Converso: Improving LLM Chatbot Interfaces and Task Execution via Conversational Forms

Gianfranco Demarco, Nicola Fanelli[a,*], Gennaro Vessio[a] and Giovanna Castellano[a]

[a]Department of Computer Science, University of Bari Aldo Moro, Italy

Abstract. Recent advancements in large language models (LLMs) have enabled more autonomous conversational AI agents. However, challenges remain in developing effective chatbots, particularly in addressing LLMs' lack of "statefulness". This paper presents Converso, a novel chatbot framework that introduces a new conversation flow based on stateful conversational forms designed for natural data acquisition through dialogue. Converso leverages LLMs, LangChain, and a containerized architecture to provide an end-to-end chatbot system with Telegram as the user interface. The key innovation in Converso is its implementation of conversational forms, which guide users through form completion via a structured dialogue flow. Converso's chatbots can be linked with multiple forms that are automatically triggered based on the user's intent. Our forms are fully integrated into the LangChain ecosystem, allowing the LLM to use tools for form completion and dynamic validation. Evaluations show that this approach significantly improves task completion rates compared to LLMs alone. Converso demonstrates how specifically designed conversational flows can enhance the capabilities of LLM-based chatbots for practical data collection applications. Our implementation is available at: https://github.com/gianfrancodemarco/converso-chatbot.

1 Introduction

Chatbots have emerged as one of the most widely adopted applications of artificial intelligence (AI) in consumer products and services. These conversational agents directly interact with end users through natural language interfaces, serving various domains such as entertainment, education, information retrieval, e-commerce, and more [1]. Since the early conceptualization of chatbots in the 1960s, numerous approaches have been explored to enhance their capabilities, transitioning from basic pattern-matching techniques to leveraging advanced machine learning algorithms and language models.

Recent advancements in large language models (LLMs) have led to a significant shift, advancing chatbots to new levels of independence and conversational ability [2]. LLMs are complex neural network architectures with billions of parameters, trained on extensive text corpora. The extensive data on which these models are trained, along with additional techniques like reinforcement learning from human feedback (RLHF) [14], enables them to generate natural language responses that closely resemble human communication. However, LLMs still face limitations, including a lack of access to up-to-date knowledge, an inability to perform complex reasoning, and difficulties interacting with external environments [13]. Researchers have developed techniques such as retrieval-augmented generation (RAG) [8] and model-calling capabilities [15] to address these challenges, allowing LLMs to leverage external data sources and tools during inference. Furthermore, open-source frameworks like LangChain [3] have emerged to streamline the development of LLM-based applications, including chatbots. LangChain provides a comprehensive suite of libraries, tools, and templates that facilitate the integration of LLMs, RAG techniques, and external tools, enabling the creation of context-aware and reasoning-capable chatbot systems.

This paper explores the design and implementation of modern chatbot systems leveraging LLMs, the LangChain ecosystem, and related techniques. It introduces Converso, a novel chatbot framework that incorporates a conversation flow based on *conversational forms*, enhancing traditional web forms for data acquisition through natural language interactions. The paper discusses Converso's system architecture, conversational flow, and use cases, highlighting the benefits of integrating LLMs and the LangChain ecosystem. Additionally, an evaluation protocol is presented to assess the effectiveness of conversational forms in improving task completion rates.

The rest of this paper is structured as follows: Section 2 discusses related work, Section 3 presents the Converso framework, Section 4 describes our experimental use cases and presents the results obtained, and Section 5 concludes the paper.

2 Related Work

Chatbots have been an active area of research for a long time, with early systems like ELIZA [18] and PARRY [6] employing pattern-matching techniques with pre-defined rules and responses. Subsequent work incorporated machine learning algorithms for intent classification and entity extraction [1], leading to more advanced chatbots capable of understanding user intents and relevant context.

With recent developments in LLMs like GPT-3 [2] and PaLM [5], there has been renewed interest in leveraging these powerful models for building conversational AI systems. LLMs have demonstrated impressive emergent abilities in few-shot prompting settings [2] and can engage in substantive multi-turn dialogues by conditioning on previous conversation history [17].

However, LLMs still face limitations such as hallucinating incorrect facts [12], being confined to their training data distributions, and lacking mechanisms to interact with external tools or information sources. Several techniques have been proposed to overcome these limitations. For instance, retrieval-augmented generation (RAG) [8] integrates external data retrieval into the language model's generation process to enhance factual accuracy and access to up-to-date information. Tool calling [15, 19] allows LLMs to call and receive

* Corresponding author. Email: nicola.fanelli@uniba.it.

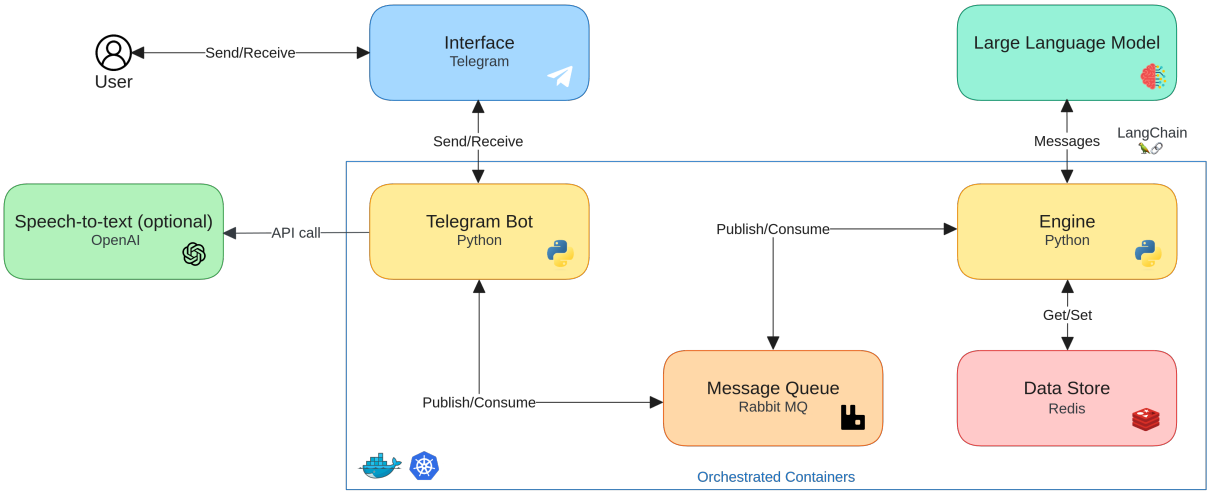


Figure 1: Overview of the system architecture of a chatbot implemented using the Converso framework. The user interacts through Telegram as the interface. User messages are written to the message queue by the Telegram bot, which also consumes the LLM messages to send chatbot responses back to the interface. We also allow for vocal user inputs via OpenAI speech-to-text APIs. The engine is responsible for providing prompts to the LLM via LangChain and maintaining the conversation history in a Redis data store. We use a fully containerized architecture, implementing our components inside Docker containers orchestrated by Kubernetes.

results from external tools/APIs, enabling capabilities beyond pure text generation, like mathematical reasoning [16] and real-world interactions [10].

Such advancements, combined with the ability to influence the behavior of LLMs through prompt engineering [4], have opened a wide range of applications for chatbot systems, such as data acquisition. For example, Hakimov et al. [9] proposed a modular system based on LLMs for form filling, creating a method to evaluate dialogues using user simulation with an LLM. Additionally, some simple (usually closed-source) implementations of forms for data acquisition with LLMs have emerged online.

In contrast to these approaches, we propose a framework for creating chatbots that is fully integrated with the LangChain ecosystem. Our innovative approach does not tie the LLM to a specific form, allowing chatbot developers to specify an arbitrary number of forms that can be invoked based on user intents. This flexibility enables users to interact with the chatbot for various goals beyond executing actions requiring form completion. Furthermore, with Converso, we propose a fully containerized architecture streamlining the chatbot creation process.

3 Our Framework: Converso

In this section, we present Converso, a framework developed as an extension of LangChain that enables building conversational forms for goal-oriented interactions. Converso introduces *stateful* conversational forms that guide users through structured data collection processes, reducing reliance on lengthy conversation histories and mitigating hallucinations or deviations from the intended goals. Converso facilitates the creation of fully containerized chatbot applications, leveraging Kubernetes for container orchestration, an event manager for asynchronous communication, and a Telegram bot interface as the front end. This enables a production-ready, scalable implementation of conversational chatbots that can be seamlessly integrated into existing systems.

3.1 System Architecture

Converso implements a fully functional chatbot system composed of several components. An overview of the system architecture of the Converso project is presented in Fig. 1.

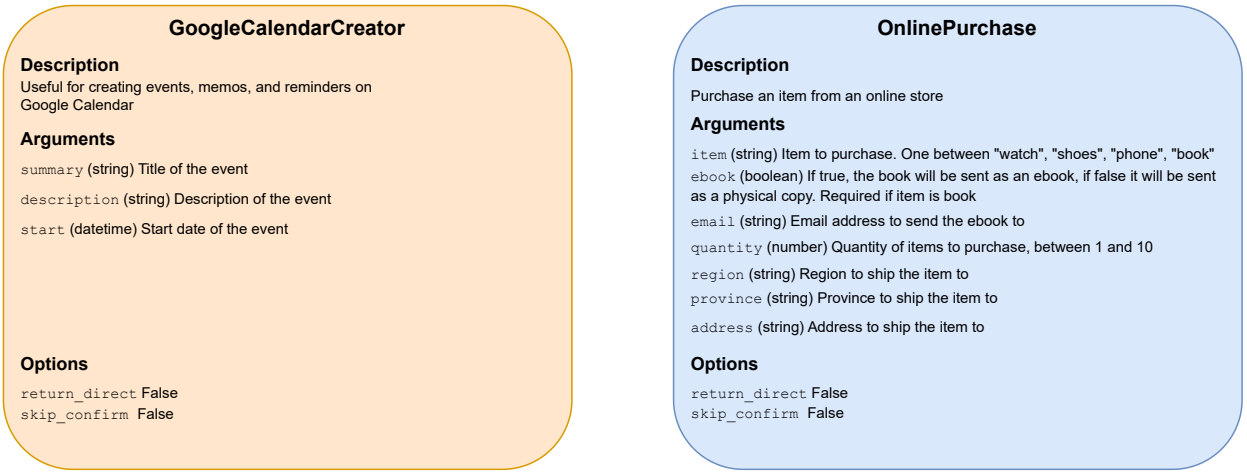
All components of the Converso chatbot are created as Docker containers orchestrated by Kubernetes. Users can interact with the LLM via a Telegram bot interface. RabbitMQ is used as a message broker to enable decoupling and asynchronous request processing. Finally, Redis stores conversation history and other data, such as user credentials.

3.2 Conversation Flow

The conversation flow generates a textual response starting from the user's input. This flow is constructed as a graph using LangGraph, a library for multi-actor interactions within LangChain. The primary components of our conversation flow are the Base Agent and the Error Agent. The Base Agent is responsible for conducting the conversation with the user, while the Error Agent assists by correcting any errors that may occur during the interaction. Agents are specific instances of the LLM, each equipped with unique system prompts that guide their behavior.

The conversation flow consists of the following steps:

1. The user's input and conversation history are injected into a prompt template to create the final input for the model.
2. The chosen LLM, instantiated as the Base Agent, is called with the rendered prompt as input.
3. The model's output is evaluated, with three possible outcomes:
   - The model produces an error, typically a formatting issue for structured outputs. In this case, a new prompt is constructed that includes the conversation history and the error. The Error Agent is then responsible for correcting the error.
   - The model produces the final answer, which is then sent back to the user, concluding the flow.



(a) *GoogleCalendarCreator* for the Personal Assistant use case

(b) *OnlinePurchase* for the Shopping Assistant use case

Figure 2: *FormTools* examples designed for our use cases and implemented using Converso. A *FormTool* includes a description, a set of arguments for the user to complete when prompted by the LLM during the conversation, and options regarding the return mode. The return mode can be either direct, where the result is given directly to the user without further interaction with the LLM, or indirect, where additional processing by the LLM is required. Additionally, the *FormTool* specifies whether a confirmation step is needed.

- The model requests a tool execution. If the execution results in an error, the error is handled as in the first case. Otherwise, the result can be sent directly to the user or modified by the LLM before being sent. The creator of the specific conversation flow can choose the best option.

3.3 Conversational Forms

A key contribution of Converso is the introduction of a multi-agent conversation flow based on conversational forms, where the latter are represented as *FormTools* within the LangChain framework. We implement *FormTools* to encapsulate user intents that require gathering structured data through a multi-turn conversation. This novel approach addresses the limitations of existing chatbots that rely solely on the current message and conversation history, which can lead to hallucinations or goal deviations, especially as the history becomes long [11].

The conversation flow in Converso is extended to recognize user intents that map to specific *FormTools*. When such an intent is detected, the corresponding *FormTool* is activated, and all other *FormTools* are temporarily inhibited. At this point, the Base Agent is replaced by the Form Agent, which uses specifically engineered prompts to drive the conversation to fill the form, prompting the user to provide the required information fields through natural language interactions. The activation mechanism for *FormTools* is inspired by the concept of *semantic frame evocation* [7], where key expressions trigger the activation of the appropriate frame. In Converso, the LLM's recognition of user intents serves as the triggering mechanism, dynamically activating the corresponding *FormTool* and its associated conversational form.

FormTools maintain an internal state that tracks the progress of the data collection process. This state can be inactive (initial state), active (collecting data from the user), or filled (all required data has been provided). The collected data is stored in an internal form object within the *FormTool*.

To enhance user experience, *FormTools* support dynamic validation of user inputs. For example, in a purchase scenario, the available shipping regions can be dynamically updated based on the user's selections, ensuring only valid options are presented. *FormTools* allows developers to define custom logic for form compilation, enabling them to determine the order in which fields should be filled, the actions to be taken based on the values provided, and to implement complex inter-field validation logic. Once all required data has been collected, the *FormTool* can execute its associated action, such as making an API call or performing a specific task. Before execution, if the developer of the specific tool requires the confirmation step, the user is presented with a summary of the collected information for confirmation, ensuring transparency and control over the process. Examples of *FormTools* are presented in Fig. 2.

The conversational forms approach, coupled with the stateful nature of *FormTools*, reduces the dependence on lengthy conversation histories, mitigating the risk of hallucinations or deviations from the intended goals. By abstracting the data collection process into structured forms, Converso simplifies the development of goal-oriented conversational applications while leveraging the power of LLMs.

4 Experiments

4.1 Use Cases

To showcase the functionalities of our framework and provide an implementation guide for developers, we implemented two use cases using Converso.

The first use case involves creating a chatbot as a Personal Assistant. The chatbot is implemented using the containerized system architecture illustrated in Fig. 1. What distinguishes different use cases in the Converso framework is the definition of the tools to use for the specific application, which in the case of the Personal Assistant are:

- The *PythonCodeInterpreter*, *GoogleSearch*, and *GmailRetriever* tools, which are *BaseTools* and take a single argument from the LLM to perform an operation with it (for example, the *PythonCodeInterpreter* takes in a valid Python script expressed as a string and executes it, returning the result to the LLM).
- The *GoogleCalendarRetriever*, *GoogleCalendarCreator*, and *GmailSender* tools which are implemented as *FormTools* to illustrate our conversational forms (Fig. 2a).

The second use case developed using Converso involves the creation of a Shopping Assistant. This example demonstrates how the framework can be used to enhance the shopping functionality of an e-commerce platform. The use case includes dynamic data validation: for instance, only certain regions are available, and once a region is selected, only the provinces within that region are shown. A single *FormTool*, named *OnlinePurchase* (Fig. 2b), is implemented for this use case. Figure 3 illustrates chat examples for both use cases.

4.2 Evaluation Protocol

In this section, we present our evaluation protocol for assessing Converso's performance. Specifically, our evaluation focus is not on the underlying LLM, which can be selected by the developer of a specific chatbot based on its use case. Instead, we aim to determine whether the conversation flow incorporating conversational forms introduced with Converso performs better in real-world scenarios than the basic conversation flow.

Our evaluation framework consists of three main components:

- The *Task Generator*, which uses predefined templates and instructions to create real-world scenarios.
- The *User Simulator*, an LLM that carries out the generated tasks by interacting with the Converso chatbot under evaluation.
- The *Converso System*, which implements either the basic conversation flow or the conversation flow with conversational forms.

Table 1: Prompt guidelines provided to the User Simulator for the evaluation protocol.

Task Type	**Prompt**
All the information contained in the first message (AFM)	State your intent to the system, and then follow its instructions to complete the task. Provide all the necessary data to the system in your first message.
No information contained in the first message (NFM)	State your intent to the system without providing any data, and then follow its instructions to complete the task. For example, "I want to create an event" or "I want to buy something."
Main information contained in the first message (MFM)	State what you want to do, providing only the main information, and then follow its instructions to complete the task. For example, "I want to create an event called Meeting" or "I want to buy a watch."
Confused user (CU)	State your intent to the system without providing any data, and then follow its instructions to complete the task. Act like a very naive user who doesn't know what to do: misspell words, give incorrect information, and then correct it.

Table 2: System prompts given to the different types of agents employed in Converso's chatbots. Information between curly braces is dynamically populated.

Agent	**Prompt**
Base Agent	You are a personal assistant trying to help the user. You always answer in English. The current datetime is {datetime}. Don't use any of your knowledge or information about the state of the world. If you need something, ask the user for it or use a tool to find or compute it.
Error Agent	[Base Agent prompt] + There was an error with your last action. Please fix it and try again. Error: {error}.
Form Agent (information needed)	Help the user fill data for form {form_tool.name}. Ask to provide the needed information. Now you must update the form with any information the user provided or ask the user to provide a value for the field {information_to_collect}. You MUST use the form {form_tool.name} tool to update the stored data every time the user provides one or more values.
Form Agent (confirmation needed)	Help the user fill data for {form_tool.name}. You have all the information you need. Show the user all of the information using bullet points and ask for confirmation: {information_collected}. If the user agrees, call the {form_tool.name} tool one more time with confirm=True. If the user doesn't want to change something, call it with confirm=False.

An evaluation task consists of the following:

- A user guideline, selected from the four listed in Table 1, which provides the User Simulator with instructions on how to behave as a user.
- A target tool, chosen from *GoogleCalendarCreator*, *GoogleCalendarRetriever*, *GmailSender*, *GmailRetriever*, or *OnlinePurchase* as defined in Section 4.1. These tools are adapted to implement the *FormTool* interface for evaluating the conversation flow with conversational forms.
- A target input, selected from 20 randomly generated payloads.

This setup brings to a total of 400 generated evaluation tasks. A task's execution is considered successful if the correct tool is invoked and filled with the correct inputs. In this case, the execution is stopped, and the run is marked as successful. The task is considered unsuccessful if the maximum number of iterations is reached or the models reach a stale phase (typically because the correct tool was called with the wrong inputs).

4.3 Implementation Details

The implementation of the conversation flows is based on LangChain. The Base Agent, Error Agent, and Form Agent are implemented as LangChain agents and connected using LangGraph. We report the system prompts used to tune each agent in Table 2. *FormTools* are implemented as subclasses of *BaseTool*. For all evaluation experiments, we use gpt-3.5-turbo-0125 via OpenAI APIs, both as the User Simulator and the LLM chatbot. The target tools for evaluation are implemented as *StructuredTools* to assess the basic conversation flow and as *FormTools*, managed by the Form Agent, to evaluate our conversation flow with conversational forms. The maximum number of iterations is set to 25.

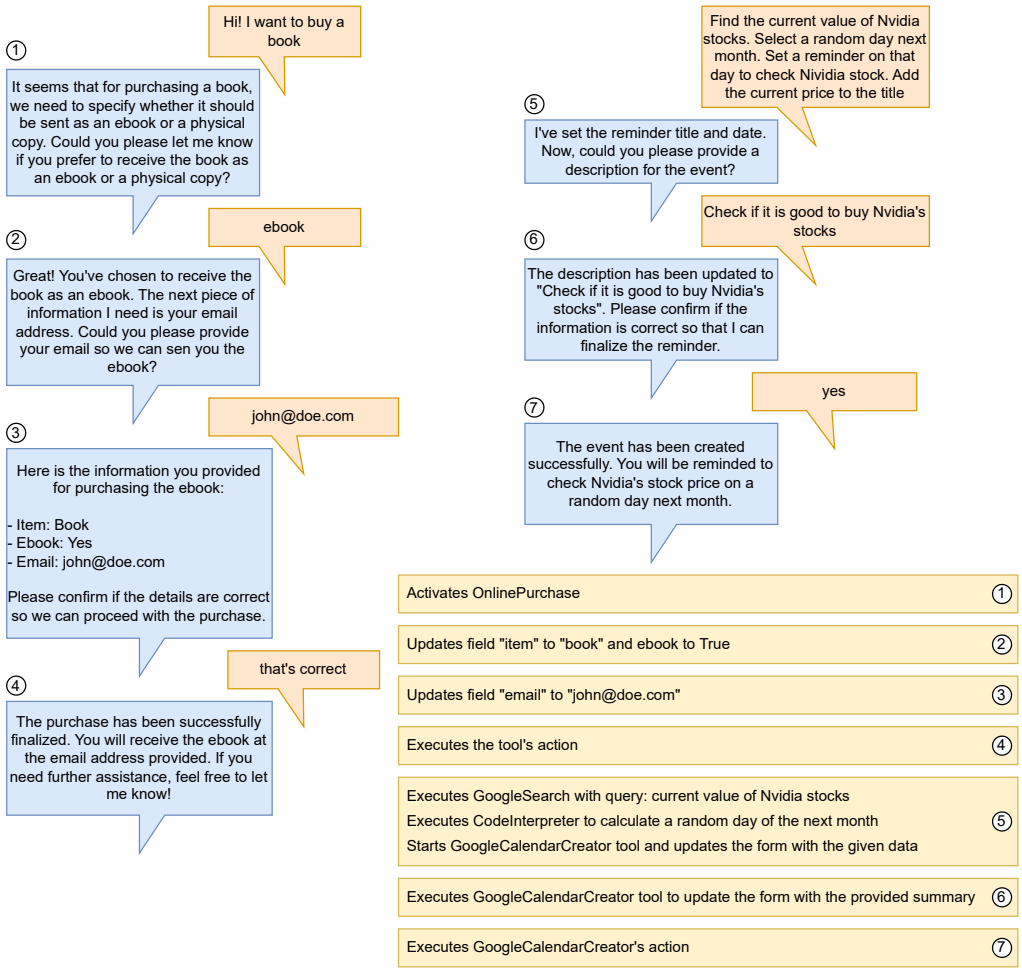


Figure 3: Two examples of chat illustrating our use cases. The chat on the left demonstrates the Shopping Assistant use case, while the chat on the right demonstrates the Personal Assistant use case. Orange messages with the tail pointing to the right represent user messages, while blue messages with the tail pointing to the left represent the chatbot. Actions executed by the chatbot using the connected tools are displayed inside yellow rectangles, which are numbered and connected to the corresponding points in the conversation. Notably, the agent can work with multiple tools simultaneously, allowing it to perform complex combinations of actions.

4.4 Results

Table 3 presents the results obtained using our evaluation protocol. The percentage of tasks executed correctly is 75.7% when using the basic conversation flow. It rises to 92.5% when employing the conversational forms, with a consistent 16.8% increase, showing the overall effectiveness of using *FormTools* with the Form Agent.

Table 3: Evaluation results, with scores expressed as percentages of success for the evaluation tasks defined in Section 4.2.

	Conversation Flow	
Tool	Basic	Ours
GmailRetriever	75.00	100.00
GmailSender	77.50	81.25
GoogleCalendarCreator	81.25	90.00
GoogleCalendarRetriever	90.00	95.00
OnlinePurchase	55.00	96.25
Task Type	Basic	Ours
AFM	91.00	99.00
NFM	73.00	93.00
MFM	62.00	89.00
CU	77.00	89.00
Total	Basic	Ours
	75.7	92.5

Analyzing the results from the perspective of tool usage, we observe that implementing conversational flows improves the use of every tool. Notably, the improvements are nearly double for the *OnlinePurchase* tool, which requires the most significant number of parameters and is, therefore, the most complex. This suggests a correlation between tool complexity and the benefits of using conversational forms. This outcome was expected, as more complex tools require keeping more detailed information in the conversation history, which can lead to goal deviation or hallucinations.

Considering the task type, we observe that the most challenging situation for the basic conversation flow occurs when the User Simulator's first prompt contains only the primary information for using the tool. We investigated this situation qualitatively (Fig. 4). In this case, the Base Agent invokes the correct tool but fills in the remaining information with hallucinated data without confirmation. This poses a risk for real-life applications, where actions could be executed with

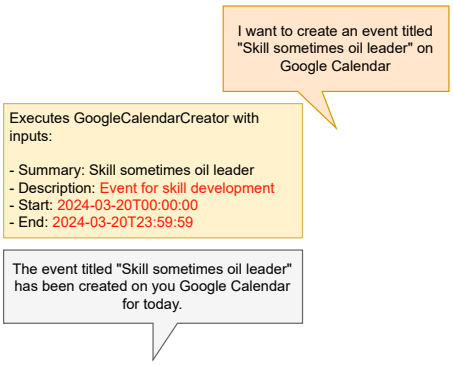


Figure 4: Chat example for an MFM task using the basic conversation flow (chatbot's response is in the gray box). The correct tool is executed but with incorrect inputs (highlighted in red), demonstrating the hallucination problem that occurs when conversational forms are not used.

incorrect inputs without the user's awareness. Conversely, conversational forms address this issue by querying the user for the missing information and requesting confirmation before executing any action.

5 Conclusion

Our work demonstrates the significant advancements in chatbot systems by integrating large language models and modern frameworks like LangChain. The proposed Converso framework, which incorporates conversational forms, showcases the potential to enhance user interactions by transforming traditional data acquisition methods into dynamic, interactive conversations. The evaluation results indicate a marked improvement in task success rates when using conversational forms, particularly with complex tools requiring detailed input.

By addressing the limitations of basic conversation flows—such as the hallucination of data and lack of confirmation—Converso improves accuracy and ensures a safer and more reliable user experience. This is especially crucial in real-world applications where incorrect data could lead to unintended and potentially harmful actions.

Furthermore, the Converso framework's robustness, demonstrated by its ability to handle diverse and complex use cases in the evaluation experiments, underscores its versatility. The consistent performance improvements, nearly doubling success rates in some cases, highlight the effectiveness of integrating stateful interactions and form-based data gathering.

Our work contributes to conversational AI by providing a practical and scalable solution for developing sophisticated chatbot applications. The insights gained from this research pave the way for future innovations in chatbot design, aiming to bridge the gap between human-like interactions and automated systems.

Acknowledgements

The research of Nicola Fanelli is funded by a PhD fellowship within the framework of the Italian "D.M. n. 118/23" - under the National Recovery and Resilience Plan, Mission 4, Component 1, Investment 4.1 - PhD Project "Analisi e valorizzazione del patrimonio artistico digitalizzato mediante tecniche di Intelligenza Artificiale" (CUP H91I23000690007).

References

[1] E. Adamopoulou and L. Moussiades. An Overview of Chatbot Technology. In I. Maglogiannis, L. Iliadis, and E. Pimenidis, editors, *Artificial Intelligence Applications and Innovations - 16th IFIP WG 12.5 International Conference, AIAI 2020, Neos Marmaras, Greece, June 5-7, 2020, Proceedings, Part II*, volume 584 of *IFIP Advances in Information and Communication Technology*, pages 373–383. Springer, 2020.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[3] H. Chase. LangChain, Oct. 2022.

[4] B. Chen, Z. Zhang, N. Langrené, and S. Zhu. Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review. *CoRR*, abs/2310.14735, 2023.

[5] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. PaLM: Scaling Language Modeling with Pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023.

[6] K. M. Colby, F. D. Hilf, S. Weber, and H. C. Kraemer. Turing-like Indistinguishability Tests for the Calidation of a Computer Simulation of Paranoid Processes. *Artif. Intell.*, 3(1-3):199–221, 1972.

[7] C. J. Fillmore and C. Baker. 313 A Frames Approach to Semantic Analysis. In *The Oxford Handbook of Linguistic Analysis*. Oxford University Press, 12 2009. ISBN 9780199544004.

[8] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang, and H. Wang. Retrieval-Augmented Generation for Large Language Models: A Survey. *CoRR*, abs/2312.10997, 2023.

[9] S. Hakimov, Y. Weiser, and D. Schlangen. Evaluating Modular Dialogue System for Form Filling Using Large Language Models. In *Proceedings of the 1st Workshop on Simulating Conversational Intelligence in Chat (SCI-CHAT 2024)*, pages 36–52, 2024.

[10] B. Hu, C. Zhao, P. Zhang, Z. Zhou, Y. Yang, Z. Xu, and B. Liu. Enabling Intelligent Interactions between an Agent and an LLM: A Reinforcement Learning Approach. *CoRR*, abs/2306.03604, 2023.

[11] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. Lost in the Middle: How Language Models Use Long Contexts. *Trans. Assoc. Comput. Linguistics*, 12:157–173, 2024.

[12] J. Maynez, S. Narayan, B. Bohnet, and R. T. McDonald. On Faithfulness and Factuality in Abstractive Summarization. In D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 1906–1919. Association for Computational Linguistics, 2020.

[13] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao. Large Language Models: A Survey. *CoRR*, abs/2402.06196, 2024.

[14] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[15] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neu-*

ral Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.

[16] K. Wang, H. Ren, A. Zhou, Z. Lu, S. Luo, W. Shi, R. Zhang, L. Song, M. Zhan, and H. Li. MathCoder: Seamless Code Integration in LLMs for Enhanced Mathematical Reasoning. *CoRR*, abs/2310.03731, 2023.

[17] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus. Emergent Abilities of Large Language Models. *Trans. Mach. Learn. Res.*, 2022, 2022.

[18] J. Weizenbaum. ELIZA - a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, 1966.

[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.