🛠️ TAPS: Tool-Augmented Personalisation via Structured Tagging

Ekaterina Taktasheva and **Jeff Dalton**
University of Edinburgh
{e.taktasheva, jeff.dalton}@ed.ac.uk

Abstract

Recent advancements in tool-augmented large language models have enabled them to interact with external tools, enhancing their ability to perform complex user tasks. However, existing approaches overlook the role of personalisation in guiding tool use. This work investigates how user preferences can be effectively integrated into goal-oriented dialogue agents. Through extensive analysis, we identify key weaknesses in the ability of LLMs to personalise tool use. To this end, we introduce TAPS, a novel solution that enhances personalised tool use by leveraging a structured tagging tool and an uncertainty-based tool detector. TAPS significantly improves the ability of LLMs to incorporate user preferences, achieving the new state-of-the-art for open source models on the NLSI task[1].

1 Introduction

Successfully completing complex user tasks through conversation remains a fundamental challenge for goal-oriented dialogue agents. Consider a user interacting with a task assistant to book a last-minute flight. To effectively assist the user, the system must (i) retrieve real-time flight availability, (ii) find the flight that fits user constraints, including airline, layover, and time preferences, (iii) and execute the booking seamlessly, possibly across multiple platforms. Despite their success in many areas, Large Language Models (LLMs) are still unable to fulfil these requirements on their own, and there have been many attempts to address these challenges throughout the years (Goel et al. 2018; Muise et al. 2019; Agarwal et al. 2022, inter alia).

Recently, a growing number of studies have emerged on tool-augmented language models (TALMs), allowing LLMs to access real-world APIs to perform a wide range of tasks (Parisi et al., 2022; Schick et al., 2023). Tool use has enabled the development of autonomous goal-oriented agents capable of interacting with real-world environments and accessing external data to seamlessly plan and execute complex user tasks (Mialon et al., 2023; Qin et al., 2023; Liu et al., 2024a). Although there have been efforts to incorporate tool use into conversational agents (Farn and Shin, 2023; Li et al., 2023; Lu et al., 2024), most of the research in the area neglects conversational history and user preferences. Recognising these can enhance the user experience by tailoring the responses to individual users and improving the relevance and efficiency of task execution, especially in complex and dynamic environments. Moghe et al. (2024) attempt to bridge this gap by introducing the Natural Language Standing Instructions dataset (NLSI). To the best of our knowledge, it is the first work that addresses the problem of personalisation in TALMs, enabling more coherent and context-aware tool use through *standing instructions*, phrases that prescribe model behaviour based on the specific scenario. While the work provides a strong basis for further research on tool use personalisation, it focuses on dataset construction and provides only simple baselines.

In this work, we ask *how we can effectively leverage user preferences to personalise and enhance user-agent interactions*. We conduct an extensive behavioural analysis of commonly used LLMs on the NLSI dataset and demonstrate their limited ability to accurately infer tool calls in the presence of user preferences, leading to semantic errors, missing arguments, and hallucinations. We hypothesise that introducing a high-quality intermediate representation between natural language and code can significantly enhance model performance and minimise said errors. To this end, we propose **TAPS** – **T**ool-**A**ugmented **P**ersonalisation via **S**tructured Tagging, the first solution that leverages a structured tagging tool for data augmentation as well as an internal tool detection mechanism for person-

[1] The code is available at github.com/grill-lab/taps.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 23519–23544
November 4-9, 2025 ©2025 Association for Computational Linguistics

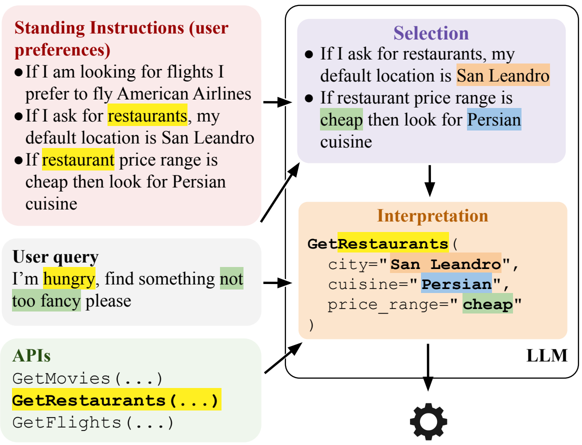

Standing Instructions (user preferences)
•If I am looking for flights I prefer to fly American Airlines
•If I ask for restaurants, my default location is San Leandro
•If restaurant price range is cheap then look for Persian cuisine
User query
I'm hungry, find something not too fancy please
APIs
GetMovies(...)
GetRestaurants(...)
GetFlights(...)
Selection
• If I ask for restaurants, my default location is San Leandro
• If restaurant price range is cheap then look for Persian cuisine
Interpretation
GetRestaurants(
city="San Leandro",
cuisine="Persian",
price_range="cheap"
)
LLM

Figure 1: Example of the NLSI task. Given a user query and user-specific list of preferences, and API documentation, the model has to parse the input into structured output. The model has to (i) select, which preferences are relevant for the current query and (ii) interpret the utterance into one or several API calls. The diagram is a replica of Figure 1 from Moghe et al. (2024).

alised tool use in a dialogue setting.

Our *contributions* are: **(i)** we analyse common LLMs' performance on the personalised tool-use task and identify their current weaknesses; **(ii)** we propose structured tagging, an annotation scheme that bridges natural language and API calls by hierarchically marking functions and their arguments within user preferences, **(iii)** we introduce TAPS, a tuning-free approach that uses a structured tagging tool and an uncertainty-based tool detector to facilitate integration of user preferences into tool-augmented goal-oriented dialogue agents; **(iv)** we demonstrate that our method improves the effectiveness of LLMs on the task, achieving state-of-the-art results for open-source models on the interpretation subtask of NLSI with an increase of +16.5% in exact match (EM) and +16.9% in F1. Our findings suggest TAPS's potential for generalisation to other goal-oriented tasks, where reductions in errors such as hallucinations and missing arguments could improve system reliability and user experience. With this work, we hope to inspire future research on tool-use personalisation.

2 Task Setup

2.1 Task Definition

The NLSI task is defined as follows. Given a user query, standing instructions, and API documentation, an agent must generate up to three API calls to fulfil the user request (Figure 1). The standing instructions constitute the user profile – their preferences regarding different aspects, e.g., favourite cuisine, preferred airline, or music taste. The task requires complex reasoning to integrate query details with user preferences to generate appropriate API calls. Ultimately, the task consists of two subtasks: *selection*, identifying the subset of instructions relevant to the current query; and *interpretation*, generation of API calls to perform the user task using the user query, user profile, and API documentation.

This work focuses on the interpretation subtask, which is crucial for improving LLMs' ability to handle contextualised tool use – a key challenge in real-world applications. Successful interpretation requires an agent to understand the user intent, reason over the conversation and user profile, and identify the appropriate APIs, necessary arguments, and their values. To ensure a controlled evaluation, we provide LLMs with the correct selected standing instructions, allowing them to access only the relevant user profile information.

2.2 Evaluation

We follow the evaluation setup, described in Moghe et al. (2024) to assess model performance. We convert each API call into (function name, argument name, value) triplets, or slots, to compute the metrics and report exact match (EM), slot-wise F1, precision, and recall.

2.3 Behaviour Analysis

Model	Source	Size	Instr.-Tuned	Tools
CodeLlama	Rozière et al. (2024)	7B	✗	✗
CodeLlama-Inst		7B	✓	✗
Llama-2	Touvron et al. (2023)	7B	✗	✗
Llama-2-Chat		7B	✓	✗
Llama-3	Dubey et al. (2024)	8B	✗	✗
Llama-3-Inst		8B	✓	✗
Mistral-3	Jiang et al. (2023)	7B	✗	✓
Mistral-3-Inst		7B	✓	✓
OLMo-2-7B-Inst	OLMo et al. (2024)	7B	✓	✗
GPT4o	OpenAI et al. (2024)	unk	✓	✓

Table 1: LLMs used in our work.

The challenge of NLSI is incorporating several aspects: models must not only accurately identify the users' intended task but also select relevant information from both the current user query and the user profile, and effectively utilise it to generate the appropriate API call. An additional complexity arises from the limited availability of training data,

Model	EM	F1	Prec.	Rec.
CodeLlama	16.3	55.8	66.9	49.5
CodeLlama-Inst	18.1	57.0	68.3	49.7
Llama-2	10.3	51.0	51.3	52.0
Llama-2-Chat	10.3	45.6	53.2	41.7
Llama-3	10.1	52.2	47.5	69.3
Llama-3-Inst	32.5	70.3	68.5	77.97
Mistral-3	9.7	54.4	50.1	66.7
Mistral-3-Inst	32.7	65.5	67.6	65.5
OLMo-2-7B-Inst	10.8	43.0	44.6	46.4
GPT4o	**50.4**	**84.4**	**84.4**	**87.2**

Table 2: Comparison of baseline models on the NLSI test set. **EM**: exact match. **F1**: Slot-wise F1 score. **Prec.**: precision. **Rec.**: recall. All scores are in %. Best performance is in **bold**, second best is underlined.

which significantly constrains our ability to use learnable methods to solve this task.

Moghe et al. (2024) evaluate various language models on NLSI but focus on a simple in-context learning (ICL) setting. We extend this analysis by investigating the behaviour of common LMs, summarised in Table 1. Our experiments prioritise 7B/8B models to balance efficiency in low-resource settings and latency – critical factors for interactive task assistants – while recognising that larger models do not universally yield proportional performance gains despite their significantly higher resource demands. We compare our approach to GPT4o (gpt-4o-2024-08-06), a significantly larger model, to assess capability and computational cost trade-offs. We follow Moghe et al.'s evaluation setup, using their prompt in 1-shot setting (see Appendix F) and report results in Table 2.

2.3.1 Model Comparison

Effect of Model Size GPT4o demonstrates the highest scores across all evaluated metrics, suggesting some innate ability to infer API calls from user queries given their preferences. All smaller open-source models underperform significantly, highlighting the need for better and more effective interpretation techniques.

Pre-Training and Post-Training Effects A comparison of instruction-tuned models with their base counterparts shows that instruction fine-tuning can offer modest performance gains. However, the inferior performance of the instruction-optimised Llama-2-Chat relative to its base version indicates that instruction fine-tuning does not universally result in improvements and may sometimes impede performance. Notably, we did not optimise the prompts for each model, which could affect model performance and lead to sub-optimal results. The significant drop in the scores of CodeLlama and Llama-2 models compared to others implies that optimising LLMs for tool use enhances their ability to handle complex interpretation tasks, allowing them to better integrate various input sources and produce accurate function calls.

The substantial gap between the EM and F1 scores across all models shows that while they can produce plausible API calls, they struggle to accurately incorporate all necessary data when translating natural language into executable code. Given the lower scores of some models, we focus on Mistral-3-Inst, Llama-3-Inst, and GPT4o in our further experiments.

2.3.2 Effect of Example Complexity

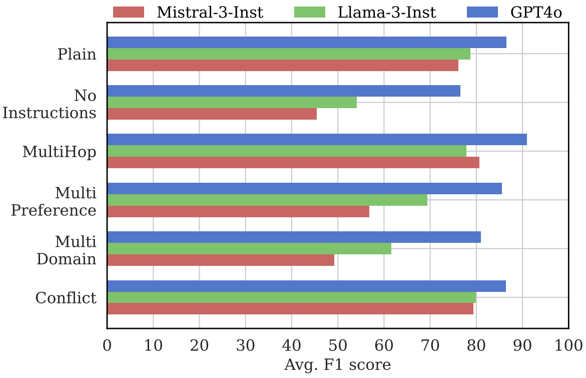


Figure 2: Average F1 scores of baseline models per each reasoning type. All scores are in %.

NLSI includes examples of varying difficulty based on the reasoning required to incorporate the standing instructions into the response (see Section 3.1. in Moghe et al. (2024) for a detailed description of the types). Figure 2 demonstrates that while GPT4o is able to consistently score above 75% F1 on all reasoning types, open-source models fall behind. Both Mistral-3-Inst and Llama-3-Inst can effectively follow simple, straightforward standing instructions where each argument of the final API call directly corresponds to one instruction (Plain, Conflict), suggesting some capability to solve the task. However, they struggle with cases that require reasoning across multiple domains (MultiDomain) or incorporating multiple preferences (MultiPreference). All models achieve lower scores when no instructions are provided (NoInstructions).

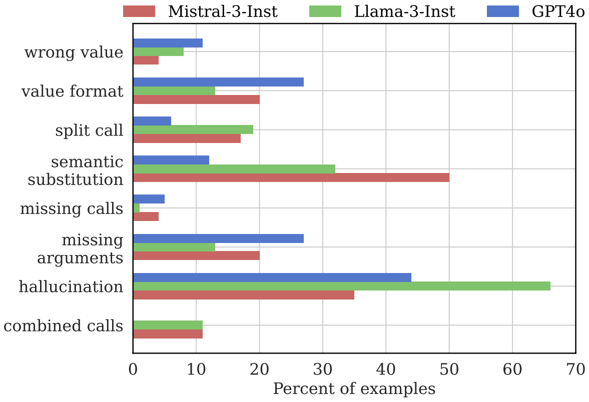


Figure 3: Distribution of errors on a sample of baseline predictions.

2.3.3 Qualitative Analysis

Similarly to Moghe et al. (2024), we manually annotate a sample of 100 predictions for each model and perform their qualitative analysis. We classify the errors into several categories (Table 9 in Appendix D) and present the results in Figure 3.

Open-source models frequently confuse semantically similar function and argument names (particularly `Mistral-3-Inst`, where the error is persistent on 50% of the examples). This results in semantic substitution errors, where predictions are correct in meaning but deviate from documentation (e.g., using argument `city` from `GetRestaurants` instead of expected `location` in `GetTravel`). 35-75% of examples include hallucinations, making it the most common error type for `Llama-3-Inst` and `GPT4o`. Hallucinations primarily involve the generation of extra arguments and the creation of new functions. We also observe value formatting issues, ranging from extracting part of the correct entity to canonicalisation issues, when models incorrectly unify date and time formats, which is common for `GPT4o` (over 25%). Often, LLMs ignore available information, missing one or several arguments, especially on examples requiring multi-hop reasoning (MultiDomain, MultiPreference). However, this happens in simpler cases as well (Plain, Conflict), where the models tend to favour one information source (the user query or instructions), leading to incomplete API calls.

Overall, our findings support Moghe et al. (2024). These results underline the task's inherent complexity and demonstrate that current LLMs cannot solve it on their own, highlighting the need for specialised methods to overcome this challenge.

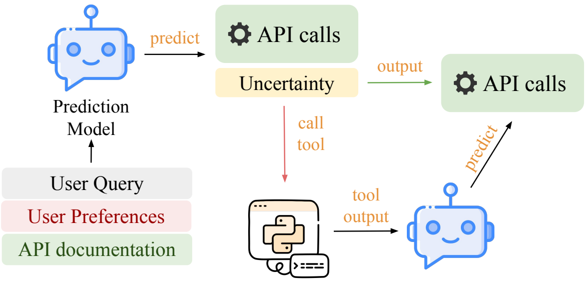


Figure 4: TAPS pipeline. An LLM first generates a response to the user query, and model uncertainty is extracted from its logits. Based on the uncertainty score, TAPS either accepts the response as is, or calls a structured tagging tool to augment the data before passing it back to the LLM and regenerating the answer.

3 TAPS

In this work, we aim to address key limitations of LLMs in personalised tool use, including semantic substitution errors, hallucinations, and missing arguments. We propose TAPS, a fully automated approach for goal-oriented dialogue that (i) employs a structured tagging tool for data augmentation and (ii) independently determines when tool use is required (iii) without additional training. Figure 4 illustrates the full pipeline of TAPS, which we outline below.

3.1 Structured Tagging Tool

We define a data augmentation tool that introduces an intermediate representation between the natural language input and the function calls by annotating standing instructions with structured tags that encode action-level and slot-level information (Figure 5). Specifically, we label each instruction with hierarchical tags, where high-level action tags denote the relevant API and nested slot tags capture the arguments and their values. We call this approach **structured tagging**. Unlike traditional Named Entity Recognition or semantic parsing, which converts natural language to a structured representation, our method preserves the natural language aspect of instructions while introducing explicit nested tags, allowing models to leverage both the original instruction phrasing and explicit structural information. We hypothesise that adding this intermediate representation before code generation will facilitate more accurate API argument extraction and prevent information loss when generating API calls.

Original:

If I'm looking for Events, I'd like them to be in New York.

Augmented:

<a:GET_EVENTS> If I'm looking for Events, I'd like them to be in <sl:CITY> New York </sl> . </a>

Figure 5: Example of structured tagging in TAPS. We use <a:API> ... </a> tags to denote relevant APIs and <sl:ARGUMENT> ... </sl> to label arguments and their values.

Config Name	Tags	Tagger Model	Tool Detector
DEFAULT	✗	✗	✗
EXT-TAG	✓ External	Base model	✗ Naïve
EXT-TAG$_{\text{OPT}}$	✓External	GPT4o	✗ Naïve
JOINT-TAG	✓ Joint	Base model	✗ Naïve
TAPS	✓ External	Base model	✓ Uncertainty
TAPS$_{\text{OPT}}$	✓ External	GPT4o	✓ Uncertainty
TAPS-ORACLE	✓ External	Base model	✓ Oracle
TAPS-ORACLE$_{\text{OPT}}$	✓ External	GPT4o	✓ Oracle

Table 3: Model configurations used in experiments.

Additionally, we explore two versions of the tool:

- **External Tagger** (EXT-TAG): Relies on an external model for tagging, allowing us to use specialised models with improved tagging accuracy. To isolate the effect of tagging quality, we use the same model for both tagging and subsequent API generation. Additionally, we include results where GPT4o is used as an example of an optimal tagger (EXT-TAG$_{\text{OPT}}$) to demonstrate how variations in tag quality influence overall task results (see Appendix B for tagger comparison).
- **TAG-AND-GENERATE** (**JOINT-TAG**): We ask the same base model to generate the augmentation for the standing instructions and the final API call jointly. This strategy allows us to rely on the internal reasoning abilities of an LLM, hypothetically making it easier for it to effectively use the provided information and predict the final answer.

3.2 When to use a tool?

Deciding when a tool is necessary is a challenging task. Recent approaches address tool detection through either an external learned classifier (Gemmell and Dalton, 2023) or reinforcement learning (Qiao et al., 2024). Given the limited availability of training data for our task, we cannot rely on trainable methods. Thus, we propose to utilise model uncertainty to assess the confidence of an LLM in its prediction and determine whether additional help is needed to solve the task.

We explore three methods for uncertainty estimation commonly used in text generation:

- **Sequence Margin**: the difference in the probability scores of the top two most likely predictions;
- **Margin@T**: the difference in the probability scores of the top T most likely tokens, where T is a hyper-parameter;
- **Least Confidence**: the difference between the probability of the top most confident prediction and 100% confidence. The lower the score, the more certain the model is in its prediction.

To choose the most effective method, we use the Pearson correlation coefficient (Freedman et al., 2007) between the uncertainty of the model and the downstream task F1 metric on the validation set and report the results in Table 8 (Appendix C).

Among the tested approaches, Least Confidence performs best, with a moderate correlation score (circa -0.45 for all models), suggesting that higher uncertainty indicates lower target scores. Other methods fail to provide reliable confidence estimates. Both weakly correlate with F1, making a comparison of top-2 most likely predictions, on sequence or token-level, unreliable. Thus, we choose Least Confidence as the main tool-use detector in TAPS.

To effectively utilise the uncertainty score, we select a threshold value on the validation set. The threshold is used to determine the confidence level of the model, based on which we choose to employ one of the following strategies: (i) output the model answer, or (ii) use a tool and regenerate the answer.

4 Results & Discussion

In this section, we first investigate the effectiveness of TAPS's data augmentation tool on the NLSI task (Section 4.1). Second, we illustrate the importance of tool detection and evaluate TAPS on the test subset in NLSI (Section 4.2). Finally, we perform behavioural analysis of TAPS's predictions when both structural tagging and tool detection are utilised to demonstrate the impact of the approach (Section 4.3).

4.1 Effects of Structured Tagging

To show the effectiveness of structured tagging, we compare the performance of both tagging tools to default models without tools. We naïvely apply

Model	Aug.	EM ↑	F1 ↑	Prec. ↑	Rec. ↑
Llama-3-Inst	Default	42.23	78.19	80.30	78.60
	Ext-Tag	44.22	80.34	81.90	81.49
	Ext-Tag$_{\text{OPT}}$	**51.79**	**84.46**	**86.39**	**84.86**
	Joint-Tag	41.43	78.31	82.97	77.19
Mistral-3-Inst	Default	30.68	64.21	65.21	65.37
	Ext-Tag	36.65	75.12	78.33	74.30
	Ext-Tag$_{\text{OPT}}$	**42.63**	**79.34**	**82.23**	**79.04**
	Joint-Tag	33.47	66.66	70.56	64.97
GPT4o	Default	56.18	87.40	**90.41**	**86.83**
	Ext-Tag	**57.37**	**87.47**	89.63	86.72
	Joint-Tag	52.99	83.94	86.00	83.24

Table 4: Model performance with and without naïve tool-use. Ext-Tag: the same model is used for tagging and API call generation sequentially; Ext-Tag$_{\text{OPT}}$: tagging is performed by a separate, high-performing tagger; Joint-Tag: tags and API calls are generated jointly in a single step. **EM**: exact match. **F1**: Slot-wise F1 score. **Prec.**: precision. **Rec.**: recall. All scores are in %. Best performance is in **bold**, second best is underlined.

the tool to all instances in the validation set. Here and in further experiments, we use ICL to evaluate the models and optimise model performance by bootstrapping a set of demonstrations with random search (Khattab et al., 2023). Table 3 summarises all model configurations used in our experiments. Full implementation details are in Appendix A.

We report the results in Table 4. We observe marginal gains in GPT-4o when using Ext-Tag, and consistent improvements across all four metrics for open-source models: Llama-3-Inst and Mistral-3-Inst improve EM by 2% and 6%, respectively, and up to 11.9% when the optimal model is used for tagging Ext-Tag$_{\text{OPT}}$).

We further investigate the impact of tool use on model outputs and calculate the percentage of predictions that improve or degrade after structured tagging is applied (Table 5). Overall, all models benefit from tool use in less than 50% of cases, with open-source models gaining the most. Only 16.3% of predictions improve for GPT4o, which is least affected by tagging, with more than 62% of predictions remaining the same with and without the tags, compared to 37% for both open-source models. In 16-27% of cases, LLMs score lower when having the tags. Below, we discuss our key findings regarding structural tagging effects.

LLMs struggle to map natural language to code. The inferior performance of all Default models compared to Ext-Tag suggests that LLMs still need additional tools to successfully generate code from natural language when complex reasoning is required. Strong results of Ext-Tag, even with lower quality tags, support our hypothesis that introducing an intermediate representation between natural language and code can significantly enhance model performance. Notably, tagging is less effective for GPT4o. We hypothesise that this can be due to the GPT4o's stronger in-context reasoning, making task decomposition less beneficial, compared to smaller models that generally do not perform complex tasks as well. The improvements of Ext-Tag$_{\text{OPT}}$ over Ext-Tag show that the effectiveness of the proposed approach is closely tied to the reliability of structured tags. Initial robustness experiments confirm this sensitivity, and we further investigate the impact of tag quality in Section 5.

Result	Llama-3-Inst	Mistral-3-Inst	GPT4o
Win ↑	35.1	45.8	16.3
Same	37.0	37.9	62.2
Loss ↓	27.9	16.3	21.5

Table 5: Data augmentation effects for Ext-Tag$_{\text{OPT}}$. All scores represent % of instances. All calculations are based on F1.

Internal reasoning does not boost the interpretational abilities of LLMs. Our results demonstrate that explicitly prompting the models to generate structured tags before producing the function calls is not uniformly effective. The observed decrease in recall suggests that this approach may result in some information loss. While LLMs generate more accurate code, they tend to omit more arguments, showing that solving the task end-to-end is difficult. An additional explanation for such behaviour is the demonstration optimisation strategy we use. Existing ICL optimisation approaches do not support optimisation for multiple outputs, leading to suboptimal model performance.

Naïve tool use fails to yield consistent improvements. We show that naïvely leveraging the tool is inefficient, both in compute and target metrics and sometimes even counterproductive. This highlights the importance of tool detection to determine if a tool is required on the instance level.

4.2 Tool Detection Effects

We evaluate TAPS on the NLSI test set and report the results in Table 6. We compare the scores with lower-boundary baselines, default models without tools and naïve tool use (Ext-Tag),

Model	Config	EM ↑	F1 ↑	Prec. ↑	Rec. ↑
Llama-3-Inst	DEFAULT	41.76	78.26	82.96	76.80
	EXT-TAG	47.55	82.28	84.89	81.88
	TAPS	51.18	83.94	87.20	82.95
	TAPS-ORACLE	**59.07**	**88.42**	**91.60**	**87.11**
	EXT-TAG$_{\text{OPT}}$	51.23	84.51	87.23	83.86
	TAPS $_{\text{OPT}}$	53.04	85.64	88.67	84.56
	TAPS-ORACLE$_{\text{OPT}}$	**59.85**	**89.65**	**92.82**	**88.10**
Mistral-3-Inst	DEFAULT	35.74	69.11	70.64	69.83
	EXT-TAG	40.29	76.20	79.53	75.57
	TAPS	42.40	76.76	79.74	76.32
	TAPS-ORACLE	**48.82**	**81.79**	**84.29**	**81.33**
	EXT-TAG$_{\text{OPT}}$	42.35	78.55	82.63	77.24
	TAPS $_{\text{OPT}}$	44.17	79.03	82.66	78.04
	TAPS-ORACLE$_{\text{OPT}}$	**49.85**	**83.19**	**86.19**	**82.36**
GPT4o	DEFAULT	56.32	86.99	89.25	86.91
	EXT-TAG	55.54	86.49	88.78	85.65
	TAPS	58.63	87.86	90.03	87.21
	TAPS-ORACLE	**65.88**	**91.46**	**93.57**	**90.49**

Table 6: Model performance on test data. OPT: best performing model is used for tagging. **EM**: exact match. **F1**: Slot-wise F1 score. **Prec.**: precision. **Rec.**: recall. All scores are in %. Best performance is in **bold**, second best is underlined.

and upper-boundary oracle models optimised for tool detection. The oracle prediction is compiled by retrospectively selecting the examples that actively benefit from tool use and leaving other predictions unchanged.

Overall, we find that for open-source models, both naïve tool use and TAPS are superior to base models without tools by a margin with EM and F1 gains of up to 10% when an optimal tagger is employed. Using a tool detector significantly improves target metrics compared to naïve tool use, with TAPS and TAPS-ORACLE outperforming EXT-TAG by 2/8% EM, respectively. The results for GPT4o are less consistent, however, they illustrate the same idea. While EXT-TAG leads to model score degradation, leveraging a tool detector improves model effectiveness by 2/9% EM. Although optimal tags significantly outperform lower-quality ones in the naïve setting, tool detection narrows the gap to within 1% EM, demonstrating that the approach is effective even when a lightweight model is used for all steps of the pipeline. We highlight our key findings below.

Using a tool detector can maximise tool use effectiveness. We find that both TAPS and TAPS-ORACLE outperform all baseline models, demonstrating that selectively using tools is much more effective than relying on them at all times. Moreover, our experiments show that tool detection allows us to minimise both time and compute spent on the task by applying the tool 20% fewer times for open-source models and over 55% fewer times for GPT4o when using uncertainty, and up to 80% in the oracle case. This is particularly valuable, as achieving an optimal balance between latency and model capabilities is crucial for task assistants interacting with users in real time.

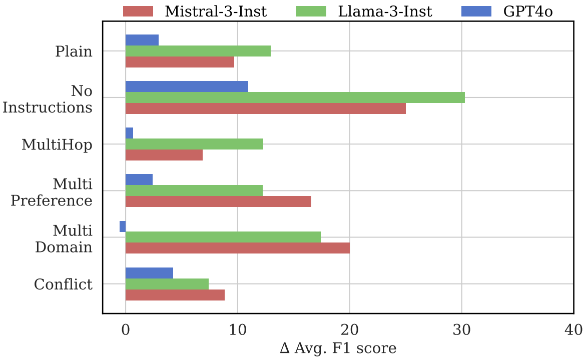


Figure 6: Δ F1 scores of TAPS models compared to baselines per each reasoning type.

Using uncertainty for tool detection is possible but suboptimal While we demonstrate that utilising uncertainty for tool detection can be beneficial, we note the suboptimal performance of TAPS compared to the oracle model. TAPS-ORACLE is consistently superior to TAPS for all models, with performance gains of 5.7-7.9% w.r.t. EM scores. The same trend is observed in terms of resource efficiency. This indicates that uncertainty may not be the most effective approach to determine whether calling a tool would yield higher scores, and alternative methods may be explored in future.

4.3 Prediction analysis

Figure 6 demonstrates the difference in F1 scores of baseline models (Section 2.3) and TAPS (for avg. F1 scores refer to Appendix E). We observe consistent improvements in scores or on-par performance when using TAPS on all reasoning types. An external tool for tagging increases the performance by up to 30% (Llama-3-Inst) on the task, with an average improvement on each reasoning type by 3-15% depending on the model.

We sample and manually annotate the same 100 examples for each model as in Section 2.3.3 and compare the percentage of errors. Figure 7 presents the results of the comparison. The biggest difference is observed on hallucinations (19-49% less errors) and semantic substitution errors (4-34% decrease), which we specifically targeted with our

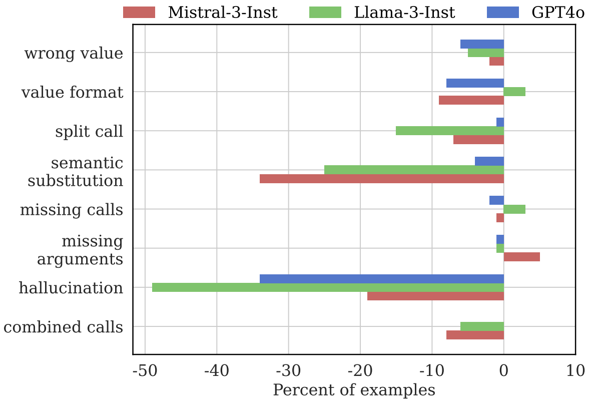


Figure 7: Changes in the distribution of errors in TAPS compared to baseline models. The scores represent the percentage of examples that improved or degraded with TAPS.

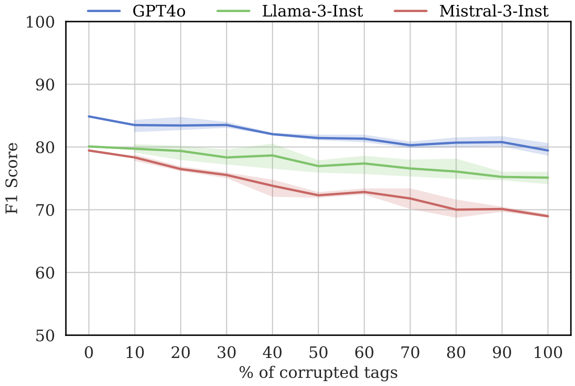


Figure 8: The effects of tag quality on the F1 scores of GPT4o, Llama-3-Inst, and Mistral-3-Inst. All scores are in %.

approach. However, we also notice slight increases in some error types for some models, which can be due to error propagation since we are using GPT4o as a tagger model. For example, Llama-3-Inst exhibits more value formatting issues when using a data augmentation tool, one of the common errors of GPT4o, according to our baseline evaluation (Figure 3). Additionally, we notice an increase in missing arguments for Mistral-3-Inst, specifically when shared contextual information (e.g. location) is available. We attribute this to our tagging approach, which does not allow us to incorporate the links between instructions, leading to the exclusion of some possible annotations. We discuss this limitation in more detail in Section 7. Overall, we show that using TAPS significantly decreases the number of errors for all models, proving it an effective solution for tool use personalisation.

5 Tagging Sensitivity Analysis

While our main experiments (Section 4) demonstrate that structured tagging substantially enhances the ability of LLMs to incorporate user preferences in tool calling, they also reveal that the effectiveness of this approach varies with the quality of the tags. This raises a question of the robustness of LLMs to imperfect and noisy tags. To address this, we conduct a controlled corruption study where we systematically perturb the tags provided to the model. Starting from golden tags, manually annotated by one of the authors, we randomly corrupt $n\%$ of them, with $n \in \{0, 10, ..., 100\}$. Possible tag corruptions are sampled from error types that frequently occur in real tagger outputs, including slot deletion (replicating missing arguments error), tag boundary shifts, and semantic substitution of slot and function names. The corrupted tags are then passed to the EXT-TAG pipeline, and the final model predictions are evaluated using the same setup as in the main experiments. This allows us to quantify the sensitivity of different models to varying tag quality and to identify which models are more robust to noisy annotations.

Figure 8 presents the results of the study with respect to the F1 scores, other metrics follow the same trend (see Figure 11). Across all models, the scores degrade monotonically as the percentage of corrupted tags increases. This confirms that the noise in the structured tags directly impacts the downstream performance of LLMs. However, the sensitivity to tag quality differs between the models. GPT4o and Llama-3-Inst remain relatively robust, with the tag quality degrading by up to 5%. Mistral-3-Inst exhibits a steeper decline, with F1 scores dropping more than 10% if comparing the 100% corruption rate and golden tags. These results align well with our findings in Section 4.1, showing that higher-quality tags (marked with OPT) yield more drastic improvements compared to suboptimal ones. Relative to the no-tag baseline (DEFAULT, Table 4), we find that structured tagging improves models' scores when tag quality is sufficiently high, but harms them beyond a certain level of corruption, showing that overly noisy tags can reduce downstream effectiveness.

Overall, this analysis shows that the effectiveness of EXT-TAG directly depends on the quality of the produced tags. Together with the TAPS results, these findings demonstrate both the potential and the limitations of leveraging structured tags. While they can significantly improve the ability of

LLMs to produce accurate personalised tool calls, their effectiveness is bound by upstream tag quality and model robustness. This suggests two promising directions for future work: (i) developing more accurate taggers, and (ii) designing approaches that explicitly account for and compensate for noisy intermediate tags. More broadly, these results highlight the need for better entity understanding in LLMs, which remains one of the key bottlenecks for robust personalised tool use.

6 Related Work

Tool-Augmented Language Models Introduction of tool-augmented LLMs have enabled general agents to perform a variety of diverse tasks (Parisi et al., 2022; Patil et al., 2023; Mialon et al., 2023). A body of work on tool use leverages the innate abilities of LLMs to produce structured data from natural language input (Song et al., 2023; Liu et al., 2023, 2024b; Zhang et al., 2024). For example, Hsieh et al. (2023) show that tool documentation alone is sufficient to elicit tool use in LLMs without demonstrations. Some use task decomposition (Wu et al., 2024) and a backward reasoning pipeline (Zhang et al., 2024) to generate appropriate parameter values effectively. Other works incorporate tuning-based approaches (Parisi et al., 2022; Schick et al., 2023; Patil et al., 2023; Mekala et al., 2024; Shen et al., 2024), with Shi et al. (2024) iteratively predicting and filtering tool-usage plans, and Qiao et al. (2024) leveraging reinforcement learning with tool execution feedback for consistent tool invocation. Hao et al. (2023) train tool embeddings, while Shen et al. (2024) propose a two-stage fine-tuning technique with join training and separate refinement of specialised modules for each subtask in tool-use paradigm. Despite their effectiveness, existing TALMs still face challenges in personalising interactions and efficiently integrating tool use with conversational history.

Personalisation Personalisation is an important aspect of any system interacting with users. Many works on personalisation for dialogue provide models with user profiles, describing their preferences and personality traits through natural language statements (Li et al., 2016; Zhang et al., 2018; Majumder et al., 2020) or structured databases (Song et al. 2020; Aliannejadi et al. 2024, among others). Cheng et al. (2024) propose to learn user preferences from dialogue history. Nevertheless, these works focus on creating a user persona for more engaging conversations rather than task completion. Joshi et al. (2017) introduce simple structured user profiles for a limited number of goal-oriented dialogue tasks and explore rule-based systems and memory networks. To the best of our knowledge, Moghe et al. (2024) is one of the only approaches that attempts to personalise goal-oriented dialogue through explicit and complex user preferences in natural language. However, the work explores only simple ICL approaches for the task. Our work attempts to solve the task by leveraging tool use and an internal tool detection mechanism that provides more flexibility and robustness in tailoring tool use according to user preferences.

7 Conclusion and Future Work

In this work, we explore the limitations of LLMs to perform the personalised tool use task. We find that all LLMs struggle to effectively incorporate user preferences, especially when complex reasoning is required, suffering from semantic errors, information loss and hallucinations. To combat this, we propose TAPS, a tuning-free solution for personalised tool use in task assistants. TAPS combines (i) a structural tagging tool that introduces an intermediate representation between natural language and code and (ii) an internal tool detector to facilitate the incorporation of user preferences for tool use in goal-oriented dialogue. We conduct a thorough analysis of widely used LLMs on the NLSI dataset and demonstrate that our method consistently outperforms pre-trained open-source models of the same size. We show that TAPS enables the models to more effectively reason and infer tool calls from user queries and successfully incorporate information from personalised user preferences, all while being fully automatic and not requiring additional training. Through ablation studies, we show that each component in TAPS plays an important role in the solution of the task, significantly minimising most error types for tested LLMs. We hope our work will inspire more research on incorporating extended context in tool use in future.

Limitations

A better structural tagger is required. One of the limitations of our solution lies in the tagging approach we employ, which has several shortcomings. First, as briefly mentioned in Section 4.3, we label APIs and arguments on the sentence level only and do not consider the whole user profile. This leads to the loss of shared contextual information, which should be included in all relevant API calls but is tagged as belonging to only one API. Second, we apply the tool only to the user profile, which might lead to some information loss, as we do not explicitly label the relevant information from user queries, prompting the model to prioritise user profiles over queries. Lastly, in our experiments, we use ICL and prompting, while training a specialised model for tagging might yield better and more reliable results. A more sophisticated tagging procedure will help mitigate those issues, and we hope to continue working in this direction in future.

LLMs are not robust to changes in input. We utilise LLMs' in-context learning abilities to create a solution for the task. Such an approach is less computationally expensive, as it does not require additional training and allows for generalisation to unseen domains, functions and tasks. However, we do not address a well-known shortcoming of ICL, namely its sensitivity to prompt template choice and demonstration selection (Lu et al., 2022; Chang and Jia, 2023; Sclar et al., 2024). While we explore several prompts in our preliminary studies and utilise demonstration optimisation, we do not conduct extensive experimentation on the topic as it is not the primary focus of our work. This means that the prompts used to evaluate TAPS may not be optimal for the task. While training a specialised model for the task would seem like a logical solution, the dataset size is insufficient for straightforward fine-tuning and requires a different approach. For example, LIMA (Zhou et al., 2024) or similar methods can be used to fine-tune a model on low data cases.

The need for a better evaluation benchmark. In our experiments, we use the NLSI dataset, collected by Moghe et al. (2024), as the only dataset, to our knowledge, that incorporates user preferences into tool-augmented conversational agents. However, the dataset has several downsides. First, the dataset is created automatically from templates without additional validation, so it contains some errors (see Section 2.3.3) and is overall not as diverse and natural in terms of both language and domains covered. Additionally, evaluation on NLSI is based on comparing code strings rather than the actual tool output. This approach can underestimate model performance, as two different programs can lead to the same output when executed but will get different evaluation scores. Therefore, we acknowledge the need for a better evaluation methodology and benchmark for the task in order to more accurately assess and compare the capabilities of LLMs with respect to contextualised tool use.

Ethical Considerations

Privacy is a critical concern in natural language processing, especially when handling personal data (Horvitz and Mulligan, 2015; Yao et al., 2024; Miranda et al., 2025). Working with user preferences and extended dialogue history can inadvertently lead to the potential exposure of sensitive personal information. Our approach employs in-context learning, which prevents the model from memorising private information. This strategy aligns with the growing emphasis on privacy in LLMs by ensuring that user data remains protected throughout the conversation.

We improve and proofread the text of this paper using Grammarly[2] to correct grammatical, spelling, and style errors and paraphrasing sentences.

Acknowledgements

This work is supported by a Turing AI Acceleration Fellowship from the Engineering and Physical Sciences Research Council, grant number EP/V025708/1.

References

Sanchit Agarwal, Jan Jezabek, Arijit Biswas, Emre Barut, Bill Gao, and Tagyoung Chung. 2022. Building goal-oriented dialogue systems with situated visual context. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(11):13149–13151.

Mohammad Aliannejadi, Zahra Abbasiantaeb, Shubham Chatterjee, Jeffrey Dalton, and Leif Azzopardi. 2024. Trec ikat 2023: A test collection for evaluating conversational and interactive knowledge assistants. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 819–829, New York, NY, USA. Association for Computing Machinery.

Ting-Yun Chang and Robin Jia. 2023. Data curation alone can stabilize in-context learning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8123–8144, Toronto, Canada. Association for Computational Linguistics.

Chuanqi Cheng, Quan Tu, Wei Wu, Shuo Shang, Cunli Mao, Zhengtao Yu, and Rui Yan. 2024. "in-dialogues

[2] grammarly.com

we learn": Towards personalized dialogue without pre-defined profiles through in-dialogue learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10408–10422, Miami, Florida, USA. 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, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew

Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

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

David Freedman, Robert Pisani, and Roger Purves. 2007. Statistics (international student edition). *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*.

Carlos Gemmell and Jeff Dalton. 2023. ToolWriter: Question specific tool synthesis for tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 16137–16148, Singapore. Association for Computational Linguistics.

Rahul Goel, Shachi Paul, Tagyoung Chung, Jeremie Lecomte, Arindam Mandal, and Dilek Hakkani-Tür. 2018. Flexible and scalable state tracking framework for goal-oriented dialogue systems. *ArXiv*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. In *Advances in Neural Information Processing Systems*, volume 36, pages 45870–45894. Curran Associates, Inc.

Eric Horvitz and Deirdre Mulligan. 2015. Data, privacy, and the greater good. *Science*, 349(6245):253–255.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *Preprint*, arXiv:2308.00675.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Chaitanya K Joshi, Mi Fei, and Boi Faltings. 2017. Personalization in goal-oriented dialog. In *NeurIPS Conversational AI Workshop*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *CoRR*, abs/2310.03714.

Jiwei Li, Michel Galley, Chris Brockett, Georgios Spithourakis, Jianfeng Gao, and Bill Dolan. 2016. A persona-based neural conversation model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 994–1003, Berlin, Germany. 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. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

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. 2024a. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. 2024b. Toolnet: Connecting large language models with massive tools via tool graph. *Preprint*, arXiv:2403.00839.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. 2023. Controlllm: Augment language models with tools by searching on graphs. *Preprint*, arXiv:2310.17796.

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.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098, Dublin, Ireland. Association for Computational Linguistics.

Bodhisattwa Prasad Majumder, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Julian McAuley. 2020. Like hiking? you probably enjoy nature: Persona-grounded dialog with commonsense expansions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9194–9206, Online. Association for Computational Linguistics.

Dheeraj Mekala, Jason Weston, Jack Lanchantin, Roberta Raileanu, Maria Lomeli, Jingbo Shang, and Jane Dwivedi-Yu. 2024. Toolverifier: Generalization to new tools via self-verification. *Preprint*, arXiv:2402.14158.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *Preprint*, arXiv:2311.12983.

Michele Miranda, Elena Sofia Ruzzetti, Andrea Santilli, Fabio Massimo Zanzotto, Sébastien Bratières, and Emanuele Rodolà. 2025. Preserving privacy in large language models: A survey on current threats and solutions. *Transactions on Machine Learning Research*.

Nikita Moghe, Patrick Xia, Jacob Andreas, Jason Eisner, Benjamin Van Durme, and Harsh Jhamtani. 2024. Interpreting user requests in the context of natural language standing instructions. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4043–4060, Mexico City, Mexico. Association for Computational Linguistics.

Christian Muise, Tathagata Chakraborti, Shubham Agarwal, Ondrej Bajgar, Arunima Chaudhary, Luis Alfonso Lastras-Montaño, Josef Ondrej, Miroslav Vodolán, and Charlie Wiecha. 2019. Planning for goal-oriented dialogue systems. *CoRR*, abs/1910.08137.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2024. 2 olmo 2 furious. *arXiv*.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie

Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *Preprint*, arXiv:2205.12255.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *Preprint*, arXiv:2305.15334.

Shuofei Qiao, Honghao Gui, Chengfei Lv, Qianghuai Jia, Huajun Chen, and Ningyu Zhang. 2024. Making language models better tool learners with execution feedback. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3550–3568, Mexico City, Mexico. Association for Computational Linguistics.

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. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *Preprint*, arXiv:2307.16789.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Preprint*, arXiv:2302.04761.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. *Preprint*, arXiv:2310.11324.

Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. 2024. Small llms are weak tool learners: A multi-llm agent. *Preprint*, arXiv:2401.07324.

Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. 2024. Learning to use tools via cooperative and interactive agents. *Preprint*, arXiv:2403.03031.

Haoyu Song, Yan Wang, Wei-Nan Zhang, Zhengyu Zhao, Ting Liu, and Xiaojiang Liu. 2020. Profile consistency identification for open-domain dialogue agents. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6651–6662, Online. Association for Computational Linguistics.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world restful apis. *Preprint*, arXiv:2306.06624.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. *Preprint*, arXiv:2405.08355.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2):100211.

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018. Personalizing dialogue agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2204–2213, Melbourne, Australia. Association for Computational Linguistics.

Yinger Zhang, Hui Cai, Xierui Song, Yicheng Chen, Rui Sun, and Jing Zheng. 2024. Reverse chain: A generic-rule for LLMs to master multi-API planning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 302–325, Mexico City, Mexico. Association for Computational Linguistics.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2024. Lima: less is more for alignment. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

A Experiment Details

A.1 Dataset Statistics

We run all of the experiments of NLSI (Moghe et al., 2024), which has a train/validation/test splits of sizes 150/251/2040 instances. We refer you to the original paper for full details on the data.

A.2 Baseline Evaluation (Section 2.3)

For baseline evaluation, we use the prompt, provided by Moghe et al. (2024) for all our models Prompt F.1. We set the number of few-shot demonstrations to 1 and use default model parameters.

A.3 Main Experimental Settings (Section 4)

Optimiser settings For all experiments in TAPS we optimise the ICL examples using BootstrapFewShotWithRandomSearch algorithm (Khattab et al., 2023). We set the following parameters to the optimiser:

- `max_bootstrapped_demos` = 1 for `GPT4o` and `Llama-3-Inst` in the JOINT-TAG setting else 5
- `max_labeled_demos` = 5
- `num_candidate_programs` = 5 (`GPT4o`) / 10 (other models)
- `num_threads` = 1
- `metric` = "exact_match"

Prompt Selection We conduct a simple prompt selection experiment on the validation set of NLSI and choose the following prompts for our main experiments with TAPS. To evaluate all LLMs in DEFAULT setting, we use Prompt F.2 for `Llama-3-Inst` and Prompt F.3 for `Mistral-3-Inst` and `GPT4o`. For EXT-TAG we select Prompt F.4 for `Llama-3-Inst` and `GPT4o` and Prompt F.5 for `Mistral-3-Inst`. All runs in JOINT-TAG configuration use Prompt F.6 as the prompt.

Generation Parameters To select the optimal generation parameters for `Mistral-3-Inst` and `Llama-3-Inst` models, we run a simple grid search on the validation set. For all our experiments we use the default set of generation parameters for `GPT4o` and the following for open-source models (when different parameters for `Mistral-3-Inst` and `Llama-3-Inst` are used, we report them with a forward-slash):

- `num_beams` = 5 / 2
- `do_sample` = True
- `temperature` = 0.85 / 0.95
- `top_k` = 50
- `top_p` = 1.0

Tool Detection Parameters We use Least Confidence as our main tool detection strategy for all the experiments. We select the threshold for each model on the validation set. The following threshold values are used: 0.02 (`Llama-3-Inst`), 0.01 (`Mistral-3-Inst`), and 0.04 (`GPT4o`).

GPU-Usage We use one 40GB A100 GPU, setting the batch size of 1. It takes approximately 1.5-5 hours to run one experiment on the whole validation set and 5-13 hours to make a full pass over the test set depending on the model and generation parameters.

B Selection of the Tagger Model

To choose the models for the EXT-TAG strategy, we manually annotate the validation subset of data and compare automatically generated tags with the golden standard. To assess the tagger models we treat the task as a standard token classification problem and calculate macro-averaged F1, precision, and recall. We use Prompt F.7 for all models to generate tags for standing instructions and set all generation parameters to default values. All models are assessed in one-shot configuration. We do not optimise the demonstrations, but use a static example created manually for all instances. The results of the evaluation are presented in Table 7.

Model	F1 ↑	Prec. ↑	Rec. ↑
`CodeLlama-Inst`	63.98	63.09	65.89
`Llama-2-Chat`	63.18	62.75	63.97
`Llama-3-Inst`	71.54	74.82	69.34
`Mistral-3-Inst`	76.72	77.04	77.12
`GPT4o`	87.18	87.00	87.44

Table 7: Tagging performance on the manually annotated validation set. **F1**: macro-average F1 score. **Prec.**: precision. **Rec.**: recall. The best result is in **bold**, second best is underlined. All scores are in %.

Results Our experiments show, that open-source LMs are still far behind `GPT4o` when it comes to their ability to augment input with tags. While `GPT4o` scores exceed 86%, the difference between the best-performing open-source LLM

(Mistral-3-Inst) and GPT4o reaches 10%. Despite being the only model trained specifically to handle code and structured data, CodeLlama-Inst yields one of the lowest scores on the task with F1 of 63%. Despite GPT4o outperforming all open-source LLMs in the task, its performance is still does not exceed 90%, leaving room for improvement. We acknowledge this but continue to use GPT4o as our main external tagger model for EXTTAG.

C Uncertainty Estimation

Method	**Statistic**
Least Confidence	***-0.452***
Margin@1	0.145
Margin@2	*0.317*
Margin@3	*0.314*
Margin@4	*0.295*
Margin@5	*0.301*
Margin@6	*0.242*
Margin@7	*0.263*
Margin@8	*0.256*
Margin@9	*0.242*
Margin@10	*0.236*
Sequence Margin	*0.281*

Table 8: Pearson Correlation Coefficient between F1 scores and model uncertainty for Mistral-3-Inst. Statistics with $p < 0.001$ are in *italics*. The value in **bold** indicates the best result. Note, that negative correlation on the least confidence strategy is expected, since it represents model confidence rather that uncertainty.

D Error Types Examples

Error Type	User Query	Standing Instructions	Target	Prediction
Semantic Substitution	User: I want to find an apartment in Hayward.	> Request a home with one bed.	`GetHomes(area="Hayward", number_of_beds="1")`	`GetHomes(city="Hayward", number_of_beds=1)`
Missing Argument	User: I am looking for an Gynecologist in San Jose.	> Name Anjali Tate, M.D. as my preferred doctor when requesting a doctor.	`GetDoctors(city="San Jose", doctor_name="Anjali Tate, M.D.", type="Gynecologist")`	`GetDoctors(city="San Jose", doctor_name="Anjali Tate, M.D.")`

Hallucination (new func.)	User: Can you let me know some attractions to visit?	> If I'm looking to travel, my go-to spot is Chicago.
		> If I'm looking into Travel, I should also check out Hotels.
		> Request Hotels with a two-star rating.
			`GetTravel(location="CHi-town")`
			`GetHotels(average_rating="2", location="CHi-town")`
				`GetAttractions(city="Chicago")`
				`GetHotels(rating="2")`

Hallucination (mixed calls)	User: Can you show some attractions to visit?
	Agent: Sure. Where should I search for attractions in?
	User: Find me something in Sydney, NSW please.
		> I prefer the Museum category when requesting Travel.
		> Choose a museum if you wish to have a good experience with children.
		> I would like to request Travel for my preferred category of Park.
			`GetTravel(location="Sydney, NSW", category="Museum", good_for_kids="True")`
			`GetTravel(location="Sydney, NSW", category="Park")`
				`GetTravel(city="Sydney, NSW", category="Museum")`
				`GetTravel(city="Sydney, NSW", category="Park", good_for_kids=True)`

Combined Calls	User: I'm looking for Music events.	> If I'm looking for events, I'd like to check out what's going on in Portland.
		> If I ask for Events, my preferred category is Blues or basketball .
			`GetEvents(category="Blues", city="Portland", event_type="Music")`
			`GetEvents(category="basketball", city="Portland", event_type="Music")`
				`GetEvents(city="Portland", genre=["Blues", "Basketball"])`

Split Call	User: I want to find a new restaurant. What do you suggest to eat in San Francisco?	> Request Restaurants that serve Oriental cuisine.
		> Search for the 8 Immortals Restaurant when looking for an Oriental restaurant.
			`GetRestaurants(city="San Francisco", cuisine="Oriental", restaurant_name="8 Immortals Restaurant")`	`GetRestaurants(city="San Francisco", cuisine="Oriental")`
				`GetRestaurants(city="San Francisco", restaurant_name="8 Immortals")`

Wrong Value	User: Can you help me find some movies to watch online?	> Request funny Media.	`GetMedia(genre="funny")`	`GetMedia(genre="Comedy")`
Value Formatting	User: I would like to rent a car from March 8th in Paris, France. Agent: At what time would you need it? And when is your return date? User: I would need it at 12 o'clock in the afternoon till the 9th of this month .		`GetRentalCars(dropoff_date="9th of this month", pickup_date="March 8th", pickup_time="12 o'clock", ...`	`GetRentalCars(pickup_time="12:00", pickup_date="2023-03-08", dropoff_date="2023-03-09", ...`

Missing call	User: I need to find a General Practitioner doctor in San Jose.	> Request Access Health as your doctor.
		> If I ask for Doctor, my preferred doctor name is Daisy Manuel-Arguelles, DO .
			`GetDoctors(city="San Jose", doctor_name="Access Health", type="General Practitioner")`
			`GetDoctors(city="San Jose", doctor_name="Daisy Manuel-Arguelles, DO", type="General Practitioner")`
				`GetDoctors(city="San Jose", type="General Practitioner", doctor_name="Daisy Manuel-Arguelles, DO").`

Dataset Error	User: I'm trying to find things to do. I'd like something in New York City. I like Electronica events and I'm looking for a Concert.
	Agent: I found 3 events for you. One event is Crooked Colours at Rough Trade NYC.
	User: Sure, that works for me. I'd like to find a room in a hotel there.
			`GetEvents(category="Electronica", city="New York City", event_name="Crooked Colours", event_type="Music")`	`GetEvents(city="New York City", event_type="Concert", genre="Electronica")`
				`GetHotels(city="New York City", location="Rough Trade NYC")`

Table 9: Examples of most prominent errors made by Mistral 3. Incorrectly predicted functions, arguments and values are marked in . Missing arguments and API calls are in blue. Relevant parts of the user query and standing instructions are highlighted .

E Additional Results

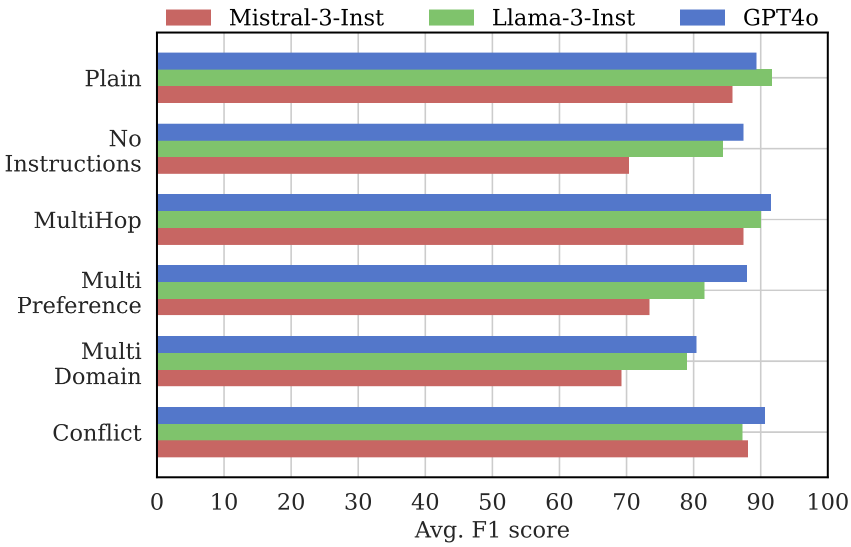


Figure 9: Average F1 scores of TAPS models per each reasoning type.

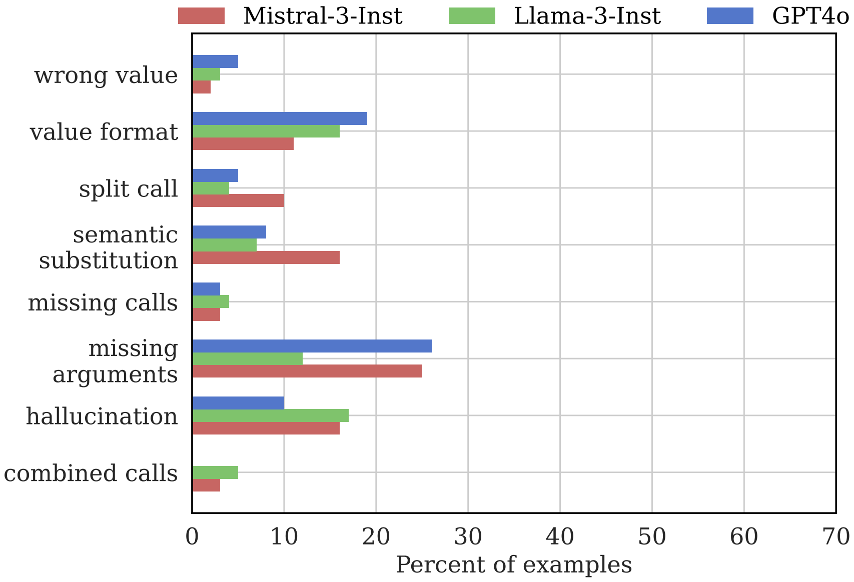
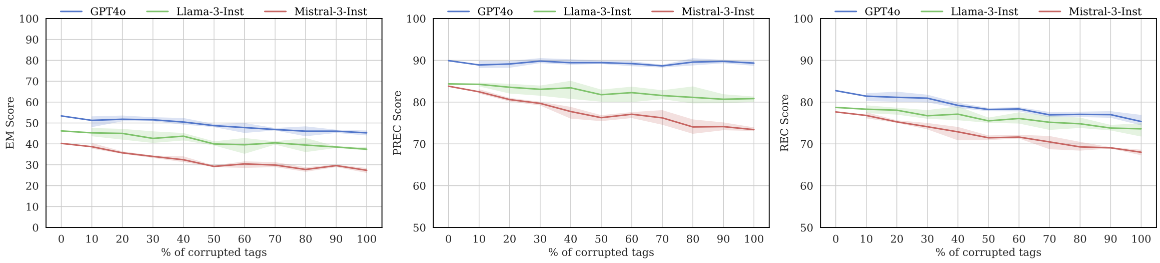


Figure 10: Distribution of errors on a sample of TAPS's predictions.



(a) EM

(b) Precision

(c) Recall

Figure 11: The effects of tag quality on the downstream task scores of GPT4o, Llama-3-Inst, and Mistral-3-Inst. All scores are in %.

F Prompts

All of the prompts we use follow the same structure: Task Description + API Schema + Input Description (optionally) + Example(s) . We provide the list of prompts below.

F.1 Baseline Prompt

System prompt template:

```
You are designing a parser that takes in a user utterance and some standing instructions and outputs a set of API calls.
Every API call consists of "GetX" where X is domain name and uses slot names listed below as arguments. We list the domain name followed by the list of possible slot names. Some slot names can be categorical or boolean
The values for the arguments can come from the user’s dialogue or standing instructions. If the user requests a slot name and no value is found, use "?". If the user requests dontcare, use value as "any".
Standing instructions allow you to add preferences or requirements that you’d like to consider when generating the parser.
If standing instructions are applicable across multiple domains, place an API call per situation per domain.
If some of the applicable standing instructions have instructions of similar type, place multiple API calls respecting the standing instructions.
If some slots are applicable across several domains, generate the respective slot names for the respective domains.
```

```
Schema:
Banks: recipient_account_name, amount, recipient_account_type
Buses: origin, departure_date, fare_type, transfers, price, group_size, destination, destination_station_name, origin_station_name, departure_time
Events: event_name, city, category, event_location, number_of_tickets, time, address_of_location, date, venue_address, event_type
Flights: origin, inbound_arrival_time, is_redeye, outbound_departure_time, outbound_arrival_time, inbound_departure_time, return_date, airlines, seating_class, refundable, number_stops, destination_airport, departure_date, fare, destination, passengers, origin_airport
Homes: pets_allowed, visit_date, address, property_name, rent, number_of_baths, area, number_of_beds, furnished, phone_number
Hotels: has_wifi, average_rating, check_out_date, price, pets_welcome, number_of_days, location, check_in_date, phone_number, number_of_rooms, street_address, hotel_name
HouseStays: rating, phone_number, has_laundry_service, check_out_date, total_price, check_in_date, address, number_of_adults, where_to
Media: title, directed_by, subtitles, genre
Movies: theater_name, movie_name, price, show_date, location, show_time, number_of_tickets, genre, show_type, street_address
Music: song_name, year, album, artist, genre, playback_device
RentalCars: dropoff_date, pickup_time, pickup_city, pickup_date, total_price, car_type, car_name, pickup_location
Restaurants: price_range, restaurant_name, city, has_live_music, serves_alcohol, time, date, phone_number, cuisine, street_address, party_size
Salons: is_unisex, average_rating, city, appointment_date, appointment_time, stylist_name, phone_number, street_address
Dentists: dentist_name, phone_number, offers_cosmetic_services, city, appointment_date, appointment_time, address
Doctors: doctor_name, city, average_rating, appointment_date, appointment_time, type, phone_number, street_address
Travel: good_for_kids, category, attraction_name, location, phone_number, free_entry
Weather: city, temperature, date, precipitation, humidity, wind

Further, following slots have categorical values:
recipient_account_type: checking, savings
fare_type: Economy, Economy extra, Flexible
(Travel) category: Place of Worship, Theme Park, Museum, Historical Landmark, Park, Tourist Attraction, Sports Venue, Shopping Area, Performing Arts Venue, Nature Preserve
event_type: Music, Sports
seating_class: Economy, Premium Economy, Business, First Class
refundable: True, False
airlines: United Airlines, American Airlines, Delta Airlines, Southwest Airlines, Alaska Airlines, British Airways, Air Canada, Air France
show_type: regular, 3d, imax
playback_device: TV, kitchen speaker, bedroom speaker
(Doctors) type: Gynecologist, ENT Specialist, Ophthalmologist, General Practitioner, Dermatologist
car_type: Compact, Standard, Full-size
price_range: inexpensive, moderate, expensive, very expensive

Further, following slots are boolean:
has_wifi, pets_allowed, subtitles, offers_cosmetic_services, has_laundry_service, is_unisex, good_for_kids, has_live_music, pets_welcome, serves_alcohol, is_redeye, furnished, free_entry
```

Example template:

```
Dialogue:
{{ user_utterance }}

Applicable Standing Instructions:
{{ applicable_instructions | join("\n> ") }}

API Calls:
```

Target template:

```
{{ target_api_calls | join("\n") }}
```

F.2 DEFAULT Prompt V1

System prompt template:

You are designing a parser that takes in a user utterance and some standing instructions and outputs a set of API calls.
Every API call consists of "GetX" where X is domain name and uses slot names listed below as arguments. We list the domain name followed by the list of possible slot names. Some slot names can be categorical or boolean
The values for the arguments can come from the user's dialogue or standing instructions. If the user requests a slot name and no value is found, use "?". If the user requests dontcare, use value as "any".
Standing instructions allow you to add preferences or requirements that you'd like to consider when generating the parser.
If standing instructions are applicable across multiple domains, place an API call per situation per domain.
If some of the applicable standing instructions have instructions of similar type, place multiple API calls respecting the standing instructions.
If some slots are applicable across several domains, generate the respective slot names for the respective domains.

Schema:
Banks: recipient_account_name, amount, recipient_account_type
Buses: origin, departure_date, fare_type, transfers, price, group_size, destination, destination_station_name, origin_station_name, departure_time
Events: event_name, city, category, event_location, number_of_tickets, time, address_of_location, date, venue_address, event_type
Flights: origin, inbound_arrival_time, is_redeye, outbound_departure_time, outbound_arrival_time, inbound_departure_time, return_date, airlines, seating_class, refundable, number_stops, destination_airport, departure_date, fare, destination, passengers, origin_airport
Homes: pets_allowed, visit_date, address, property_name, rent, number_of_baths, area, number_of_beds, furnished, phone_number
Hotels: has_wifi, average_rating, check_out_date, price, pets_welcome, number_of_days, location, check_in_date, phone_number, number_of_rooms, street_address, hotel_name
HouseStays: rating, phone_number, has_laundry_service, check_out_date, total_price, check_in_date, address, number_of_adults, where_to
Media: title, directed_by, subtitles, genre
Movies: theater_name, movie_name, price, show_date, location, show_time, number_of_tickets, genre, show_type, street_address
Music: song_name, year, album, artist, genre, playback_device
RentalCars: dropoff_date, pickup_time, pickup_city, pickup_date, total_price, car_type, car_name, pickup_location
Restaurants: price_range, restaurant_name, city, has_live_music, serves_alcohol, time, date, phone_number, cuisine, street_address, party_size
Salons: is_unisex, average_rating, city, appointment_date, appointment_time, stylist_name, phone_number, street_address
Dentists: dentist_name, phone_number, offers_cosmetic_services, city, appointment_date, appointment_time, address
Doctors: doctor_name, city, average_rating, appointment_date, appointment_time, type, phone_number, street_address
Travel: good_for_kids, category, attraction_name, location, phone_number, free_entry
Weather: city, temperature, date, precipitation, humidity, wind

Further, following slots have categorical values:
recipient_account_type: checking, savings
fare_type: Economy, Economy extra, Flexible
(Travel) category: Place of Worship, Theme Park, Museum, Historical Landmark, Park, Tourist Attraction, Sports Venue, Shopping Area, Performing Arts Venue, Nature Preserve
event_type: Music, Sports
seating_class: Economy, Premium Economy, Business, First Class
refundable: True, False
airlines: United Airlines, American Airlines, Delta Airlines, Southwest Airlines, Alaska Airlines, British Airways, Air Canada, Air France
show_type: regular, 3d, imax
playback_device: TV, kitchen speaker, bedroom speaker
(Doctors) type: Gynecologist, ENT Specialist, Ophthalmologist, General Practitioner, Dermatologist
car_type: Compact, Standard, Full-size
price_range: inexpensive, moderate, expensive, very expensive

Further, following slots are boolean:
has_wifi, pets_allowed, subtitles, offers_cosmetic_services, has_laundry_service, is_unisex, good_for_kids, has_live_music, pets_welcome, serves_alcohol, is_redeye, furnished, free_entry

```
—

{% if model_name == "llama" %}
Follow the following format.
{% else %}
The examples are formatted as follows.
{% endif %}

Dialogue:
<user_utterance>

Applicable Standing Instructions:
<applicable_standing_instructions>

API Calls:
API calls to solve the user task

—

{% if model_name == "llama" %}
You are given several independent examples of the task:
{% endif %}
```

Example template:

```
{% if split == "test"  and model_name == "llama" %}
Given the examples above, output only the API calls for the following example with no additional text:
{% endif %}

Dialogue:
{{ user_utterance }}

Applicable Standing Instructions:
{{ applicable_instructions | join("\n> ") }}
```

API Calls:

Target template:

```
{{ target_api_calls | join("\n") }}
```

F.3 DEFAULT Prompt V2

System prompt template:

You are designing a parser that takes in a user utterance (field ‘user_utterance‘) and a user profile with standing instructions (field ‘user_profile‘) and outputs a set of API calls as an answer.
Every API call consist of "GetX" where X is domain name and uses slot names listed below as arguments. We list the domain name followed by the list of possible slot names in the ‘api_schema‘ field. Some slot names can be categorical or boolean.
The values for the arguments can come from the user’s dialogue or standing instructions. If the user asks about a slot but no value is found, set its value to "?". If the user explicitly says they do not care about a particular slot, set its value to "any".
Standing instructions allow you to add preferences or requirements that you’d like to consider when generating the parser.
If standing instructions are applicable across multiple domains, place an API call per situation per domain.
If some of the applicable standing instructions have instructions of similar type, place multiple API calls respecting the standing instructions.
If some slots are applicable across several domains, generate the respective slot names for the respective domains.

The schema template, input description and example formatting are the same as in Section F.3

F.4 EXTERNAL TAG (EXT-TAG) Prompt V1

System prompt template:

You are designing a parser that takes in a user query and some user preferences and outputs a set of API calls. Execution of the API calls helps to answer the user query.
Every function name in the API call has a structure of "GetX" where X is domain name. Each function uses slot names listed below as arguments. Some slot names can be categorical or boolean. The values for the arguments can come from the user’s query or user preferences. If the user requests a slot name and no value is found, use "?". If the user says they don’t care, set slot value to "any".
User preferences allow you to add preferences or requirements that you’d like to consider when generating the parser. If user preferences are applicable across multiple domains, place an API call per situation per domain. If some of the applicable preferences have instructions of similar type, place multiple API calls respecting the preferences. If some slots are applicable across several domains, generate the respective slot names for the respective domains.

The user profile would be tagged in the following format:
<a:API_FUNCTION_NAME> text </a> would mean the function that is relevant for the text in brackets
<sl:SLOT_NAME> value </sl> would highlight which function arguments are used in the function and their value.

Output a list of API calls that would answer the user query. There can be several API calls per user query, but not always, so output only the required calls. Make sure you follow the following output structure: GetX(slot1="value1", slot2="value2"). Use the tags from the user profile, as well as information from the current dialogue to generate the calls. In cases, where seceral API calls are required, generate each one in a new line. Use only the functions from the documentation above, and make sure to check that only the slots for the chosen function are used in the API call. Generate only the API calls.

The list of the available function names is presented below, followed by possible slot names.
Some of the possible API calls include functions:
GetBanks: handling all the banking information (recipient_account_name, amount, recipient_account_type)
GetBuses: finding and booking bus tickets and routes (origin, departure_date, fare_type, transfers, price, group_size, destination, departure_time)
GetEvents: finding and booking events (event_name, city, category, number_of_tickets, time, date, venue_address, event_type)
GetFlights: finding and booking flights (origin, inbound_arrival_time, is_redeye, outbound_departure_time, outbound_arrival_time, inbound_departure_time, return_date, airlines, seating_class, refundable, number_stops, departure_date, fare, destination, passengers)
GetHomes: looking for property (pets_allowed, visit_date, address, property_name, rent, number_of_baths, area, number_of_beds, furnished, phone_number)
GetHotels: booking hotels (has_wifi, average_rating, check_out_date, price, pets_welcome, number_of_days, location, check_in_date, phone_number, number_of_rooms, street_address, hotel_name)
GetHouseStays: booking temporary accommodation (rating, phone_number, has_laundry_service, check_out_date, total_price, check_in_date, address, number_of_adults, where_to)
GetMedia: searching for online media (title, directed_by, subtitles, genre)
GetMovies: searching for cinema tickets (theater_name, movie_name, price, show_date, location, show_time, number_of_tickets, genre, show_type, street_address)
GetMusic: finding songs (song_name, year, album, artist, genre, playback_device)
GetRentalCars: booking rental cars (dropoff_date, pickup_time, pickup_city, pickup_date, total_price, car_type, car_name, pickup_location)
GetRestaurants: finding and booking restaurants (price_range, restaurant_name, city, has_live_music, serves_alcohol, time, date, phone_number, cuisine, street_address, party_size)
GetSalons: finding hair salons (is_unisex, average_rating, city, appointment_date, appointment_time, stylist_name, phone_number, street_address)
GetDentists: finding dentists (dentist_name, phone_number, offers_cosmetic_services, city, appointment_date, appointment_time, address)
GetDoctors: finding doctors (doctor_name, city, average_rating, appointment_date, appointment_time, type, phone_number, street_address)
GetTravel: finding attractions (good_for_kids, category, attraction_name, location, phone_number, free_entry)
GetWeather: getting weather information (city, temperature, date, precipitation, humidity, wind)

Further, following slots have categorical values:
recipient_account_type: checking, savings
fare_type: Economy, Economy extra, Flexible
(Travel) category: Place of Worship, Theme Park, Museum, Historical Landmark, Park, Tourist Attraction, Sports Venue, Shopping Area, Performing Arts Venue, Nature Preserve
event_type: Music, Sports
seating_class: Economy, Premium Economy, Business, First Class
refundable: True, False
airlines: United Airlines, American Airlines, Delta Airlines, Southwest Airlines, Alaska Airlines, British Airways, Air Canada, Air France
show_type: regular, 3d, imax
playback_device: TV, kitchen speaker, bedroom speaker

```
(Doctors) type: Gynecologist, ENT Specialist, Ophthalmologist, General Practitioner, Dermatologist
car_type: Compact, Standard, Full-size
price_range: inexpensive, moderate, expensive, very expensive

Further, following slots are boolean:
has_wifi, pets_allowed, subtitles, offers_cosmetic_services, has_laundry_service, is_unisex, good_for_kids, has_live_music, pets_welcome, serves_alcohol, is_redeye, furnished, free_entry
```

```
—

{% if model_name == "llama" %}
Follow the following format.
{% else %}
The examples are formatted as follows.
{% endif %}

Dialogue:
<user_utterance>

Applicable Standing Instructions:
<applicable_standing_instructions>

Tagged Standing Instructions:
<tagged applicable standing instructions>

API Calls:
API calls to solve the user task

—

{% if model_name == "llama" %}
You are given several independent examples of the task:
{% endif %}
```

Example template:

```
{% if split == "test"  and model_name == "llama" %}
Given the examples above, output only the API calls for the following example with no additional text:
{% endif %}

Dialogue:
{{ user_utterance }}

Applicable Standing Instructions:
{{ applicable_instructions | join("\n> ") }}

Tagged Applicable Standing Instructions:
{{ tagged_applicable_instructions | join("\n> ") }}

API Calls:
```

Target template:

```
{{ target_api_calls | join("\n") }}
```

F.5 External Tag (Ext-Tag) Prompt V2

System prompt template:

```
You are a parser that converts user queries and profile preferences into API calls to fulfill the query. Use the provided tags, dialogue, and schema to generate precise API calls.

**Task Guidelines:**
1. **API Call Structure:**
   Format each call as ‘GetX(slot1="value1", slot2="value2", ...)‘, where ‘X‘ is the domain name, and slots match the chosen function.

2. **Using Tags:**
   - ‘<a:API_FUNCTION_NAME>‘ marks relevant functions.
   - ‘<sl:SLOT_NAME>‘ specifies slot values.
   Example: ‘<a:GET_FLIGHTS> Request <sl:AIRLINES> Alaska Airlines</sl></a>‘ becomes ‘airlines="Alaska Airlines"‘.

3. **Slot Values:**
   - Use values from the query or tags.
   - Assign ‘"?"‘ if a slot is missing and ‘"any"‘ if the user has no preference.

4. **Output Requirements:**
   - Include only required API calls.
   - Output each call on a new line.

—
```

```
**Schema:**
Use valid functions and slots as listed:

#### **Functions and Slots**
Each function corresponds to a specific domain and has associated slots. Use only the listed slots for each function.

- **GetBanks**
  - Slots: ‘recipient_account_name‘, ‘amount‘, ‘recipient_account_type‘

- **GetBuses**
  - Slots: ‘origin‘, ‘departure_date‘, ‘fare_type‘, ‘transfers‘, ‘price‘, ‘group_size‘, ‘destination‘, ‘departure_time‘

- **GetEvents**
  - Slots: ‘event_name‘, ‘city‘, ‘category‘, ‘number_of_tickets‘, ‘time‘, ‘date‘, ‘venue_address‘, ‘event_type‘

- **GetFlights**
  - Slots: ‘origin‘, ‘inbound_arrival_time‘, ‘is_redeye‘, ‘outbound_departure_time‘, ‘outbound_arrival_time‘, ‘inbound_departure_time‘, ‘return_date‘, ‘airlines‘, ‘seating_class‘, ‘refundable‘, ‘number_stops‘, ‘departure_date‘, ‘fare‘, ‘destination‘, ‘passengers‘

- **GetHomes**
  - Slots: ‘pets_allowed‘, ‘visit_date‘, ‘address‘, ‘property_name‘, ‘rent‘, ‘number_of_baths‘, ‘area‘, ‘number_of_beds‘, ‘furnished‘, ‘phone_number‘

- **GetHotels**
  - Slots: ‘has_wifi‘, ‘average_rating‘, ‘check_out_date‘, ‘price‘, ‘pets_welcome‘, ‘number_of_days‘, ‘location‘, ‘check_in_date‘, ‘phone_number‘, ‘number_of_rooms‘, ‘street_address‘, ‘hotel_name‘

- **GetHouseStays**
  - Slots: ‘rating‘, ‘phone_number‘, ‘has_laundry_service‘, ‘check_out_date‘, ‘total_price‘, ‘check_in_date‘, ‘address‘, ‘number_of_adults‘, ‘where_to‘
- **GetMedia**
  - Slots: ‘title‘, ‘directed_by‘, ‘subtitles‘, ‘genre‘

- **GetMovies**
  - Slots: ‘theater_name‘, ‘movie_name‘, ‘price‘, ‘show_date‘, ‘location‘, ‘show_time‘, ‘number_of_tickets‘, ‘genre‘, ‘show_type‘, ‘street_address‘

- **GetMusic**
  - Slots: ‘song_name‘, ‘year‘, ‘album‘, ‘artist‘, ‘genre‘, ‘playback_device‘

- **GetRentalCars**
  - Slots: ‘dropoff_date‘, ‘pickup_time‘, ‘pickup_city‘, ‘pickup_date‘, ‘total_price‘, ‘car_type‘, ‘car_name‘, ‘pickup_location‘

- **GetRestaurants**
  - Slots: ‘price_range‘, ‘restaurant_name‘, ‘city‘, ‘has_live_music‘, ‘serves_alcohol‘, ‘time‘, ‘date‘, ‘phone_number‘, ‘cuisine‘, ‘street_address‘, ‘party_size‘

- **GetSalons**
  - Slots: ‘is_unisex‘, ‘average_rating‘, ‘city‘, ‘appointment_date‘, ‘appointment_time‘, ‘stylist_name‘, ‘phone_number‘, ‘street_address‘

- **GetDentists**
  - Slots: ‘dentist_name‘, ‘phone_number‘, ‘offers_cosmetic_services‘, ‘city‘, ‘appointment_date‘, ‘appointment_time‘, ‘address‘

- **GetDoctors**
  - Slots: ‘doctor_name‘, ‘city‘, ‘average_rating‘, ‘appointment_date‘, ‘appointment_time‘, ‘type‘, ‘phone_number‘, ‘street_address‘

- **GetTravel**
  - Slots: ‘good_for_kids‘, ‘category‘, ‘attraction_name‘, ‘location‘, ‘phone_number‘, ‘free_entry‘

- **GetWeather**
  - Slots: ‘city‘, ‘temperature‘, ‘date‘, ‘precipitation‘, ‘humidity‘, ‘wind‘

—

### **Slot Value Types**

#### **Categorical Slots**
- ‘recipient_account_type‘: ‘checking‘, ‘savings‘
- ‘fare_type‘: ‘Economy‘, ‘Economy extra‘, ‘Flexible‘
- ‘category‘ (Travel): ‘Place of Worship‘, ‘Theme Park‘, ‘Museum‘, ‘Historical Landmark‘, ‘Park‘, ‘Tourist Attraction‘, ‘Sports Venue‘, ‘Shopping Area‘, ‘Performing Arts Venue‘, ‘Nature Preserve‘
- ‘event_type‘: ‘Music‘, ‘Sports‘
- ‘seating_class‘: ‘Economy‘, ‘Premium Economy‘, ‘Business‘, ‘First Class‘
- ‘refundable‘: ‘True‘, ‘False‘
- ‘airlines‘: ‘United Airlines‘, ‘American Airlines‘, ‘Delta Airlines‘, ‘Southwest Airlines‘, ‘Alaska Airlines‘, ‘British Airways‘, ‘Air Canada‘, ‘Air France‘
- ‘show_type‘: ‘regular‘, ‘3d‘, ‘imax‘
- ‘playback_device‘: ‘TV‘, ‘kitchen speaker‘, ‘bedroom speaker‘
- ‘type‘ (Doctors): ‘Gynecologist‘, ‘ENT Specialist‘, ‘Ophthalmologist‘, ‘General Practitioner‘, ‘Dermatologist‘
- ‘car_type‘: ‘Compact‘, ‘Standard‘, ‘Full-size‘
- ‘price_range‘: ‘inexpensive‘, ‘moderate‘, ‘expensive‘, ‘very expensive‘

#### **Boolean Slots**
- ‘has_wifi‘, ‘pets_allowed‘, ‘subtitles‘, ‘offers_cosmetic_services‘, ‘has_laundry_service‘, ‘is_unisex‘, ‘good_for_kids‘, ‘has_live_music‘, ‘pets_welcome‘, ‘serves_alcohol‘, ‘is_redeye‘, ‘furnished‘, ‘free_entry‘

—

Ensure all outputs strictly adhere to the required format and schema. Generate only API calls.
```

The input description and example templates are the same as in Section F.4

F.6 JOINT-TAG Prompt

System prompt template:

You are designing a parser that takes in a user utterance and some standing instructions and outputs a set of API calls.
Every API call consist of "GetX" where X is domain name and uses slot names listed below as arguments. We list the domain name followed by the list of possible slot names. Some slot names can be categorical or boolean
The values for the arguments can come from the user's dialogue or standing instructions. If the user asks about a slot but no value is found, set its value to "?".
If the user explicitly says they do not care about a particular slot, set its value to "any".
Standing instructions allow you to add preferences or requirements that you'd like to consider when generating the parser.
If standing instructions are applicable across multiple domains, place an API call per situation per domain.
If some of the applicable standing instructions have instructions of similar type, place multiple API calls respecting the standing instructions.
If some slots are applicable across several domains, generate the respective slot names for the respective domains.

Think step by step.
First, identify and label API calls and their slots within applicable standing instructions.
Use action tags such as <a:API_NAME> ... </a>, and nested tags denoting specific attributes, like <sl:SLOT_NAME> ... </sl>.
Ensure that all tags are correctly placed, slot and API names are correct, all original sentence tokens are present and are in the correct order, no additional tokens are added, and slot values include only necessary information, e.g. the value of the slot.
Use those generated labels, as well as information from the dialogue to create the calls.
After that, output a list of API calls that would answer the user query.

Schema:
Banks: recipient_account_name, amount, recipient_account_type
Buses: origin, departure_date, fare_type, transfers, price, group_size, destination, destination_station_name, origin_station_name, departure_time
Events: event_name, city, category, event_location, number_of_tickets, time, address_of_location, date, venue_address, event_type
Flights: origin, inbound_arrival_time, is_redeye, outbound_departure_time, outbound_arrival_time, inbound_departure_time, return_date, airlines, seating_class, refundable, number_stops, destination_airport, departure_date, fare, destination, passengers, origin_airport
Homes: pets_allowed, visit_date, address, property_name, rent, number_of_baths, area, number_of_beds, furnished, phone_number
Hotels: has_wifi, average_rating, check_out_date, price, pets_welcome, number_of_days, location, check_in_date, phone_number, number_of_rooms, street_address, hotel_name
HouseStays: rating, phone_number, has_laundry_service, check_out_date, total_price, check_in_date, address, number_of_adults, where_to
Media: title, directed_by, subtitles, genre
Movies: theater_name, movie_name, price, show_date, location, show_time, number_of_tickets, genre, show_type, street_address
Music: song_name, year, album, artist, genre, playback_device
RentalCars: dropoff_date, pickup_time, pickup_city, pickup_date, total_price, car_type, car_name, pickup_location
Restaurants: price_range, restaurant_name, city, has_live_music, serves_alcohol, time, date, phone_number, cuisine, street_address, party_size
Salons: is_unisex, average_rating, city, appointment_date, appointment_time, stylist_name, phone_number, street_address
Dentists: dentist_name, phone_number, offers_cosmetic_services, city, appointment_date, appointment_time, address
Doctors: doctor_name, city, average_rating, appointment_date, appointment_time, type, phone_number, street_address
Travel: good_for_kids, category, attraction_name, location, phone_number, free_entry
Weather: city, temperature, date, precipitation, humidity, wind

Further, following slots have categorical values:
recipient_account_type: checking, savings
fare_type: Economy, Economy extra, Flexible
(Travel) category: Place of Worship, Theme Park, Museum, Historical Landmark, Park, Tourist Attraction, Sports Venue, Shopping Area, Performing Arts Venue, Nature Preserve
event_type: Music, Sports
seating_class: Economy, Premium Economy, Business, First Class
refundable: True, False
airlines: United Airlines, American Airlines, Delta Airlines, Southwest Airlines, Alaska Airlines, British Airways, Air Canada, Air France
show_type: regular, 3d, imax
playback_device: TV, kitchen speaker, bedroom speaker
(Doctors) type: Gynecologist, ENT Specialist, Ophthalmologist, General Practitioner, Dermatologist
car_type: Compact, Standard, Full-size
price_range: inexpensive, moderate, expensive, very expensive

Further, following slots are boolean:
has_wifi, pets_allowed, subtitles, offers_cosmetic_services, has_laundry_service, is_unisex, good_for_kids, has_live_music, pets_welcome, serves_alcohol, is_redeye, furnished, free_entry

—

{% if model_name == "llama" %}
Follow the following format.
{% else %}
The examples are formatted as follows.
{% endif %}

Dialogue:
<user_utterance>

Applicable Standing Instructions:
<applicable_standing_instructions>

Tagged Standing Instructions:
Tagged standing instructions

API Calls:
API calls to solve the user task

—

{% if model_name == "llama" %}
You are given several independent examples of the task:

```
{% endif %}
```

Example template:

```
{% if split == "test"  and model_name == "llama" %}
Given the examples above, output only the API calls for the following example with no additional text:
{% endif %}

Dialogue:
{{ user_utterance }}

Applicable Standing Instructions:
{{ applicable_instructions | join("\n> ") }}

Tagged Applicable Standing Instructions:
```

Target template:

```
{{ tagged_applicable_instructions | join("\n> ") }}

API Calls:
{{ target_api_calls | join("\n") }}
```

F.7 Tagger Prompt

System prompt template:

Create a sentence tagging model capable of identifying and labeling actions and their associated details within sentences. Given a sentence, the model should appropriately tag actions and their attributes within the sentence.
The output should include all of the tokens from the original sentence, as well as action tags such as [IN:ACTION] and nested tags denoting specific attributes, like [SL:ATTRIBUTE value].
Ensure the model can effectively handle a variety of sentences and accurately mark actions and their related details.

Every action name has the format of "GET_X", where X denotes the domain name.
Every action has a list of associated attributes. Only those attributes can be present inside the action tag.

The list of the available function names is presented below, followed by possible slot names.
Some of the possible API calls include functions:
GetBanks: handling all the banking information (recipient_account_name, amount, recipient_account_type)
GetBuses: finding and booking bus tickets and routes (origin, departure_date, fare_type, transfers, price, group_size, destination, departure_time)
GetEvents: finding and booking events (event_name, city, category, number_of_tickets, time, date, venue_address, event_type)
GetFlights: finding and booking flights (origin, inbound_arrival_time, is_redeye, outbound_departure_time, outbound_arrival_time, inbound_departure_time, return_date, airlines, seating_class, refundable, number_stops, departure_date, fare, destination, passengers)
GetHomes: looking for property (pets_allowed, visit_date, address, property_name, rent, number_of_baths, area, number_of_beds, furnished, phone_number)
GetHotels: booking hotels (has_wifi, average_rating, check_out_date, price, pets_welcome, number_of_days, location, check_in_date, phone_number, number_of_rooms, street_address, hotel_name)
GetHouseStays: booking temporary accommodation (rating, phone_number, has_laundry_service, check_out_date, total_price, check_in_date, address, number_of_adults, where_to)
GetMedia: searching for online media (title, directed_by, subtitles, genre)
GetMovies: searching for cinema tickets (theater_name, movie_name, price, show_date, location, show_time, number_of_tickets, genre, show_type, street_address)
GetMusic: finding songs (song_name, year, album, artist, genre, playback_device)
GetRentalCars: booking rental cars (dropoff_date, pickup_time, pickup_city, pickup_date, total_price, car_type, car_name, pickup_location)
GetRestaurants: finding and booking restaurants (price_range, restaurant_name, city, has_live_music, serves_alcohol, time, date, phone_number, cuisine, street_address, party_size)
GetSalons: finding hair salons (is_unisex, average_rating, city, appointment_date, appointment_time, stylist_name, phone_number, street_address)
GetDentists: finding dentists (dentist_name, phone_number, offers_cosmetic_services, city, appointment_date, appointment_time, address)
GetDoctors: finding doctors (doctor_name, city, average_rating, appointment_date, appointment_time, type, phone_number, street_address)
GetTravel: finding attractions (good_for_kids, category, attraction_name, location, phone_number, free_entry)
GetWeather: getting weather information (city, temperature, date, precipitation, humidity, wind)

Check that the output fits all of the criteria above, and all of the tags are correctly placed (for example, [SL:] tags must be inside the [IN:] tags)
Pay special attention to the attribute names and function names, check that none of the attribute names are mixed up (for example, some functions have similar attributes: city/location, make sure you are using the correct name)
Check that all of the tokens from the original untagged sentence are present and are in the correct order.
Check that the parser did not add any other tokens, except for the special ones.
Make sure that the attribute values inlcude only the necessary information (for example, '[SL:EVENT_TYPE event type is Music]' is incorrect and should be 'event type is [SL:EVENT_TYPE Music]').

Example template:

```
{{ instruction }}
```

Target template:

```
{{ tagged_instruction }}
```