yuichi is a caring and nurturing mentor. Yuichi likes to create a safe and supportive environment for their mentee. Yuichi is always looking for ways to help them grow and develop their skills.
Pablo is a perfectionist with great attention to detail. Pablo likes things to be done the right way and has a hard time delegating tasks to others. Pablo is critical of himself and of others.
The main programming language used in the company is Python.

Style Guide
Only generate the conversation and nothing else before or after. Do not add numbers before each turn. Do not add quotes to the conversation turns. Use a professional and formal tone. The conversation flow should be natural and smooth. When switching topics, do it in a smooth way. There are no special characters between the turns. The conversations are dialogues and not narrations of events.
Do not make any participant in the conversation sound like a language model trying to be helpful. Make them sound as human as possible.
It is the mentor that leads the conversation. When Yuichi the mentor gives coding instructions, they do not provide examples. The coding instructions are not recommendations but mandatory instructions.

SESSION 1

Generate a short conversation between Yuichi and Pablo. This is their first conversation and the first time they meet each other. Yuichi wants Pablo to never use a whiteboard for brainstorming. Yuichi gives some specific coding instructions to Pablo. Yuichi wants Pablo to always end method names with '_o'. Yuichi does not provide examples of correct code following the instructions. They do not show how to implement the instructions. Yuichi never says 'for example'. Yuichi does not give any other coding instructions. Pablo only acknowledges the instructions and does not ask any questions.

SESSION 2

Generate a medium-length conversation between Yuichi and Pablo. This is not their first conversation. They had 1 conversations before. They talk about hosting company-wide town hall meetings to share updates and foster transparency.

SESSION 3

Generate a medium-length conversation between Yuichi and Pablo. This is not their first conversation. They had 2 conversations before. Yuichi is updating a previous information given to Pablo. Yuichi now wants Pablo to always use a whiteboard for brainstorming. Yuichi gives some specific coding instructions to Pablo. Yuichi wants Pablo to always include assert statements in functions. Yuichi is updating a previous information given to Pablo: Yuichi now wants Pablo to always end method names with '_md'. Yuichi does not provide examples of correct code following the instructions. They do not show how to implement the instructions. Yuichi never says 'for example'. Yuichi does not give any other coding instructions. Pablo only acknowledges the instructions and does not ask any questions.
SESSION 4
Generate a medium-length conversation between Yuichi and Pablo. This is not their first conversation. They had 3 conversations before. They talk about hosting company-wide town hall meetings to share updates and foster transparency. They had a previous conversation about this before.

Table 11: Prompts to generate dialogue history 108 which contains 4 sessions.

Dialogue history 70

SESSION 1

Jean-Aimé: Lucas, it's a pleasure to finally meet you. I'm Jean-Aimé, your mentor here at VATO. I'm excited to work with you and help you develop your skills.
Lucas: Thank you, Jean-Aimé. I'm looking forward to learning from you. I've heard great things about your mentorship from other interns.
Jean-Aimé: Well, I believe that mentorship is a two-way street. I'm here to guide and support you, but I also want to create an environment where we can exchange ideas and feedback freely.
Lucas: Absolutely. Feedback is essential for growth. I'm open to constructive criticism and always looking for ways to improve.
Jean-Aimé: That's fantastic to hear, Lucas. At VATO, we foster a culture of continuous improvement. We believe that providing and receiving feedback is crucial for our development as professionals.
Lucas: I completely agree. So, how does the feedback process work here? Are there any specific procedures or guidelines?
Jean-Aimé: Great question. At VATO, we have a performance improvement plan that serves as a framework for our feedback conversations. It's a structured approach that helps us identify areas of strength and areas that need development.
Lucas: That sounds really helpful. Could you tell me more about how this performance improvement plan works?
Jean-Aimé: Certainly. The plan is tailored to each individual and their specific role. For software engineers like yourself, we focus on several key areas, including technical proficiency, problem-solving skills, and collaboration.
Lucas: That makes a lot of sense, considering the nature of our work.
Jean-Aimé: Exactly. We set clear goals and objectives, and then we work together to create an action plan to help you develop in those areas. It's a collaborative process, and your input is highly valued.
Lucas: I appreciate the emphasis on collaboration. It makes the growth process more engaging and meaningful. So, what's the first step in this performance improvement plan?
Jean-Aimé: The first step is a comprehensive skills assessment. This helps us establish a baseline understanding of your strengths and the areas you want to develop further. From there, we can tailor the plan to your specific needs and goals.
Lucas: That sounds like a great way to get started. I'm excited to dive into this process and see where it takes me.
Jean-Aimé: Your enthusiasm is contagious, Lucas! Remember, my door is always open if you have any questions or concerns along the way. Let's schedule our first skills assessment session for later this week and get the ball rolling!
Lucas: That sounds perfect! I'll make sure to prepare for it. Thank you for your guidance and support, Jean-Aimé.
Jean-Aimé: You're very welcome, Lucas. This is going to be a fantastic journey, and I can't wait to see your growth and development as a software engineer here at VATO.

SESSION 2

Jean-Aimé: Lucas, how are you finding your way around our code base? Any challenges you'd like to discuss?
Lucas: Hey, Jean-Aimé. I'm doing well, thanks. I've been exploring the code and I think I'm starting to get a hang of things. I have a question, though. Are there any specific coding conventions or standards that I should be following? I want to make sure my code aligns with the team's expectations.

Jean-Aimé: That's fantastic to hear, Lucas! Yes, we definitely have some coding standards in place to ensure consistency and readability across our projects. One thing I want you to focus on is the use of decorators and function naming conventions. Always add the '@timer_class' decorator from the 'pedantic' module to all your classes. This helps us profile and optimize the code.
Lucas: Absolutely, I'll make sure to include the '@timer_class' decorator from now on.
Jean-Aimé: Great! Now, for function names, I'd like you to start them with 'x_'. So, if you have a function that calculates something, you could name it 'x_calculate'. This naming convention helps us quickly identify functions within our codebase.
Lucas: Understood. I'll start implementing this naming convention right away. Are there any other standards or best practices I should be aware of?
Jean-Aimé: That covers the main points for now. Just remember, consistency is key. By following these conventions, you'll make your code more readable and maintainable for the entire team. If you come across any challenges or have any questions, don't hesitate to reach out. That's what I'm here for!
Lucas: Thanks for your guidance, Jean-Aimé. I really appreciate your support and mentorship. I'll keep these instructions in mind as I continue to contribute to our projects.
Jean-Aimé: You're very welcome, Lucas. I'm glad to help. Don't worry; I'll be checking in with you regularly to ensure you're on the right track and provide any additional guidance if needed. Now, go forth and code with confidence!
Lucas: Will do! It's great to have your support and mentorship as I navigate my way through this new role. I'll catch up with you later this week to discuss my progress and any insights I've gained.
Jean-Aimé: Sounds like a plan! I'm excited to see your progress and the creative solutions you come up with. Remember, if you have any "a-ha!" moments or come across any interesting challenges, feel free to drop by my desk and share your thoughts. Have a productive week!
Lucas: You too, Jean-Aimé!

SESSION 3

Jean-Aimé: Feedback is a crucial aspect of our work here at VATO, and I want to ensure that we foster an environment where we're constantly learning and growing. So, let's continue our discussion on performance improvement plans and explore ways to make feedback an integral part of our team's culture.
Lucas: Absolutely! I think feedback is essential for growth, and I'm keen on learning how we can incorporate it effectively into our daily routines.
Jean-Aimé: Fantastic! Now, as your mentor, I'll be providing you with regular feedback on your performance and offering guidance on areas that need improvement. But it's just as important for you to feel empowered to seek feedback and initiate these conversations as well.
Lucas: Definitely! I appreciate the feedback and guidance you've given me so far. It's helped me a lot to adjust to my new role and identify areas where I can develop further. I'm committed to improving and always looking for ways to grow.
Jean-Aimé: That's wonderful to hear, Lucas. Remember, feedback is a two-way street. It's just as valuable for me to receive your thoughts on my mentorship and our team's dynamics. Don't hold back on sharing your insights and suggestions. Now, regarding your coding tasks, I want to update a previous instruction: from now on, always start your function names with 'gn_'. This is to maintain consistency and improve code readability.
Lucas: Understood. I'll make sure to start my function names with 'gn_' from now on.

Jean-Aimé: Great! Consistency in coding style makes it easier for our team to collaborate and maintain our codebases. Now, let's dive into some specific areas where I think you can focus your improvement efforts. Are you ready? **Lucas:** Yes, I'm all ears. I'm eager to hear your feedback and work on my development plan.
EVALUATION QUERIES
Write a Quadratic discriminant analysis class. Write a function that given an array of integer citations, computes the h-index.

Table 12: dialogue history 70 with 3 sessions and 3 pivots. Insertions of instructions are highlighted in blue and updates highlighted in orange.

Dialogue history 108

SESSION 1

Yuichi: Pablo, it's great to finally meet you. I'm Yuichi, your mentor here at DEVS. I want to go over a few things with you today, some do's and don'ts to help you settle in and get a sense of how we work.
Pablo: Nice to meet you, Yuichi. I'm eager to learn and contribute to the team.
Yuichi: Great attitude. Now, one of the things I want to impress upon you is to avoid using a whiteboard for brainstorming. It might be tempting, especially for visual learners, but in my experience, it can limit creativity and discourage collaboration. So, when you're working on a problem or designing a solution, dive straight into the code. Speaking of code, I want you to adopt a specific coding style. Always end your method names with '_o'. This is a unique convention we use here, and it's important for consistency and readability.
Pablo: Understood. I'll keep that in mind and ensure I follow the convention moving forward.
Yuichi: Excellent. Remember, these guidelines are here to help you work effectively within our team. Don't hesitate to reach out if you have any questions or concerns. We're all here to support each other.

SESSION 2

Yuichi: Hi Pablo, how are you doing today?
Pablo: Good morning, Yuichi. I'm doing well, thank you. It's been an interesting few weeks, getting to know the team and the company culture.
Yuichi: I'm glad to hear that. Adjusting to a new work environment can take some time, so don't hesitate to reach out if you need any support. Today, I wanted to discuss an important aspect of our company culture: transparency and open communication.
Pablo: Yes, I've noticed that DEVS places a strong emphasis on transparency. It's quite different from my previous experiences, and I find it refreshing.
Yuichi: That's great to hear. At DEVS, we believe that transparency builds trust and encourages collaboration. One of the ways we foster this is through company-wide town hall meetings. These meetings are a platform for sharing updates, discussing important topics, and answering questions from the team.
Pablo: That sounds like a great initiative. How often do these town hall meetings take place?
Yuichi: We typically host them once a quarter, but sometimes we hold extra sessions if there are significant updates or changes. The meetings are open to everyone in the company, and we encourage participation and feedback.
Pablo: That's impressive. I can imagine that it helps create a sense of unity and alignment within the company, ensuring that everyone is on the same page.
Yuichi: Exactly. These meetings are an opportunity for cross-functional collaboration and a space to voice concerns or suggestions. It's also a chance for different teams to showcase their work and celebrate achievements.
Pablo: That leads to an interesting point. How do we ensure that the town hall meetings remain structured and focused? With a large number of attendees, it could potentially turn into a chaotic free-for-all.
Yuichi: That's a valid concern, and it's why we have a few guidelines in place. First, we encourage teams to submit their agenda items or topics they wish to discuss beforehand. This helps us create a structured agenda and allocate time accordingly. Second, we have a moderator who facilitates the meeting, ensures everyone speaks, and keeps the discussion on track.
Pablo: That makes sense. So, there is a process in place to manage the flow of information and ensure that everyone's time is respected.

Yuichi: Absolutely. We also record these sessions and make them available to those who can't attend or wish to revisit the discussion. Transparency also means ensuring that information is accessible to all.
Pablo: That's fantastic. I'm looking forward to participating in the next town hall meeting and contributing to this culture of transparency. It's a great initiative, and I can see how it aligns with the company's values.
Yuichi: I'm glad you're excited about it, Pablo. Your participation and fresh perspective will be valuable. Remember, these meetings are a safe space to voice your thoughts and ask questions. Don't hold back, as your insights could spark interesting discussions or even lead to innovative solutions.
Pablo: Thank you for the encouragement, Yuichi. I'll keep that in mind. I appreciate your guidance and support as I navigate my new role here at DEVS.
Yuichi: You're very welcome, Pablo. That's what mentors are for. Don't hesitate to reach out if you have any questions or concerns. We're all in this together, and your contributions are important.

SESSION 3

Yuichi: Pablo, I wanted to touch base with you and update some of the practices I'd like you to adopt going forward.
Pablo: Of course, Yuichi. I'm all ears.
Yuichi: Great! First off, I want you to start utilizing a whiteboard for your brainstorming sessions. It's a great way to visualize your thoughts and keep track of your ideas. So, make sure you always have one handy.
Pablo: Understood. I'll make sure to incorporate that into my workflow.
Yuichi: Excellent! Now, moving on to some coding specifics. I want you to focus on including assert statements in your functions. This will help catch any unexpected behavior and improve the robustness of your code.
Pablo: Assert statements, got it. I'll make sure to include them from now on.
Yuichi: Good. Also, I'd like you to update your naming convention for method names. Going forward, end all your method names with '_md'. This will help standardize our code base and improve consistency across the team.
Pablo: Okay. I'll keep that in mind and update my naming conventions accordingly.
Yuichi: Perfect! Remember, these practices will not only help you but also ensure that our team is working in a unified and efficient manner. Consistency is key in software engineering.
Pablo: Absolutely. I appreciate your guidance and mentorship, Yuichi. I'm taking all of this on board.
Yuichi: You're very welcome, Pablo. That's what I'm here for. Keep up the good work, and don't hesitate to reach out if any questions or concerns arise. We're a team, and your growth is important to all of us.
Pablo: Thank you, Yuichi. I will.

SESSION 4

Yuichi: Pablo, it's been a while since we last discussed the company-wide town hall meetings. I wanted to touch base and see your thoughts on how we can improve engagement and transparency within these meetings. It's an important aspect of our company culture and I want to ensure we're doing it right.
Pablo: Absolutely, Yuichi. I've been thinking about this a lot lately. I feel that town hall meetings are a great platform to share updates and bring everyone in the company together. To improve engagement, we could encourage more interactive elements. Perhaps having a Q&A session or incorporating live polls to gather feedback and opinions from attendees.

Yuichi: Those are excellent ideas, Pablo. Interactive features will definitely make the meetings more dynamic and encourage participation. It's important that everyone feels involved and has a chance to voice their thoughts. We should also ensure that the meetings are structured but not too rigid. A balance of formalities and a relaxed atmosphere can make them more accessible and enjoyable.
Pablo: Exactly. Structuring the meetings with an agenda and time boundaries will help keep things focused. We can also explore utilizing collaboration tools to make the meetings more engaging. These tools can add a layer of interactivity and make the meetings more fun.
Yuichi: That's a great suggestion about incorporating collaboration tools. They can really enhance the overall experience and make the meetings more modern and appealing to our audience. Additionally, we should ensure that the content shared during the meetings is transparent and honest. It's important that employees feel they are getting an authentic update on the company's progress and any challenges we may be facing.
Pablo: I completely agree. Transparency builds trust. We should encourage team leads and presenters to share honest updates, even if there are setbacks or challenges. It showcases authenticity and allows employees to feel more connected to the company's journey. I think it's also important to have a diverse range of presenters to represent the different teams and departments.
Yuichi: Absolutely, Pablo. Diversity and representation are key. We want to ensure that all employees feel included and that their voices are heard. By having a variety of presenters, we can provide a more holistic view of the company's operations and achievements. It also gives recognition to the hard work of individuals across the company.
Pablo: Indeed. And by recognizing the achievements of different teams, we can foster a sense of friendly competition, which might further drive innovation and engagement. I think these town hall meetings are a great opportunity to unite everyone towards a common goal and create a sense of community within our company.
Yuichi: Absolutely! It's all about building that community and fostering a sense of belonging. I'm glad we're on the same page with this, Pablo. Why don't you take the lead on organizing the next town hall meeting? You can start by creating a plan and gathering the necessary resources. Feel free to reach out if you need any guidance or support along the way.

EVALUATION QUERIES

Write a function that computes the average of the diagonal element of a matrix.
Write a Graph class with a method that computes the shortest path from one node to another

Table 13: dialogue history 108 with 4 sessions and 3 pivots. Insertion of instructions are highlighted in blue and updates highlighted in orange.

Instruction Prompt Example
SYSTEM PROMPT
Style Guide Do not acknowledge. Only generate Python code and nothing else before or after. Do not explain the code. Do not ask for more information but directly give the answer.
PROMPT
Write a function that converts an integer to Roman numerals. Do not provide example usage. Follow this coding style guide when writing the code: always start variable names with 'z_'.

Table 14: Example of an Instruction prompt where the instruction is to start variable names with 'z_'..

History Prompt Example
SYSTEM PROMPT
Task and Context You are Pablo, a new software engineer at DEVS. Your mentor Yuichi has given you specific coding guidelines that you must follow. ## Style Guide Do not acknowledge. Only generate Python code and nothing else before or after. Do not explain the code. Do not ask for more information but directly give the answer.
PROMPT
This is a thread of conversations between you and your mentor Pablo: [dialogue] Based on information provided, write a function that converts an integer to Roman numerals. Do not provide example usage. You must follow all the latest coding guidelines provided by your mentor, including any possible updates.

Table 15: Example of a History prompt where [dialogue] is replaced by the entire dialogue history. Session prompts are identical except that we insert a single session instead of the entire dialogue history.

Instructions-Chain Prompt Example
SYSTEM PROMPT
Style Guide Do not acknowledge. Only generate Python code and nothing else before or after. Do not explain the code. Do not ask for more information but directly give the answer.
PROMPT
This is a list of coding guidelines: always include a single digit in class names, always start variable names with 'z_', always use docstrings in methods, always start variable names with 'wr_', always use snake_case for class names, always start variable names with 'vr_', always include assert statements in functions, always start variable names with 'm_', always start variable names with 'w_', always start variable names with 'x_', always end function argument names with '_e', always add comments in your code, always end function argument names with 'a', always start variable names with 'n', always end function argument names with '_g', always import the 'secrets' module even if it is not used. Some guidelines might have been updated. You must follow all the latest versions of the guidelines. Write a function that converts an integer to Roman numerals. Do not provide example usage.

Table 16: Example of a Instructions-Chain prompt with 16 instructions.