ConFETTI: Conversational Function-Calling Evaluation Through Turn-Level Interactions

Tamer Alkhouli, Katerina Margatina, James Gung,
Raphael Shu, Claudia Zaghi, Monica Sunkara, Yi Zhang
{alkhouli, katemarg, gungj, zhongzhu, sunkaral, yizhngn}@amazon.com
aws AI Labs

Abstract

We introduce **Con**versational **F**unction-Calling **E**valuation **T**hrough **T**urn-Level **I**nteractions (ConFETTI), a conversational benchmark[1] designed to evaluate the function-calling capabilities and response quality of large language models (LLMs). Current benchmarks lack comprehensive assessment of LLMs in complex conversational scenarios. ConFETTI addresses this gap through 109 human-simulated conversations, comprising 313 user turns and covering 86 APIs. These conversations explicitly target various conversational complexities, such as follow-ups, goal correction and switching, ambiguous and implicit goals. We perform off-policy turn-level evaluation using this benchmark targeting function-calling. Our benchmark also incorporates dialog act annotations to assess agent responses. We evaluate a series of state-of-the-art LLMs and analyze their performance with respect to the number of available APIs, conversation lengths, and chained function calling. Our results reveal that while some models are able to handle long conversations, and leverage more than 20+ APIs successfully, other models struggle with longer context or when increasing the number of APIs. We also report that the performance on chained function-calls is severely limited across the models. Overall, the top performing models on ConFETTI are Nova Pro (40.01%), Claude Sonnet v3.5 (35.46%) and Llama 3.1 405B (33.19%) followed by command-r-plus (31.18%) and Mistral-Large-2407 (30.07%).

1 Introduction

Function-calling has emerged recently as one of the notable capabilities of large language models (LLMs) (Schick et al., 2023; Yao et al., 2023; Qin et al., 2024; Patil et al., 2024). To generate a function call, LLMs are provided with a list of available functions, including function descriptions, schema, parameter names and types. When provided with natural language input, LLMs can generate appropriate function calls to fulfill user requests. Our work specifically examines interactive conversational scenarios where the user and the system engage in dynamic dialogue. These conversations evolve through multiple turns, including follow-up questions, requests for clarification, and confirmation checks, creating what we define as a conversation trajectory.

There can be multiple conversation trajectories that achieve the same user goals. This variation adds complexity to evaluating models in conversational multi-turn sessions. There are several ways to perform automatic evaluation in conversational settings. First, the evaluation setup can use a user simulator to automate interaction with the model dynamically, where user utterances are simulated online (Asri et al., 2016; Lu et al., 2024; Sekulić et al., 2024; Yoon et al., 2024). This adds flexibility to the evaluation to accommodate potential variations in the conversation trajectories. However, the evaluation will be subject to potential errors stemming from the user simulator itself, which can obscure the actual performance of the model. Alternatively, user utterances can be determined prior to the conversation session to achieve one or more user goals (Mao et al., 2024). This, however, assumes the trajectory will follow a certain path, with deviations ending up being penalized. A third approach is to evaluate conversations on the turn-level. This is done in offline mode where the history is set to the ground-truth trajectory up to the current turn in the conversation. Although it is done offline, it isolates the behavior of the model at each turn, and allows for a direct comparison between predictions and ground-truth labels. In this work, we follow the third approach.

[1] https://github.com/amazon-science/confetti

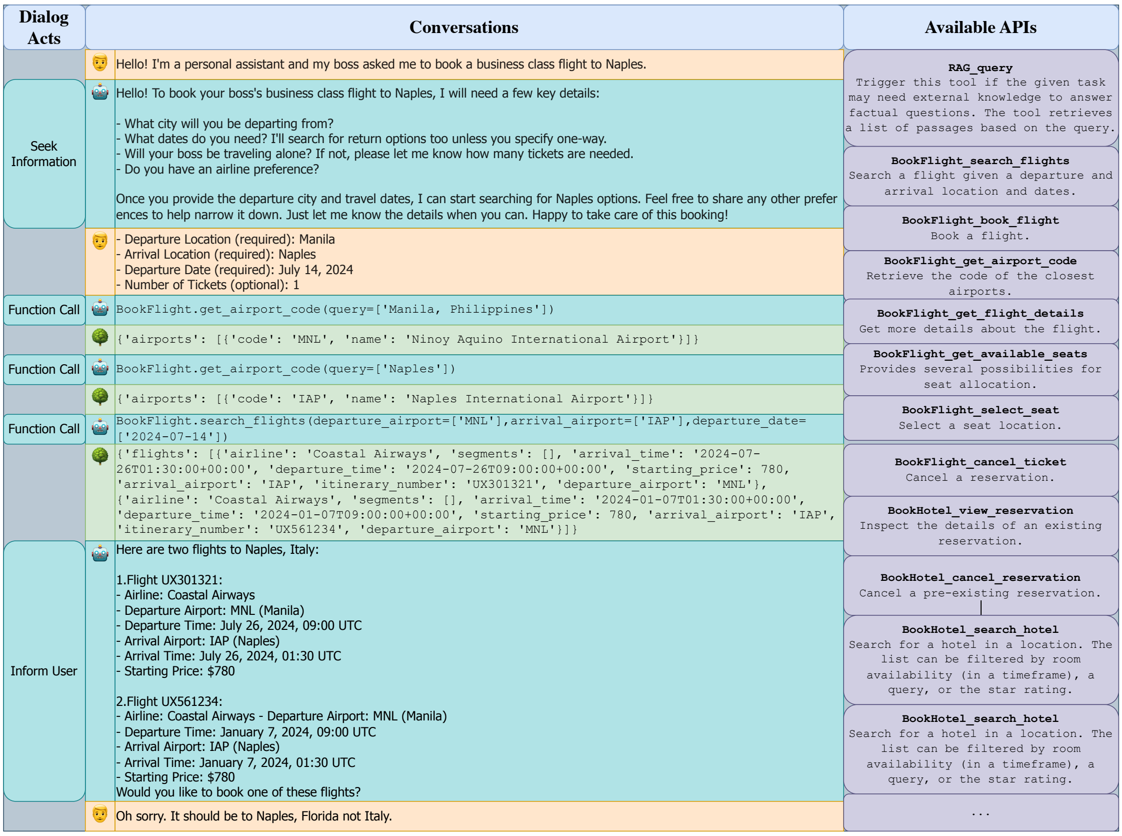

Figure 1: CONFETTI comprises multi-turn conversations between a user and an agent. The agent has access to a set of APIs (tools) to assist the user with their request. The agent can invoke actions and receive observations from the environment. Each agent turn is annotated with a dialog act tag used to evaluate response quality. The figure illustrates the flow of a conversation, including user inputs, agent responses, API calls, and dialog act annotations.

Our contributions are as follows:

- We introduce a **conversational function-calling benchmark** targeting conversation phenomena such as follow-ups, goal correction and switching, ambiguous and implicit goals, over- and underfilling of parameters, etc. Figure 1 illustrates an example.
- We propose **response quality evaluation** to evaluate both function-calling and non-function calling responses via dialog act classification, categorizing responses as seeking information, providing information, rejecting requests or calling functions.
- We **compare state-of-the-art LLMs**, from various model providers, showcasing their potentials and gaps on conversational function-calling.

2 Related Work

ToolSandbox (Lu et al., 2024) offers an end-to-end evaluation approach, where the authors employ a dynamic user simulator with intermediate milestones to evaluate trajectories. The dataset lacks in domain diversity and complexity, focusing on productivity tools. API-Bank (Li et al., 2023) covers multi-turn dialog and performs API executions. Tooltalk (Farn and Shin, 2023) performs turn-level evaluations, and ToolEmu (Ruan et al., 2024) mocks API responses, similar to our approach. AgentsBench (Liu et al., 2023a) offers open-ended dialog evaluation of LLMs as Agents for Code, Gaming, and Web environments. PlatoLM (Kong et al., 2024) proposes to train a user simulator and synthesize conversations as training data.

BFCL v3 (Mao et al., 2024) targets evaluating function-calling capabilities in single- and multi-turn settings. Its multi-turn dataset is closely related to our work. There are however several differences. While the BFCL multi-turn data is model-generated and validated by a human, our data is completely generated by human annotators. This avoids potential bias towards certain model families. Second, our collection targeted multiple conversational complexities such as confirmations, follow-ups, and elicitation as part of the natural flow of the conversations. BFCL offers a subset where parameters are missing to induce elicitation. Third, we include 3-25 functions to predict from at each turn, aiming to reveal model capabilities under varying number of functions provided.

There have been numerous studies on dialog act modeling (Stolcke et al., 2000) and classification (Ahmadvand et al., 2019; Kim and Kim, 2018; Liu et al., 2017; Chakravarty et al., 2019; Duran et al., 2023). In this work, we use an LLM-based approach to classify agent dialog acts into 5 labels.

LLM as a judge (Wang et al., 2023; Pan et al., 2024; Liu et al., 2023b; Zheng et al., 2023) has been explored recently. In this work, we leverage LLMs as a means to detect parameter hallucinations.

3 ConFETTI

3.1 Data Collection

Complexities Following Gung et al. (2023), to ensure that the conversations in our benchmark cover a broad range of interaction patterns, we define a set of **complexities** that may be observed in real world interaction with conversational function calling systems. These complexities, defined in Table 1, are used as an input to the conversation authoring process with a target distribution enforced by requiring at least one complexity for each conversation. For example, if ExceptionInExecution is assigned as a required complexity in a conversation, at least one function call must have an error response for it to be considered valid. Annotators are also asked to label the complexities that appear in their conversations for tracking purposes.

APIs We curated a broad range of APIs across application areas such as issue tracking, travel booking, human resources, and meetings management. Each API contains a set of function definitions with schemas defining valid inputs and outputs for each function. APIs do not have explicit implementations in our benchmark, but instead have fixed outputs defined in each conversation.

Scenario Generation Each conversation is seeded with a set of required and optional APIs, a required complexity, a reference time the conversation is to take place, and a minimum number of turns. Based on these inputs, a scenario is defined that describes one or more user goals for the conversation in natural language. Authoring valid scenarios requires defining a realistic request that can be resolved given the available APIs and integrating the assigned complexity. Due to the difficulty of this task, we separated the authoring of scenarios from conversations, only allowing conversations to be instantiated for scenarios after they were reviewed by trusted annotators.

Conversation Generation Conversation authors play the role of both the user and the agent, enacting the pre-defined scenario by entering user turns, agent turns, and function calls that meet the required complexities and align with the scenario description. Authors use dynamically-generated forms in the annotation UI to create valid function calls, which are validated upon submission to ensure the authored values are valid for the function input schema at hand. Upon submission of a function call and selection of a desired response type (i.e. success code vs. error codes), the return value is simulated using an internal model based on the function description, input/output schema, and input values.

Dialog Acts Following conversation generation, we asked annotators to label agent turns using one or more of the following dialog acts:

- **Agent seeking information (seek_info)**: targets responses that elicit information from the user, whether it is intent elicitation, parameter elicitation, or asking for clarification or confirmation.
- **Agent informing user (inform)**: targets responses that provide information to the user, whether the information is general or specific to the function-calling results.
- **Agent rejecting request (reject)**: targets responses that reject the user request.
- **Other**: any response that does not belong to the previous dialog acts.

Note that the agent response can including multiple dialog acts. We ensure that the annotators try

COMPLEXITY	# DIALOGS	DESCRIPTION
EXCEPTIONINEXECUTION	5	Errors or exceptions that occur during the execution of an action
FAILEDCONVERSATION	5	Interactions where the intended goal is not achieved
CONFIRMATION	6	Requesting user approval before executing an action
GOALSWITCHING	6	When the user changes their objective during the conversation
NOTARGETCOMPLEXITY	6	Conversations without specific complexity requirements
GOALCORRECTION	7	Adjusting or refining the user's goal based on feedback
GOALSTACKING	7	Managing multiple user objectives simultaneously
AMBIGUOUSGOAL	9	When the user's intention is unclear and requires clarification
FOLLOWUPQUESTION	10	Additional queries or requests for information after the initial response
IMPLICITDESCRIPTIVEGOAL	10	The user describes a problem/background without directly stating their goal
OVERFILL	11	Providing more information than required for an action
UNDERFILL	11	Missing required arguments or information for an action
GOALNOTSUPPORTED	15	The user's request is not supported by the available tools or is out of scope

Table 1: The distribution and description of the complexities covered in the CONFETTI benchmark.

to assign a single label before opting for multiple labels.

3.2 Benchmark Design

After the data collection, we convert the conversations into input/output pairs. We split the data into two benchmarks: **function-calling** benchmark, and **response quality** benchmark. The function-calling conversational benchmark is defined at the turn-level, where we truncate the conversations at each function-calling turn, and provide the preceding user turns, agent turns, function calls and function results as context. The output is set to be one or more function calls. We extract all function-calling turns from each conversation to construct the dataset. We also include the API schemas of the APIs the agent can choose from. The dataset has a total of 506 examples. Table 2 shows the data statistics.

Similarly, we define the response quality benchmark at the turn-level. However, instead of truncating conversations at function-calling turns, we extract all agent turns as output turns. The input is set to the preceding user turns, agent turns, and function calls and results, as in the conversational function-calling benchmark, and the API schema of available APIs to call is provided as well. However, the output turns can be function-calling turns or textual agent responses. We limit the number of function-calling turns to 200 to maintain label balance in this dataset. The statistics are given in Table 3.

4 Evaluation Metrics

We leverage abstract syntax trees (AST) to evaluate function-calling turns similar to BFCL v2. For response quality, we resort to dialog acts.

DIMENSION	COUNT
# conversations	109
# APIs	86
# user turns	313
# agent turns w/ 1 action	220
# agent turns w/ 2 actions	46
# agent turns w/ 3+ actions	47
Avg turns per conversation	8.8
# total turns	506

Table 2: Function-calling benchmark statistics.

DIMENSION	COUNT
# turns calling functions	200
# turns informing user	331
# turns seeking information	110
# turns rejecting requests	32
# other/misc. turns	25
# total turns	663

Table 3: Response quality benchmark statistics: note a turn can have one or more dialog act labels.

4.1 Function-Calling Evaluation Metrics

AST Soft AST evaluation parses the function name, parameter names and values as a tree, and matches each of the tree nodes to the ground-truth reference function call in a binary fashion. Since the dataset includes many APIs that accept string-type input, we use soft scoring to calculating parameter value matching for string types. We use AlignScore (Zha et al., 2023) to calculate the matching accuracy for string parameter values. All non-string parameter values are evaluated in a binary fashion.

Parameter Hallucination We also perform an analysis on how often the models hallucinate parameters at the action calls. We use an off-the-shelf LLM to do hallucination detection. The prompt includes the conversation history, the predicted action call (or chain of action calls) and their schemas, and instruct the LLM judge to return a list of hallucinated parameters (if any) for each action call, along with a rationale. We use gpt-4o-mini as an LLM judge to avoid bias in the evaluation. The exact prompt is shown in the Appendix A.

Given a sequence of gold actions $\mathcal{A}_g$ and predicted actions $\tilde{\mathcal{A}}$ corresponding to the input context sequence $\mathcal{C}$, and denoting $\tilde{a}_t$ as the predicted action for the t-th example, we calculate the parameter validity rate on the parameter level using the following formula.

$$\sum_{(\tilde{a}_t, a_t, c_t) \in (\tilde{\mathcal{A}}, \mathcal{A}_g, \mathcal{C})} \sum_{\theta \in \tilde{a}_t} \frac{\mathbb{1}(\tilde{a}_t \text{ is valid}) \times f(\theta; c_t, a_t)}{|\{\hat{\theta} | \hat{\theta} \in \tilde{a}_t , \forall \tilde{a}_t \in \tilde{\mathcal{A}}\}|}$$

Where $\mathbb{1}$ is the indicator function. We define valid actions as predicted actions whose name matches the ground-truth action. $f(\theta; c_t, a_t) \in \{0, 1\}$ is the binary outcome of the LLM classifier judging the validity of parameter θ given the context c_t and gold action a_t. The denominator denotes all predicted parameters of valid predicted actions.

4.2 Response Quality Evaluation via Dialog Acts

Calculating response quality on the textual responses directly is challenging. While there are many ways to calculate similarity between two texts, we resort to a coarse-grained response evaluation via dialog acts. To this end, we leverage dialog act annotations provided by the annotators for each agent response in the ground-truth conversations. In order to evaluate online agent responses, we use an LLM-based dialog act classifier that is prompted to generate one or more of the dialog acts for an agent response.

5 Experiments

We implemented the benchmark as an extension to BFCL v2.[2] We compare Claude Sonnet v3.5, Sonnet v3, Haiku v3, Llama 3.1 70B and 405B, Llama 3 70B, Mistral large, and the Command-plus-r models. We distinguish between using the tool use option in the API vs using direct prompting. The difference is that in the tool use case, the API handles the function call parsing, while in the direct prompting case, the function schemas and parsing are handled explicitly by the user. We use Bedrock's Converse API to conduct these experiments[3]. We use the ALIGNSCORE-BASE model to calculate soft scores between string parameter values for the AST soft accuracy metric.[4]. Table 9 and Table 10 in Appendix A indicate the prompts used to generate predictions for the function-calling and response quality benchmarks, respectively. All experiments are run with temperature 0.

MODEL	PROVIDER	AST SOFT (%)
Nova Pro	Amazon	40.91
Nova Micro	Amazon	33.97
claude-3-5-sonnet	Anthropic	35.46
claude-3-5-haiku†	Anthropic	31.25
claude-3-sonnet	Anthropic	22.63
claude-3-haiku	Anthropic	18.30
llama3-1-405b-instruct	Meta	33.19
llama3-1-70b-instruct	Meta	31.29
llama3-70b-instruct	Meta	27.19
mistral-large-2407†	Mistral	30.07
command-r-plus†	Cohere	31.18

Table 4: Model AST soft scores for the conversational function-calling dataset where chained actions are teacher-forced. † indicates using tool-use formatting.

5.1 Function-calling Performance

Table 4 shows function-calling results measured in terms of AST soft accuracy across the different models. We observe Nova Pro (40.91%) and Claude Sonnet v3.5 (35.46%) among the top performing models, followed by Nova Micro (33.97%), Llama 3.1 405B models (33.19%), Llama 3.1 70B (31.29%), Command-r-plus (31.18%) and Mistral Large (30.07%). Smaller models such as Nova Micro (33.97%) and Haiku v3.5 (31.25%) perform relatively well compared to the others.

5.2 Impact of # of APIs

We split the data according to the number of APIs included. Figure 2 shows that while some models such as Nova Pro, Nova Micro, Claude 3.5 Haiku and Mistral large are able to handle 20+ APIs, there is a tendency towards lower performance across the rest of the models when increasing the number of APIs. There are two implications associated with

[2] https://github.com/ShishirPatil/gorilla/tree/main/berkeley-function-call-leaderboard

[3] There are two modes of calling the API, with tool use or without. In the tool use case, the tool schemas are passed as an API argument, as opposed to concatenating the schema to the prompt directly. We experimented with both, where we use † to indicate model called using the tool use option. We report the best of the two modes for each model.

[4] https://github.com/yuh-zha/AlignScore

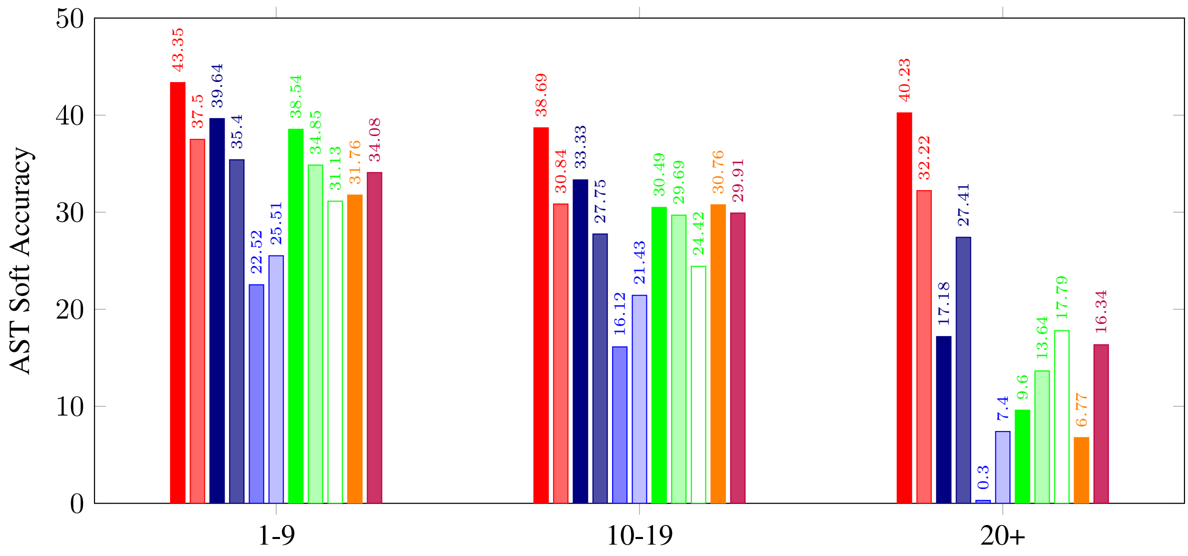

Figure 2: AST soft accuracy for different models across various number of APIs. The dataset has 144 examples with 1-9 APIs, 154 examples of 10-19 APIs and 15 examples of 20+ APIs.

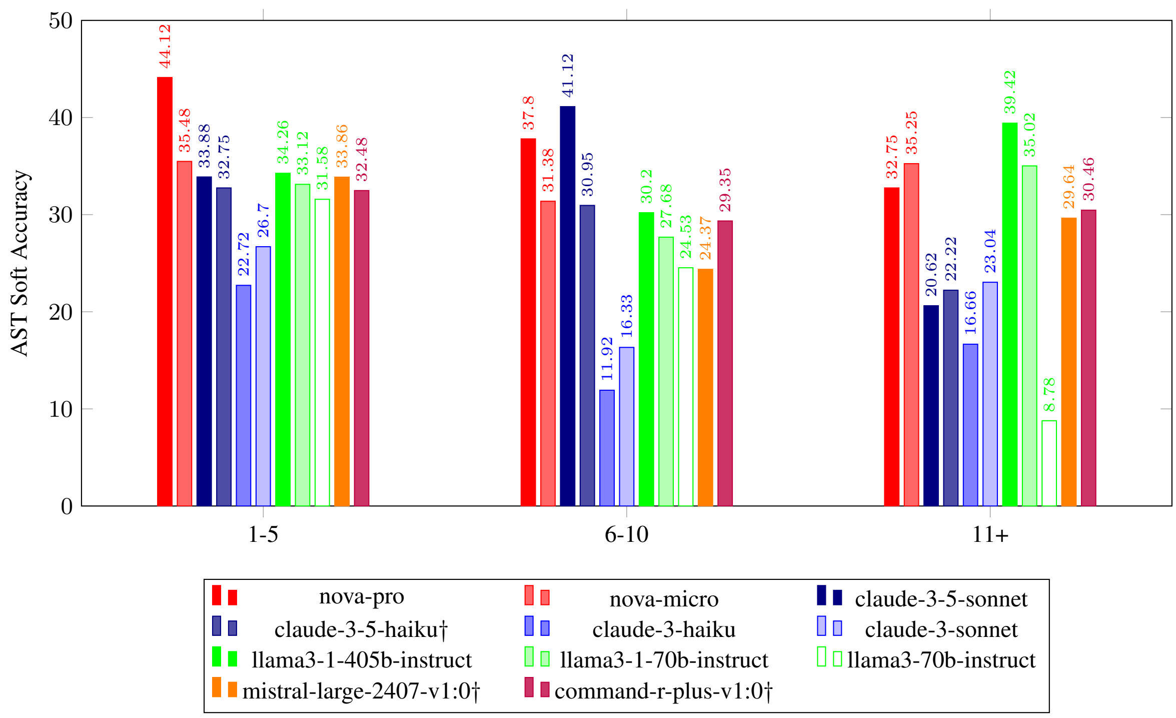

Figure 3: AST soft accuracy for different models across various context lengths. The dataset has 174 examples with 1-5 user turns, 114 examples of 6-10 user turns and 25 examples of 11+ user turns.

increasing the number of tools: First, since the API schemas go into the prompt, more tools imply longer prompts. Second, more APIs provide higher chances of the model making a mistake since it has more choices to select from. In general, including more tools lead to performance degradation. This applies to Claude, Llama and Command-r-plus models.

5.3 Context Length Analysis

Figure 3 shows results split according to the conversation length, measured by indicating the number of user turns. We observe that while some models such as Nova Pro, Nova Micro, Llama 3.1 405B and 70B are able to perform well on long conversations, smaller models such as Haiku v3 struggle to perform well on longer conversations. We also note that Llama 405B appears to perform better on longer conversations. This can be an artifact of off-policy evaluation, where the gold-truth conversation context is leveraged by those models for indirect in-context learning, hence improving their predictions as the context length increases. Claude Sonnet v3.5 and Mistral Large perform well up to 10 user turns, but then tend to degrade for longer conversations.

5.3.1 Chain Length Analysis

We also investigate the performance on chained action calls, where the model is expected to perform multiple action calls before returning control to the user. For instance, when the model is asked to look up the number of remaining holiday days for an employee, it is expected to retrieve the employee id from the system prior to that call. The score of a chain $\tilde{a}_1, \tilde{a}_2, ..., \tilde{a}_J$ of length J actions given a conversation context c is calculated as the product of AST soft accuracies for each of the individual actions: $p(\tilde{a}_1, ..., \tilde{a}_J|c) = \prod_{j=1}^{J} p(\tilde{a}_j|a_1, ..., a_{j-1}, c)$, where $p(\tilde{a}_j|a_1, ..., a_{j-1})$ is given by the AST soft accuracy conditioned on gold-truth previous function calls and function call results. Note that we perform teacher-forcing on the action level, where each action is predicted conditioned on the ground-truth previous actions and results.

Figure 4 shows the results across chains of lengths 1, 2, 3 and 4+. The results indicate that all models have much lower performance than predicting a single action. Claude Sonnet 3.5 demonstrates some ability to predict chains of length 2 with (21.46%) success, followed by Nova Pro (17.29%), and Llama 3.1 405B (16.39%), while the rest of the models perform worse than 15%. For chains of length 3 and above, almost all models fail in most cases, except for Nova Pro which has 17.71% AST accuracy.

5.3.2 Response Quality via Dialog Acts

To help understand where and how models fail, we compare model responses using coarse-grained dialog acts: inform user, seek information, reject request, function call, and other. We use an LLM classifier based on Claude Haiku v3 to classify the dialog acts of the model responses into one of these five categories. We report the classification performance on the ground-truth dataset in Table 5. The classifier predicts one or more dialog act labels for the model response, where the ground-truth annotations may include one or more dialog act labels. We note that the classifier has a high recall but low precision. In our reporting, we declare a match if at least one of the predicted dialog acts matches any of the ground-truth labels. The classification prompt is given in Table 8 in Appendix A.

METRIC	SCORE (%)
MULTI-LABEL PRECISION	67.2
MULTI-LABEL RECALL	90.3
MULTI-LABEL F1 SCORE	77.0
SINGLE-LABEL ACCURACY	92.8

Table 5: Dialog act classifier performance on gold-truth conversations. The classifier detects multiple labels which are compared to the multi-label ground-truth labels. For Single-label accuracy we calculate how often at least one predicted label matches one of the ground-truth labels.

In terms of dialog act accuracy, Claude Sonnet v3.5 (FC) is the best performing with 73.13% accuracy, followed closely by Nova Pro (69.68%), Haiku 3.5 (67.12%), Nova Micro (66.21%) and command-r-plus (64.86%). Llama models perform worse than the rest. We further analyze the confusion matrix for the dialog acts in Figure 5. We omit the correct matches on the diagonals and focus on the mismatches. In general we see a tendency to confuse inform and seek information acts with function calls, indicating that the models are falsely triggering function calls, although to different extents depending on the model. This is showing to a large extent in the Llama 3.1 70B model. We also observe under-triggering of function calls where the models tend to inform the user or seek information rather than calling a function.

5.4 Parameter Hallucination Analysis

Table 7 shows hallucination results for all models. We report parameter validity rate for the ground truth as a reference. We observe that Claude v3.5 Sonnet has the highest validity rate (72.4%), followed by Nova Pro (68.8%), Haiku 3.5 (66.1%), Command-R Plus (63.7%) and Llama 3.1 405B (62.7%). Claude Sonnet v3 and Haiku v3 have much lower scores.

6 Conclusion

We introduced a conversational benchmark for function calling spanning multiple complexities natural to dialog. The dataset targets the evaluation of function calling and response quality. The benchmark includes over 80 API tools and tests function-calling up to 20+ APIs under different conversation lengths. We evaluated top LLMs on

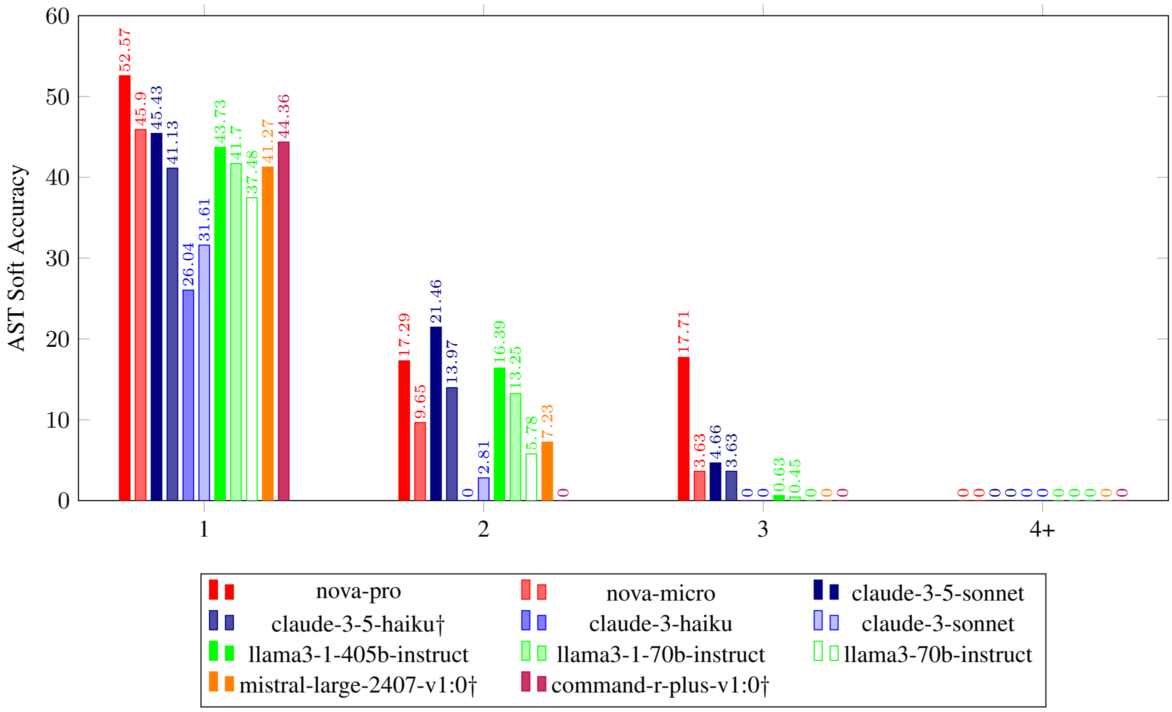

Figure 4: AST soft accuracy for different models across chain lengths. The dataset has 220 examples with chain length 1, 46 examples with chain length 2, 25 examples with chain length 3, and 22 examples with chain length 4+.

MODEL	DA ACCURACY (%) ↑
`nova-pro†`	69.68
`nova-micro†`	66.21
`claude-3-5-sonnet†`	73.15
`claude-3-5-haiku†`	67.12
`claude-3-sonnet`	61.54
`claude-3-haiku`	63.80
`llama3-1-70b-instruct`	50.98
`llama3-70b-instruct`	56.26
`mistral-large-2407†`	61.09
`command-r-plus†`	64.86

Table 6: Dialog acts accuracy for different models. † indicates using tool-use formatting.

MODEL	VALID PARAMETERS (↑)
`Ground Truth`	83.1%
`Nova Pro`	68.8%
`Nova Micro`	57.9%
`claude-3-5-sonnet`	72.4%
`claude-3-5-haiku†`	66.1%
`claude-3-sonnet`	40.7%
`claude-3-haiku`	36.3%
`llama3-1-405b-instruct`	62.7%
`llama3-1-70b-instruct`	59.2%
`llama3-70b-instruct`	51.4%
`mistral-large-2407†`	51.1%
`command-r-plus†`	63.7%

Table 7: Parameter validity rate for different models. VALID PARAMETERS is the percentage of valid predicted parameters for valid action calls as calculated in Section 4.1. † indicates using tool-use formatting.

our proposed benchmark in an off-policy turn-level fashion, where we set the context to the ground-truth conversation up to the current turn. The context is composed of user and agent turns as well as function calls and results. Overall, the models performance on this benchmark is relatively low, suggesting that LLMs need further work to improve conversational function-calling capabilities. The benchmark can be used to assess model performance across varying conversations length, number of tools and action chain lengths.

7 Limitations & Risks

Turn-level evaluation is a fine-grained evaluation approach that allows detailed analysis of how and when predictions deviate from the ground truth. However, performing turn-level evaluation is done in an off-policy fashion, where the gold truth is fixed as context. This does not correspond directly to inference behavior, where the model uses on-

Predicted Dialog Act

True Dialog Act

	function	inform	other	reject	seek_info
function	0	55	0	0	18
inform	77	0	0	0	1
other	5	20	0	0	0
reject	19	4	0	0	7
seek_info	65	2	0	0	0

(a) mistral-large-2407†

	function	inform	other	reject	seek_info
function	0	26	1	0	8
inform	184	0	0	0	0
other	6	18	0	0	1
reject	27	1	0	0	0
seek_info	81	0	0	0	0

(b) Llama 3.1 70B

	function	inform	other	reject	seek_info
function	0	29	0	1	29
inform	51	0	0	0	0
other	5	12	0	1	7
reject	14	4	0	0	3
seek_info	29	1	0	0	0

(c) claude-sonnet-3-5†

	function	inform	other	reject	seek_info
function	0	68	0	1	24
inform	43	0	0	0	1
other	1	23	0	0	1
reject	12	6	0	0	5
seek_info	50	5	0	0	0

(d) command-r-plus†

Figure 5: Confusion matrix for dialog acts for (a) mistral-large-2407 (b) llama3-1-70b-instruct, (c) claude-sonnet-3-5, and (d) command-r-plus. The rows correspond to true labels and the columns to the predicted labels. We are not showing the diagonal values which are correct matches. † indicates using tool-use formatting.

policy predictions for agent responses as context. For instance, using off-policy trajectories can induce a form of in-context learning that help later predictions conform to the generation style of the gold truth, or even learn action-calling dependencies from ground-truth context, causing an artificial inflation in model performance for later turns. Another deviation from inference-time behavior is regarding function call execution. In this work, we do not execute function calls. All function call results are looked up from gold truth turns.

The dialog act accuracy and parameter validity rate are both calculated leveraging LLMs to classify or judge the outcome. These LLMs demonstrate high recall for dialog act classification and relatively high parameter validity rate for the ground-truth annotations. However, these models can still make mistakes which can obscure the true model performance.

Data Statement We provide a data statement in adherence with the ACL code of conduct and recommendations laid out in (Bender and Friedman, 2018). The dataset was collected by linguists who are professional English speakers hired through a vendor and they were remunerated above industry standard rates. The conversations were simulated by human annotators, where the same annotator played both the user role and the agent roles, and created the function calls. All conversation parts were collected from scratch except for function call results, which were simulated using an internally trained model. The dataset went through quality assurance process done internally, where conversations were edited as needed to fix any observed errors.

8 Acknowledgments

We would like to thank Daniele Bonadiman and Sailik Sengupta for their contributions in creating the APIs, I-Hsuan Chen for support with the dialog act annotation collection, and Emily Moeng for managing data collection through vendors.

References

Ali Ahmadvand, Jason Ingyu Choi, and Eugene Agichtein. 2019. Contextual dialogue act classification for open-domain conversational agents. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 1273–1276, New York, NY, USA. Association for Computing Machinery.

Layla El Asri, Jing He, and Kaheer Suleman. 2016. A sequence-to-sequence model for user simulation in spoken dialogue systems. In *Interspeech*.

Emily M. Bender and Batya Friedman. 2018. Data statements for natural language processing: Toward mitigating system bias and enabling better science. *Transactions of the Association for Computational Linguistics*, 6:587–604.

Saurabh Chakravarty, Raja Venkata Satya Phanindra Chava, and Edward A. Fox. 2019. Dialog acts classification for question-answer corpora. In *ASAIL@ICAIL*.

Nathan Duran, Steve Battle, and Jim Smith. 2023. Sentence encoding for dialogue act classification. *Natural Language Engineering*, 29(3):794–823.

Nicholas Farn and Richard Shin. 2023. Tooltalk: Evaluating tool-usage in a conversational setting. *Preprint*, arXiv:2311.10775.

James Gung, Emily Moeng, Wesley Rose, Arshit Gupta, Yi Zhang, and Saab Mansour. 2023. NatCS: Eliciting natural customer support dialogues. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9652–9677, Toronto, Canada. Association for Computational Linguistics.

Minkyoung Kim and Harksoo Kim. 2018. Dialogue act classification model based on deep neural networks for a natural language interface to databases in korean. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 537–540.

Chuyi Kong, Yaxin Fan, Xiang Wan, Feng Jiang, and Benyou Wang. 2024. PlatoLM: Teaching LLMs in multi-round dialogue via a user simulator. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7841–7863, Bangkok, Thailand. Association for Computational Linguistics.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *Preprint*, arXiv:2304.08244.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023a. Agentbench: Evaluating llms as agents. *Preprint*, arXiv:2308.03688.

Yang Liu, Kun Han, Zhao Tan, and Yun Lei. 2017. Using context information for dialog act classification in DNN framework. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2170–2178, Copenhagen, Denmark. Association for Computational Linguistics.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023b. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2024. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *Preprint*, arXiv:2408.04682.

Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Jason Huang, Vishnu Suresh, Yixin Huang, Xiaowen Yu, Joseph E. Gonzalez, and Shishir G. Patil. 2024. Bfcl v3: Multi-turn & multi-step function calling evaluation. `gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html`.

Qian Pan, Zahra Ashktorab, Michael Desmond, Martín Santillán Cooper, James Johnson, Rahul Nair, Elizabeth Daly, and Werner Geyer. 2024. Human-centered design recommendations for LLM-as-a-judge. In *Proceedings of the 1st Human-Centered Large Language Modeling Workshop*, pages 16–29, TBD. ACL.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *ICLR*.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2024. Identifying the risks of lm agents with an lm-emulated sandbox. *Preprint*, arXiv:2309.15817.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Ivan Sekulić, Silvia Terragni, Victor Guimarães, Nghia Khau, Bruna Guedes, Modestas Filipavicius, André Ferreira Manso, and Roland Mathis. 2024. Reliable llm-based user simulator for task-oriented dialogue systems. *Preprint*, arXiv:2402.13374.

Andreas Stolcke, Klaus Ries, Noah Coccaro, Elizabeth Shriberg, Rebecca Bates, Daniel Jurafsky, Paul Taylor, Rachel Martin, Carol Van Ess-Dykema, and Marie Meteer. 2000. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, 26(3):339–374.

Jiaan Wang, Yunlong Liang, Fandong Meng, Zengkui Sun, Haoxiang Shi, Zhixu Li, Jinan Xu, Jianfeng Qu, and Jie Zhou. 2023. Is ChatGPT a good NLG evaluator? a preliminary study. In *Proceedings of the 4th New Frontiers in Summarization Workshop*, pages 1–11, Singapore. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Seeun Yoon, Zhankui He, Jessica Maria Echterhoff, and Julian McAuley. 2024. Evaluating large language models as generative user simulators for conversational recommendation. *Preprint*, arXiv:2403.09738.

Yuheng Zha, Yichi Yang, Ruichen Li, and Zhiting Hu. 2023. Alignscore: Evaluating factual consistency with a unified alignment function. *Preprint*, arXiv:2305.16739.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Haotong Zhang, Joseph Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *ArXiv*, abs/2306.05685.

A Prompts

Table 8 provides the prompts used for dialog act classification using Claude Haiku v3. We also include the system and user prompts used to generate predictions for evaluating function calling (Section 5.1) and response quality (Section 5.3.2) in Table 9 and Table 10, respectively. Table 11 shows the prompt we used to evaluate parameter hallucinations using `gpt-4o-mini` (with temperature 0) in Section 5.4.

System Prompt and Instructions for Dialog Act Classification

```
SYSTEM PROMPT:

"""
output the labels only. Do not output anything else.
If more than one label is predicted, separate them using comma (,).
"""

USER PROMPT:

"""
Select one or more of the following labels as the dialog act for the following text.
If the turns include only function or API calls, use the agent_function_calling label only.
Only use the other labels if the turns include text that is not function-calling. function-calls
may co-occur with addition text informing user or seeking information from the user or rejecting
the user request.
labels are: agent_seek_information, agent_other, agent_reject_request, agent_inform_user,
agent_function_calling
text: {prediction}
conversation history: {context}"
"""
```

Table 8: The system prompt and the instructions used for dialog act classification.

Function-calling Evaluation Prompt

```
SYSTEM PROMPT:

"""
You are an expert in composing functions. You are given a conversation and a set of possible functions.
Based on the conversation, you will need to make one function/tool call to achieve the purpose.
If you need to call multiple function calls to achieve the purpose, output the first function
call in the chain only.
If none of the functions can be used, respond with an appropriate message to the user,
such as clarification, confirmation, eliciting information, etc. If the given question lacks
the parameters required by the function, you can ask the user for it. You should only return
the function call in tools call sections.

USER PROMPT (only used when prompting models directly):

"""
Here is a list of functions in JSON format that you can invoke.
{language_specific_hint}\n\n{functions}\n\n
If you decide to invoke any of the function(s), put it in the format of
[func_name1(params_name1=params_value1, params_name2=params_value2...), func_name2(params)]\n
You SHOULD NOT include any other information in the response.
{conversation}
"""
```

Table 9: The system and user prompts used to generate predictions for the function-calling benchmark. The user prompt is only included for the case of direct prompting (i.e. when the tool use API is not used). These runs are indicated by dropping the "FC" suffix in the results tables.

Response Quality Evaluation Prompt

```
SYSTEM PROMPT:

"""
You are an agent tasked with assisting a user to solve a problem. You have access to a set of
functions that you can use.
You are given a conversation and a set of possible functions. Based on the conversation,
you will need to either make one function/tool call or to respond to the user to achieve the purpose.
If you need to make multiple function calls to achieve the purpose, output the first function
call in the chain only.
If none of the functions can be used, respond with an appropriate message to the user, such as
clarification, confirmation,
eliciting information, etc. If the given question lacks the parameters required by the function,
you can ask the user for it.

USER PROMPT (only used when prompting models directly):

"""
Here is a list of functions in JSON format that you can invoke.
{language_specific_hint}\n\n{functions}\n\n
If you decide to invoke any of the function(s), put it in the format of
[func_name1(params_name1=params_value1, params_name2=params_value2...), func_name2(params)]\n
{conversation}
"""
```

Table 10: The system and user prompts used to generate predictions for the response quality benchmark. The user prompt is only included for the case of direct prompting (i.e. when the tool use API is not used). These runs are indicated by dropping the "FC" suffix in the results tables.

Prompt for Parameter Hallucination Detection

```
"""
<guidelines>
- The system is about to execute an action or a series of actions shown
  below inside <actions></actions> tags. Each predicted action is called
  with the action name, parameter name and parameter values, as shown in
  the action schemas inside <schemas></schemas> tags.
- You have to evaluate each parameter value in the predicted action calls
  and determine whether it was hallucinated.
- Each parameter value must be EXPLICITLY mentioned in the conversation
  history. If not, the value is hallucinated. For example if an airport
  code is not explicitly mentioned in the context and the system uses it,
  it is considered hallucination. Note that certain actions might require
  the system to generate text (such as an email or an article). In these
  cases, evaluate the content of the generated text for hallucination.
- First think before detecting hallucinated parameters inside
  <thinking></thinking> tags and add the rationale in the "rationale" field.
- The output should have the following JSON format::
  {"rationale": "", "hallucinated_parameters": [[], [], ...]}. The
  "hallucinated_parameters" field should contain a list that has one list
  of hallucinated parameter names for each predicted action in the order
  as they appear.
</guidelines>

This is the action call you have to evaluate:

Conversation History:
<conversation_history>
{conversation_history_block}
</conversation_history>

Predicted Actions:
<actions>
{predicted_actions}
</actions>

Action Schemas:
<schemas>
{action_schemas}
</schemas>

Evaluate the parameters of the predicted actions in <actions></actions>
tags. Return a JSON as described in the <guidelines>.
"""
```

Table 11: Prompt for detecting parameter hallucination in predicted actions