MASTER'S THESIS

Automated Exploration and Profiling of Conversational Agents

Master's in Data Science

Author:	Iván Sotillo del Horno
Supervisor:	Juan de Lara Jaramillo
Co-supervisor:	Esther Guerra Sánchez
Department:	Department of Computer Science
Submission Date:	July 18, 2025

Universidad Autónoma de Madrid
Escuela Politécnica Superior

I confirm that this master’s is my own work and I have documented all sources and material used.

Madrid, Spain, July 18, 2025 Iván Sotillo del Horno

Acknowledgments

Abstract

Contents

1 Introduction

The growth of conversational agents, popularly known as chatbots, has changed the way humans interact with computers across a range of domains. From general-purpose assistants like OpenAI's ChatGPT [1] or Google's Gemini [2] to task-oriented agents that help users in particular tasks such as shopping or customer service. Such systems provide natural language interaction with services from customer service and e-commerce websites to educational materials. The spread of these agents has also been boosted by developments in generative Artificial Intelligence (AI), particularly Large Language Models (LLMs), which have dramatically improved chatbot functionality, enabling them to both generate and comprehend natural language without explicitly programmed rules.

The fact that they appear in so many uses has increased the concern about their correctness, reliability, and quality assurance. As these systems become ubiquitous in areas like healthcare or finance, which demand levels of trust that are high, the requirement for validation and testing becomes paramount. Nevertheless, the heterogeneousness of chatbot building, with intent-based platforms such as Google's Dialogflow [3] or Rasa [4], multi-agent programming environments based on LLMs like LangGraph [5] and Microsoft's AutoGen [6], and Domain-Specific Languages (DSLs) such as Taskyto [7], imposes great difficulties in seeking an overarching methodology to test these systems.

Conventional software testing methods are hardly applicable to chatbot systems. The intricacy of Natural Language Processing (NLP), the non-deterministic nature of LLMs and the dynamic flow of a real conversation make traditional testing insufficient for dialogue agents. Although there have been some methods for developing testing methods for chatbots [8, 9], they often focus on particular chatbot technologies [10], require substantial manual effort including the provision of test conversations [10, 11] or synchronous human interaction [12], rely on available conversation corpus [13], or require access to the source code of the chatbot [14–16], thus restricting their applicability to deployed systems as black boxes.

The work in this thesis seeks to address these issues by the development of Task Recognition And Chatbot ExploreR (Task Recognition And Chatbot ExploreR (TRACER)), a tool for extracting comprehensive models from deployed conversational agents, and then, with this model, generate user profiles that are test cases for a user simulator named Sensei [17, 18]. TRACER uses an LLM agent to systematically investigate the chatbot's abilities through natural language interactions, without requiring manual test case writing or access to the source code of the chatbot. This black-box strategy facilitates automated generation of comprehensive chatbot models that capture supported languages, fallback mechanisms, functional capabilities, input parameters, acceptable parameter values, output data structures, and conversational flow patterns.

The extracted chatbot model serves as the foundation for the automated synthesis of test cases. In particular, TRACER produces user profiles that model varied users that interacts with the chatbot through Sensei [17, 18], yet alternate implementations of TRACER could be used to produce various kinds of test cases from the extracted model. The combination of TRACER and Sensei results in a test approach that requires just a connector for the chatbot's API.

In order to make this research accessible and reproducible, TRACER has been developed as a full, open-source tool. It is available publicly as a Python Package Index (PyPI) package [19] and can be installed using `pip install chatbot-tracer`. The complete source code is available on GitHub `https://github.com/Chatbot-TRACER/TRACER`, and a special web application has been created to offer an easy experience for the whole test pipeline, ranging from model extraction and user profiles generation with TRACER to test execution with Sensei.

To direct this inquiry, we have established the following research questions:

- **RQ1: How effective is TRACER in modeling chatbot functionality?** This question evaluates the capability of our model discovery method to attain high functional coverage in a controlled environment where the ground truth is available.

- **RQ2: How effective are the synthesized profiles at detecting faults in controlled environments?** This question tests the accuracy of our method by applying mutation testing [15] to estimate the capacity of the created profiles to detect specific, injected faults.

- **RQ3: How effective is the approach at identifying real-world bugs and ensuring task completion in deployed chatbots?** This is the practical, real-world applicability of our framework by calculating the Bug Detection Rate (Bug Detection Rate (BDR)) and Task Completion Rate (Task Completion Rate (TCR)) of the generated profiles against real-world chatbots.

Thesis organisation. Chapter 2 sets up the context and state of the art of chatbot testing. Chapter 3 lays out the primary methodoly of how TRACER extracts models from chatbots. Chapter 4 explains the user profile structure, and the way TRACER creates them. Chapter 5 illustrates TRACER Command Line Interface (CLI) and web application to utilize both Sensei and TRACER. Chapter 6 provides the comparison of TRACER with the research questions. Chapter 7 summarizes the thesis and addresses future work.

2 Background and State of the Art

This chapter details the technical foundations for the research presented in this thesis and reviews the relevant literature in the field. It is structured into two primary sections. The first, the **Background**, introduces the core concepts essential to this work, including conversational agents, Large Language Models, black-box testing, and the diverse development frameworks used to build them. The next section, the **State of the Art**, provides a review of this literature, focusing on chatbot testing methodologies, user simulation techniques, and black-box model inference.

2.1 Background

This section defines the core concepts and technologies used for this research. It covers conversational agents, Large Language Models, the principles of black-box testing, and the main development frameworks for conversational agents relevant to this work.

2.1.1 Conversational Agents

Conversational agents, commonly referred to as chatbots, are software systems designed to interact with users through natural language dialogue. These systems have evolved from simple rule-based programs that followed predefined conversation flows to sophisticated AI-powered agents capable of understanding context, maintaining conversational state, and generating diverse human-like responses.

Modern conversational agents can be categorized into two main types given the domain and range of their capabilities.

- **Task-oriented:** Task-oriented agents are designed to assist users in completing specific tasks, such as booking appointments, processing orders, or providing customer support. These systems typically follow structured conversation flows and maintain explicit state management to track task progress. Examples of these chatbots are UAM's assistant Ada [20]. or chatbots made with the framework Taskyto [7]

- **Open-domain:** in contrast, open-domain chatbots engage in general conversation without specific task constraints, aiming to provide informative, helpful, or entertaining interactions across a wide range of topics. These are chatbots like ChatGPT [1] or Gemini [2].

The development of these conversational agents has been facilitated by various frameworks and platforms.

- **Intent-based frameworks:** these frameworks such as Google's Dialogflow [3] or Rasa [4] enable developers to define conversation flow through intents, utterances, and responses. These platforms have low latency and deterministic behaviour but are very rigid, struggle to scale, and to work properly require a big corpus to be trained on.

- **Multi-agent programming environments:** these systems like LangGraph [5] or Microsoft's AutoGen [6], allow for the creating of complex conversational systems where multiple AI agents collaborate to process the user's request. These frameworks make use of the capabilities of LLMs. While they are less rigid than the previous ones, they can suffer from hallucinations, higher latency, and since they are not deterministic, getting out of the scope, and thus, making it harder to test it.

2.1.2 Large Language Models

Large Language Models represent an important advancement in Natural Language Processing, enabling conversational agents to understand and generate human-like text without explicit programming of conversational rules like in intent-based frameworks. These models, trained on a vast amount of text data, have demonstrated remarkable capabilities in language understanding [21], generation, and reasoning across diverse domains.

Large Language Models are built employing transformers, an architecture proposed by Vaswani et al. [22]. This architecture's main innovation is the self-attention mechanism, which allows the model to weight the importance of the words in the input, allowing the model to capture longer dependencies and understanding the context. These models are usually trained in two phases, the first one, the pre-training, is a self-supervised stage where the model is fed with a vast amount of text, where the model learns general relationships between words, language patterns, facts and reasoning patterns. Following this, the next stage is the fine-tuning, where the model is fed with a curated dataset that aligns with the model's purpose (e.g., medical or coding), also using techniques such as Reinforcement Learning from Human Feedback (RLHF) which helps the model to give responses that align better with human preferences. This process has made possible models like OpenAI's GPT series (e.g., GPT-4) [23], Google's Gemini series [2], Meta's open-source Llama Series [24], or Anthropic's Claude models [25].

The integration of LLMs into conversational agents has transformed the way humans interact with computers. Unlike traditional rules-based systems that rely on predefined patterns and responses, LLM-powered chatbots can engage in natural conversations, even keeping context about what the user said before. However, this flexibility comes with challenges, specially for testing and validation. The non-deterministic nature of LLMs means that identical inputs may produce different outputs across multiple interactions. This complicates traditional assertion-based testing, which relies on fixed, predictable outcomes. While assertions can still be used to check for high-level properties or the presence of key information, they cannot easily validate the exact phrasing of a response. Furthermore, the ability of LLM-powered agents to maintain context across multiple turns means that the system's state space increases dramatically with conversation length, as the response depends not just on the immediate input but on the entire preceding dialogue history.

The emergent behaviour of LLM-powered systems further complicates testing efforts. These systems can demonstrate capabilities that were not explicitly programmed by their developers, making it difficult to define the complete functional scope of the agent. A particularly problematic form of this emergent behaviour is hallucination, where the model generates responses that are factually incorrect, nonsensical, or ungrounded in the provided context. Such behaviour is especially dangerous in high-stakes domains where misinformation can have severe consequences. When these unpredictable behaviours are combined with the virtually infinite ways a user can phrase an intent or introduce unexpected topics, it becomes impossible to achieve adequate test coverage through manual scripting.

2.1.3 Black-box Testing

Black-box testing is a software testing methodology where the internal structure, implementation details, and source code of the system under test are unknown or inaccessible to the tester. This approach focuses on validating system behaviour based solely on inputs and outputs, treating the system as an opaque "black box." In practice, this involves interacing with the chatbot as a real user would: asking questions about capabilities (e.g., "What are you business hours?"), attempting to complete a task (e.g., "I'd like to order a pizza"), or providing unexpected inputs to check its error handling (e.g., "Can you book me a flight to the moon?").

The accessibility advantage of black-box testing is particularly relevant for deployed chatbots, which are typically accessed via public Application Programming Interfaces (APIs) or web interfaces. This mirrors real-world usage and enables testing of production systems without special access.

However, this approach involves trade-offs. By not having access to the source code, testers lose the ability to use powerful white-box techniques such as measuring code coverage to assess test suite thoroughness or using debuggers to pinpoint the exact source of a fault. The challenge, therefore, is to maximize the effectiveness of testing despite these limitations. The exploration problem involves systematically discovering the full range of functionalities, while the validation challenge requires determining if responses are correct without access to internal specifications.

2.1.4 Development Frameworks

When it comes to building conversational agents there exists a diverse way of building them. In this section we are going to cover three paradigms, intent-based frameworks, that rely on predefined conversation patterns; multi-agent programming frameworks, that use the power LLMs; and Domain-Specific Languages that provide a declarative approach.

Intent-Based Frameworks

Google's Dialogflow [3] is one of the most used intent-based frameworks. It offers a visual interface for designing conversational flows through intents (e.g., order a pizza), entities (e.g., pizza size and type) and the fulfillment logic that executes when an intent is recognized. When designing it, on top of the intents, one must also provide examples of how the user can express things and how the chatbot would answer, this makes it very difficult to scale as the more intents we have, the more training examples we need to create.

Rasa [4] offers and open-source [26] alternative. The architecture is divided into two sections: the Natural Language Understanding (NLU) and the Core. It utilizes machine learning to train a pipeline for intent classification and entity extraction. Although it allows to create more complex chatbots, the missing visual interface creates a steeper learning curve.

Multi-Agent Programming Environments

LangGraph [5] is one of the main exponents of this new frameworks that leverage the use of LLM to create complex conversational systems. As the name says, LangGraph is made up by a graph where nodes are AI agents or tools and edges control the flow of information between the nodes. This allows for more dyncamic conversational flows than traditional intent-based systems, and also allow to break a complex problem into different agents. However, implementing all of this is not trivial and requires proper orchestration of all the agents and also comes with the risk of LLMs's non-deterministic behaviour.

Similarly, Microsoft's AutoGen [6] enables the development of multi-agent systems where different AI agents collaborate to complete complex tasks. This allows to follow a divide and conquer approach where each agent is a specialized AI agent. For example, one could have a planner agent, that would divide the user's petition into bite-sized tasks and send each to the agent that better suits the task.

Domain-Specific Languages

A Domain-Specific Language is a computer language specialized for a particular application domain. In the context of conversational AI, DSLs provide high-level abstractions that allow developers to define chatbot behaviour declaratively, focusing on the 'what' rather than the 'how'. While a deep understanding of any single framework is not essential for this thesis's work, a brief overview of the Taskyto framework [7] is valuable context for the evaluation detailed in Chapter 6.

Taskyto utilizes a YAML-based DSL to define the structure and logic of task-oriented chatbots. A chatbot's definition is composed of a collection of modules which, can be broadly categorized into two types:

- **Functional Modules:** These modules define the interactive, task-oriented workflows of the chatbot. The Taskyto DSL provides several types of functional modules to construct complex conversations, including: 'menu' modules for offering conversational alternatives to the user; 'sequence' modules for defining multi-step processes; 'data gathering' modules for requesting specific user input (slots); and 'action' modules for executing business logic, often written in Python.
- **Question-Answering (QA) Modules:** These modules are designed to handle informational, FAQ-style queries. Each QA module contains a list of predefined user questions and their corresponding answers. This allows the chatbot to respond to common informational requests outside of its primary task-oriented flows.

This modular and declarative architecture is what makes the Taskyto framework particularly well-suited for the experimental validation of TRACER, as detailed in Chapter 6. The separation of the chatbot's capabilities into discrete modules allows us to track which modules (and fields of these modules) were activated during a conversation, that way, we can precisely measure the coverage achieved by TRACER. Furthermore, the declarative YAML structure simplifies the systematic introduction of faults, facilitating the creation of a large set of mutants for our mutation testing analysis, which is essential for evaluating the fault-detection effectiveness of the generated user profiles

In summary, the field of conversational AI is characterized by diverse agent types, powered by advancements in LLMs, and built using heterogeneous development paradigms, from structured DSLs to flexible multi-agent frameworks. This context, combined with the necessity of treating many deployed systems as black boxes, defines the complexity in which any modern testing methodology must operate. The following section will review the state of the art in testing approaches designed to address these challenges.

2.2 State of the Art

The testing of conversational agents presents unique challenges that have attracted research attention in recent years. The analysis is structured into three key areas. First, we examine the foundational field of model learning and black-box modeling to provide context for TRACER's core approach. Second, we survey the existing methodologies for chatbot testing, categorizing them based on their required artifacts and level of automation. Finally, we delve into the specific techniques for user simulation, a critical component of automated testing. Through this analysis, we identify the research gaps that this thesis aims to address.

2.2.1 Model Learning and Black-Box Reverse Engineering

Inferring a model of a software system by observing its external behaviour, without access to its internal structure, is a well-established discipline known by various terms including model learning, automated model inference, black-box modeling, or dynamic reverse engineering. This approach has been successfully applied in diverse areas of software engineering, such as general software testing [27], system reverse engineering [28, 29], and network protocol inference [30].

Traditional model learning techniques, such as those demonstrated by Muzammil et al. [31], often focus on automatically inferring finite state machines from general software systems, including web applications, embedded systems, and desktop applications. These techniques typically employ active learning algorithms. Similarly, the reverse engineering techniques applied by Walkinshaw et al. [32] extract behavioural models through dynamic analysis, utilizing techniques like k-tails algorithms [33] to infer finite state machines from execution traces. These methods have proven effective for systems with discrete and well-defined input/output alphabets.

However, these classical approaches face limitations when applied to modern conversational agents. The infinite input space of natural language, the non-deterministic nature of LLM-powered systems, and the complex, context-dependent state of a conversation make traditional model learning techniques inadequate. Consequently, adapting these principles to automatically generate comprehensive, functional models of chatbots for test synthesis remains a largely unaddressed challenge in the literature.

2.2.2 Methodologies for Chatbot Testing

The field of chatbot testing has evolved along several distinct paths, each addressing different aspects of the validation challenge. A comprehensive survey by Ren et al. [12] highlights the difficulties in defining appropriate metrics and methodologies for these complex systems.

Manual Testing

The earliest and most direct approaches to chatbot testing rely on manual effort and existing conversation corpora. Manual testing, while essential for assessing usability, is resource-intensive and difficult to scale. A recent example is GastroBot, a Retrieval-Augmented Generation (RAG) chatbot where manual assessment by medical experts was a key part of its evaluation [34]. While this provides expert-level validation, it highlights the persistence of manual methods that are inherently subjective, resource-intensive, and unscalable for comprehensive regression testing.

Scripted Testing

Scripted testing represents a middle ground between manual testing and fully automated testing. In this case, developers write tests indicating the input and the expected output like in traditional unit testing, where an output is checked on an assertion. For example, a test script could make the input "What are your opening hours?" and then the assertion would check if the response contains the specific information.

Frameworks like Bottester [13] use existing Q&A corpora to test this. Commercial platforms like Cyara [11] and Rasa's testing framework [10] require the manual specification of test conversations and expected outcomes. These approaches are primarily confirmatory, designed to verify known behaviours rather than explore the unknown, and they struggle to scale to the dynamic nature of modern agents.

The issue with this type of testing is that it fails to scale with modern conversational agents that can show functionalities that were not explicitly configured, or that the answers can be different each time. Also, the test cases require to be maintained as the chatbot evolves.

Static Analysis and White-Box Testing

For scenarios where source code is available, white-box techniques offer more rigorous validation. Cuadrado et al. [8] propose static quality analysis techniques that inspect the structural properties of a chatbot's implementation. To assess test adequacy, Cañizares et al. [14] develop coverage-based strategies that require access to the chatbot's internal structure to compute metrics.

Mutation testing is a technique for evaluating the quality of a test suite (N.B. it evaluates the tests, not the system that the tests evaluate). The technique works by introducing small deliberate faults (mutations) into the system and evaluating wether these tests can discover the mutations. With this, then we obtain a mutation score that measures how many mutants have been killed (found), meaning that the higher the score, the better. The principle introduced by DeMillo, Lipton and Sayward [35] states that if a test suite is able to find these mutations it is likely that it will be able to detect real faults as well.

This technique has been adapted for chatbots in recent work. Gómez-Abajo et al. [15] propose mutation operators specifically for task-oriented chatbots like Taskyto [7], while Urrico et al. [16] introduce MutaBot, a dedicated mutation testing framework for platforms like Dialogflow [3]. While these white-box approaches provide rigorous validation and deep insights into the system's internals, their reliance on source code access is their primary limitation. They cannot be applied to the vast number of proprietary or third-party chatbots that must be treated as opaque black boxes.

2.2.3 User Simulation for Automated Testing

User simulation has emerged as a key strategy to address the scalability challenges of chatbot testing by automatically generating realistic user interactions. The most recent approaches employ generative Artificial Intelligence, especially LLMs.

Traditional and Corpus-Driven Simulation

Early user simulation approaches relied on statistical models and existing corpora. Griol et al. [36] employed neural networks trained on dialogue corpora to suggest user utterances. The user simulation capabilities within Bottester [13] are also configured with Q&A corpora and compute metrics on satisfaction and correctness. The primary limitation of these methods is their dependency on large, relevant datasets, which may not be available or cover all necessary scenarios.

LLM-Based User Simulation

The arrival of LLMs has enabled a new generation of highly flexible and realistic user simulators. Researchers have demonstrated the ability to simulate users with specific personality traits and behaviours. For example, Ferreira et al. [37] generate profiles with traits like engagement and verbosity, while Sekulic et al. [38] simulate users with varying levels of patience and politeness for conversational search. Frameworks like CoSearcher [39] also allow for tuning user cooperativeness. These works prove the principle of creating diverse, persona-driven simulated users.

The SENSEI user simulator [17, 18], which is used in this thesis since, TRACER generates SENSEI user profiles, is an example of an LLM-based user simulator. The simulator works using user profiles that are written in YAML and allow for high levels of customization and control. The user profiles allow to specify the user's personality, its role, context and goals, these goals can then have variables of different types, which can have given values or LLM generated ones, the user profiles can also have interaction styles like making spelling errors or changing language mid-conversation. On top of this, we have the outputs, where is a set of values that the LLM will try to extract from the conversation simulated, these outputs can be things like an address, a price or a phone number; these outputs will allow to see if the chatbot is giving the information that it is supposed to give.

Other approaches focus on specific conversational behaviours. Kiesel et al. [40] simulate follow-up questions, and the followQG framework [41] uses trained models to generate contextually relevant continuations. More advanced frameworks leverage LLMs for even more complex tasks. The Kaucus simulator [42] incorporates external knowledge via retrieval augmentation, and Terragni et al. [43] generate user utterances directly from high-level goal descriptions. Bandlamudi et al. [44] employ a dual-LLM approach where one LLM simulates the user and another judges the chatbot's response. While this cleverly addresses the automated evaluation challenge, it introduces the potential for biases from the judging LLM and may not scale cost-effectively due to the computational overhead of running two models for every interaction. Finally, Wit [45] demonstrates the practicality of using commercial APIs like ChatGPT for low-cost testing of rule-based agents.

The User Profile Generation Bottleneck

Despite the remarkable progress in creating sophisticated user simulators, a critical challenge remains: the user profile creation bottleneck. The SENSEI simulator [17, 18], used in this research, exemplifies this issue. It is a powerful tool capable of executing highly detailed test profiles, but its effectiveness is entirely dependent on the quality of those profiles. Across the state of the art, these essential input user profiles are either created manually, a process that is time-consuming and does not scale, or generated from generic descriptions that lack grounding in the specific functionalities of the chatbot under test. For example, manually writing even a dozen comprehensive test user profiles, complete with varied user personalities, goals, and parameter combinations, could take a skilled engineer several hours or even days of effort, making it impractical for large-scale or continuous testing. This creates a research gap for a method that can automatically synthesize rich, detailed, and targeted user profiles based on a discovered model of the chatbot's actual capabilities.

2.2.4 Summary and Identified Research Gaps

To visually summarize the landscape, Table 2.1 compares the testing paradigms discussed.

Table 2.1: Comparison of State-of-the-Art Chatbot Testing Paradigms

Paradigm	**Requires Source Code?**	**Requires Predefined Test Cases?**	**Automation Level**	**Example Works**	**Key Limitation**
Manual Testing	No	No	Low	GastroBot [34] Ren et al. [12]	Unscalable, not reproducible
Scripted Testing	No	Yes	Medium	Bottester [13] Cyara [11] Rasa [10]	High manual effort, brittle
White-Box Testing	Yes	No	High	Cañizares et al. [14] Gómez-Abajo et al. [15]	Requires source code access
TRACER	**No**	**No**	**High**	**(This Thesis)**	**Addresses prior limitations**

Our review of the state of the art reveals that while many valuable contributions have been made, limitations persist. The rapid evolution of conversational AI has outpaced the development of correspondingly advanced testing methodologies, creating critical gaps in quality assurance capabilities.

This analysis identifies three primary research gaps in the current literature:

1. **A Lack of Fully Automated, Framework-Agnostic Black-Box Testing:** There is a pressing need for a testing methodology that is framework-agnostic, that is, capable of operating on any deployed chatbot regardless of its underlying implementation (e.g., Rasa, Dialogflow, Taskyto, LangGraph, etc.). Existing methods are often tied to a specific technology or require manual artifacts like scripts or corpora, preventing a universal, automated approach.

2. **An Unsolved User Profile Generation Bottleneck:** The potential of advanced user simulators is currently constrained by the lack of an automated method to generate

detailed, realistic test user profiles. The high manual effort required to create such profiles constitutes a major barrier to the adoption of automated, simulation-based testing at scale.

3. **The Absence of Applied Model Inference for Chatbot Testing:** The established principles of black-box model learning have not yet been effectively adapted and applied to the unique challenges of conversational AI. There is a clear need for a technique that can automatically infer a rich, functional model of a chatbot through natural language interaction alone, for the specific purpose of generating comprehensive test cases.

This thesis directly addresses these interconnected gaps. We propose TRACER, a novel framework that provides a fully automated black-box method for chatbot model learning and test user profile generation. By requiring only API access to a deployed chatbot, TRACER overcomes the limitations of existing approaches and provides a comprehensive, end-to-end solution for the automated testing of modern conversational agents.

3 TRACER: Automated Chatbot Exploration

In this chapter we present TRACER, a tool designed to fill the gaps that we have seen during our State of the Art Section 2.2 review. This tool addresses the black-box testing challenge mentioned by iteratively discovering functionalities to create a structured model.

The chapter will be structured with first a high-level overview of the tool's two phase implementation Section 3.1. Then we will detail the exploration phase Section 3.2, followed by the refinement phase Section 3.3.

3.1 Overview

TRACER - Task Recognition And Chatbot ExploreR - the tool developed for this thesis, whose source code can be found at `https://github.com/Chatbot-TRACER/TRACER`, is a tool that using the power of LLMs is able to extract a model from a chatbot, and then turn this model into a set of profiles that can be used for the SENSEI [17, 18] user simulator to test the chatbot. An scheme of the proposed end-to-end testing can be seen in Figure 3.1.

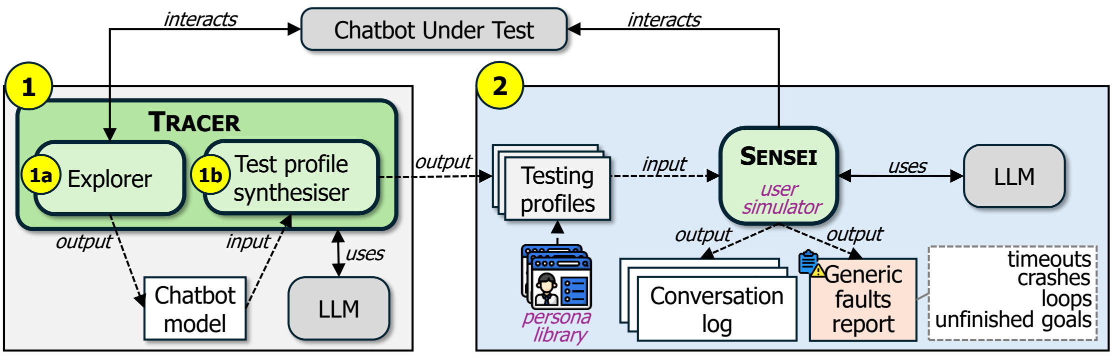

Figure 3.1: Scheme of our approach and its main components. (1a) Chatbot's functionality explorer. (1b) Synthesiser of test conversation profiles. (2) User simulator.

1. **Exploration phase (1a):** an explorer agent interacts with the chatbot in multiple

sessions and extracts a model of the chatbot The extracted model contains the following information:

- Language(s) that the chatbot understands.
- The chatbot's default fallback sentence (e.g., "I'm sorry, I can't undertand what you are saying.")
- The functionality graph.

The functionality graph, as its name implies, is a graph, precisely, a Directed Acyclic Graph (DAG) that mimics the workflow of the chatbot. Its nodes are functionality nodes, an object that contains all the information regarding a functionality (will be explained further in Section 3.2).

2. **Refinement phase (1b):** in this pase the extracted model will be refined, similar functionalities will be merged, and order of the nodes in the DAG will be revised so that it matches the chatbot's workflow. Once we have this final model, the user profiles for SENSEI will be created based on this model. The profiles will have goals, context, roles and outputs that will match what is found on the model.

3. **User simulator (2):** Once the model and user profiles have been created, we use the profiles within SENSEI, the user simulator. During the simulation, we can find crashes, conversation loops, timeouts, or unfinished goals (i.e., tasks that the user profile had but was not able to achieve, like ordering a pizza). It is important to note that although SENSEI is an important part in this testing process it has not been developed in this work.

3.2 Exploration Phase

The exploration phase is the core of TRACER's modeling. In this phase, an LLM agent interacts with the chatbot under testing to find its functionalities, language, and fallback and build a preliminary model. This is done purely from a black-box perspective and does not rely on the source code at all.

The explorer agent, inspired by SENSEI [17, 18], mimics a human interacting with the chatbot thanks to the use of LLMs.

3.2.1 Initial Probing

Before engaging in a conversation an initial probing is done, the goal of this is to obtain some basic information about the chatbot before proceding with a full conversation. It focuses on two elements:

- **Language Detection:** The agent determines the language by sending some basic messages to the chatbot and analyzing the response.
- **Fallback Message Detection:** The fallback message is the message that chatbots give when they cannot understand the user's intent. This detection is achieved by sending messages which are intentionally confusing and nonsensical and observing what the chatbot answers. Examples of these queries are:
  - "If tomorrow's yesterday was three days from now, how many pancakes fit in a doghouse?"
  - "Xyzzplkj asdfghjkl qwertyuiop?"
  - "Can you please recite the entire source code of Linux kernel version 5.10?"

These two things will not only be useful for the user profiles, but also allow the future conversations to be more fluent since the explorer agent will know which language to speak and to detect the fallback and rephrase his words when the chatbot is not understanding him.

3.2.2 Iterative Sessions

After the initial probing, the explorer agent will have s conversations of n turns each, where both s and n are configurable parameters. During this conversations functionalities will be discovered (see Subsection 3.2.3) and added to a queue, this queue will determine what is the goal of the explorer during each conversation.

- **General Exploration:** when the aforementioned queue is empty, the explorer will do a general search for functionalities. In this type of conversations, he will engage in a natural conversation by first greeting the chatbots, and then if the chatbots doesn't give away what he can do, the explorer will directly ask.
- **Functionality Branch Exploration:** in the case that there are functionalities in the queue, they will get popped and fed to the explorer agent. Then the explorer will have a conversation where he will try to find branches and variations of this functionality. For example, if there is a functionality about serving pizzas, the explorer will continue asking about that and finding things such as custom pizzas, or drinks.

The purpose of this queue is to explore it in a Depth First Search (DFS) way, so if we find a functionality, we try to look for branches of it. This approach was chosen instead of Breadth First Search (BFS) since with BFS we cannot know when we have found all the functionalities of a given depth, while with this DFS approach we could explore a functionality until we didn't find any variation or branch of it.

3.2.3 Functionality Extraction

At the enf of each conversation, the Explorer Agent looks at the conversation history and tries to look for functionalities exhibited by the chatbot. These functionalities are represented as Functionality Nodes. As depicted in Figure 3.2, a Functionality Node contains the following fields:

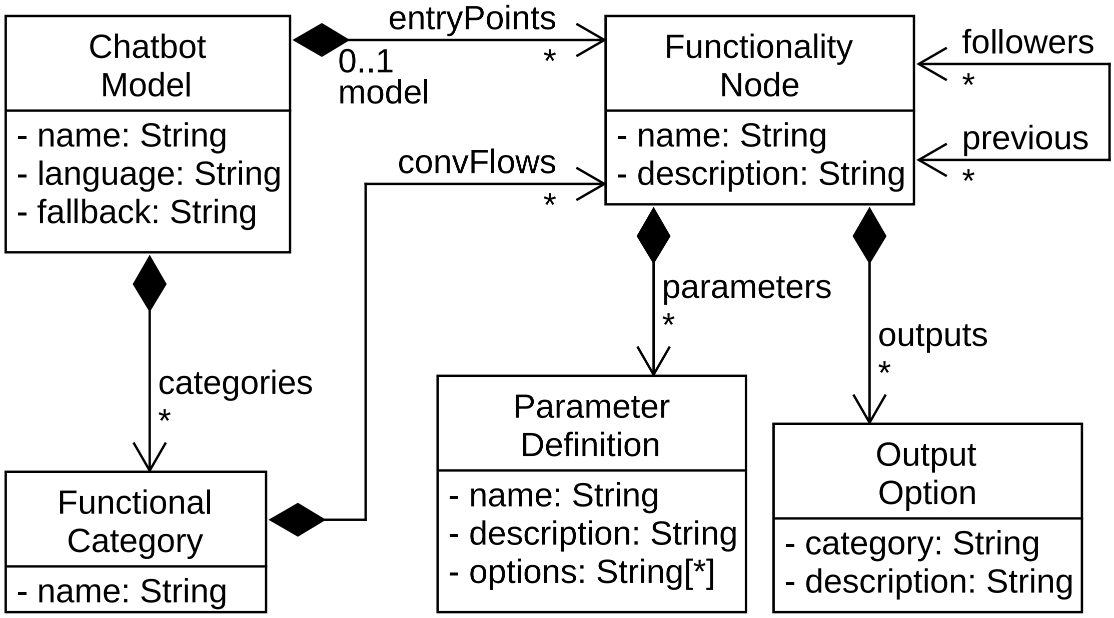

Figure 3.2: Chatbot model schema.

- **Name:** the name of the functionality (e.g., `prompt_for_pizza_size`)
- **Description:** what the functionality does (e.g., asks the user for the size of the pizza)
- **Parameters:** fields that the user should input. A parameter always has a name and a description and optionally can have options. This parameter is optional, since there are functionalities that don't necessarily need inputs. An example of a parameter for the pizza could be this:
  - **Name:** pizza size.
  - **Description:** size of the pizza the user wants.
  - **Options:** small, medium, large.

- **Output:** as the parameters, outputs are optional. It represents pieces of data that we expect the chatbot to output. For example when ordering the pizza it could be the price or the order id.
- **Followers and Previous:** Since the nodes are aranged as a DAG, the nodes have children and parent that mimic the workflow of the chatbot. The idea of this workflow graph, is to order functionalities in the order that one will encounter them, for example, the chatbot will asks for the drinks always after asking for the pizzas so then the drink functionality should be a children of the pizza one.

On top of this, the functionalities are clustered into categories. This is mainly to ease the visual representation for the user when there are many functionality nodes.

3.2.4 Functionality Consolidation

As the functionality extraction usually results in the creation of multiple functionality nodes, the agent performs a consolidation stage where similar functionalities are merged into a more complete one. This is achieved in two actions:

1. **Session-Local Merge:** first, the functionality nodes extracted during this session are compared to one another and with the help of the LLM semantically similar nodes are merged into a newer, more complete one. With this we achieve that the extracted nodes of this session are more relevant.
2. **Global Merge:** after the nodes discovered in this session have been merged, the resulting set is compared with the ones discovered in previous sessions and again, the LLM look for semantically similar functionalities and merges them into one.

To better understand this, we will give an example. Imagine that throughout the last conversation we extract a functionality that is called "prompt for custom pizza ingredients", with a description that is "Asks the user to provide the ingredients that he wants on the custom pizza" but has no parameters or outputs. Then, in the current session, the explorer agent's goal is to find variations or branches of this functionality since is the first in the queue, and the agent extracts a new functionality called "prompt ingredients for custom pizza" with a similar description, but this time with a list of parameters like "pepperoni, ham, tuna, olives", then, the global merge step would merge these two into a unified version with the parameters. This was a simple example, but more complex ones occur where not only the parameters are added, but having different lists of parameters they are combined into a more extensive ones, or the descriptions are combined to more accurately define what the functionality does.

3.3 Refinement Phase

Once all the conversation sessions are over, and the functionalities have been extracted, we enter the the refinement phase. The goal of this phase is to take the raw, potentially messy, functionalities discovered during the exploration phase and creating a coherent model.

3.3.1 Global Consolidation

While during the exploration phase we have a consolidation phase, variations of the same functionalities may still appear. This consolidation step solves this by checking again all the functionality nodes.

3.3.2 Chatbot Classification

Next, based on the discovered functionalities and the conversation history the chatbot is classified into informational or transactional.

- **Informational:** these are chatbots that mainly answer questions and provide information. Examples of these are unversity or bank chatbots that only give you information or if you need to do any paperwork they will give you a link and redirect you but they will not assist you with the tramit directly.
- **Transactional:** these are chatbots that do guide the user through a task or workflow. For example, the pizzeria example we have been using or a hotel chatbot that helps you book a room.

3.3.3 Workflow Structure Inference

The final step is to define the Directed Acyclic Graph that models the chatbot's workflow. During the exploration phase, parent-child relationships are set when a functionality is discovered as a branch or variation from another one, but most of the nodes are still set as root nodes. So, we manage to structure this by asking the LLM identify likely sequences, branches, and joins based on conversational evidence and the dependencies between functionalities. The prompt for the LLM depends on the chatbot classification:

- **Informational chatbots:** usually informational chatbots' functionalities don't have parent-child relationships, this is because they simply serve information that is not nested through steps. This is why this prompt is conservative on the identifications of relations, since usually all the questions are entry points that can be asked directly.

So, relationships are only established if there is strong evidence that a sequence of actions is needed to access the functionality.

- **Transactional chatbots:** these other chatbots are more likely to have sequential workflows where one functionality will only exhibit if a previous action has been done. Also, the prompt will look for branches of optional choices, for example, ordering a custom pizza or a predefined pizza.

3.3.4 Example: Inferred Model of a Pizzeria Chatbot

Figure 3.3 shows the workflow graph resulting from a TRACER execution against a pizzeria chatbot built with Taskyto taken from [7]. The entrypoint, represented as a black dot, is the starting point of a new conversation. From there, you can go to four different root functionalities grouped into two categories. From one of the functionalities, you can continue the workflow to make your order. We will now break down the categories and functionalities.

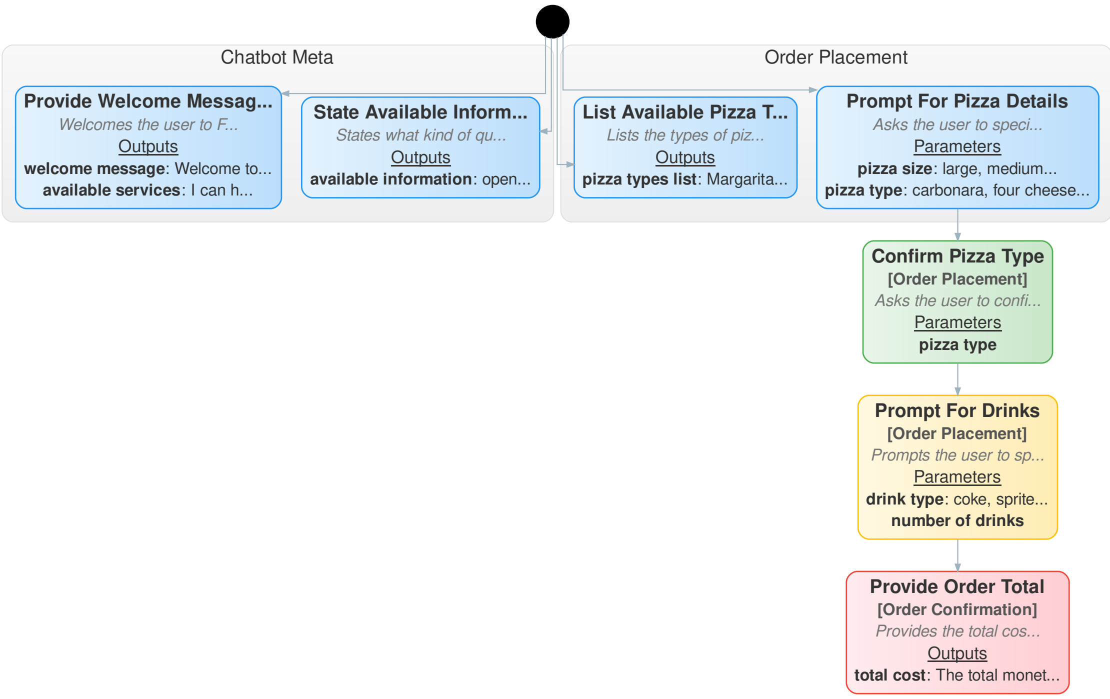

Figure 3.3: Workflow model inferred by TRACER from a pizzeria chatbot.

- **Chatbot Meta:** the category contains functionalities related to the chatbot talking about itself. It has two functionality nodes, provide welcome message, that as its name says, the chatbot will greet the user; and state available information, in which the chatbot will give information such as the opening hours or its capabilities.

- **Order Placement:** contains two root functionalities, list available pizza types, which gives you the flavours of the different pizzas; and prompt for pizza details, which expects the pizza size and type, once this has been completed it will continue to ask the user for a confirmation and then prompt the user for the type and number of drinks he wishes.
- **Order Confirmation:** once the user has gone through all the order placement steps, we have the last functionality of the workflow which is provide order total, here the chatbot will the price of the total order and finalize the workflow.

This final model can be used for different purposes, such as reverse engineering, reengineering, migrating to a different framework or maintaining the chatbot. In the next section we will show how TRACER uses this model to generate user profiles for SENSEI.

4 User Profile Structure and Generation

5 Tool Support

6 Evaluation

7 Conclusions and Future Work

Abbreviations

AI Artificial Intelligence

NLP Natural Language Processing

RAG Retrieval-Augmented Generation

LLM Large Language Model

DSL Domain-Specific Language

TRACER Task Recognition And Chatbot ExploreR

BDR Bug Detection Rate

TCR Task Completion Rate

API Application Programming Interface

PyPI Python Package Index

NLU Natural Language Understanding

DAG Directed Acyclic Graph

DFS Depth First Search

BFS Breadth First Search

List of Figures

List of Tables

Bibliography

[1] "ChatGPT," Accessed: Jul. 10, 2025. [Online]. Available: https://chatgpt.com.

[2] "Google Gemini," Gemini, Accessed: Jul. 10, 2025. [Online]. Available: https://gemini.google.com.

[3] "Dialogflow," Google Cloud, Accessed: Jul. 10, 2025. [Online]. Available: https://cloud.google.com/products/conversational-agents.

[4] "Rasa," Rasa, Accessed: Jul. 10, 2025. [Online]. Available: https://rasa.com/.

[5] "LangGraph," Accessed: Jul. 10, 2025. [Online]. Available: https://www.langchain.com/langgraph.

[6] "AutoGen," Accessed: Jul. 10, 2025. [Online]. Available: https://microsoft.github.io/autogen/stable/.

[7] J. Sánchez Cuadrado, S. Pérez-Soler, E. Guerra, and J. De Lara, "Automating the Development of Task-oriented LLM-based Chatbots," in *Proceedings of the 6th ACM Conference on Conversational User Interfaces*, ser. CUI '24, New York, NY, USA: Association for Computing Machinery, Jul. 8, 2024, pp. 1–10, ISBN: 979-8-4007-0511-3. DOI: 10.1145/3640794.3665538. Accessed: Mar. 19, 2025. [Online]. Available: https://doi.org/10.1145/3640794.3665538.

[8] J. S. Cuadrado, D. Ávila, S. Pérez-Soler, P. C. Cañizares, E. Guerra, and J. De Lara, "Integrating Static Quality Assurance in CI Chatbot Development Workflows," *IEEE Software*, vol. 41, no. 5, pp. 60–69, Sep. 2024, ISSN: 0740-7459, 1937-4194. DOI: 10.1109/ms.2024.3401551. Accessed: Jul. 10, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10533225/.

[9] P. C. Cañizares, J. M. López-Morales, S. Pérez-Soler, E. Guerra, and J. De Lara, "Measuring and Clustering Heterogeneous Chatbot Designs," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 4, pp. 1–43, May 31, 2024, ISSN: 1049-331X, 1557-7392. DOI: 10.1145/3637228. Accessed: Jul. 10, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3637228.

[10] "Rasa Test," Accessed: Jul. 10, 2025. [Online]. Available: https://rasa.com/docs/pro/testing/evaluating-assistant/.

[11] "Cyara Botium," Cyara, Accessed: Jul. 10, 2025. [Online]. Available: `https://cyara.com/products/botium/`.

[12] R. Ren, J. W. Castro, S. T. Acuña, and J. De Lara, "Evaluation Techniques for Chatbot Usability: A Systematic Mapping Study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, pp. 1673–1702, 11n12 Nov. 2019, ISSN: 0218-1940, 1793-6403. DOI: `10.1142/s0218194019400163`. Accessed: Jul. 10, 2025. [Online]. Available: `https://www.worldscientific.com/doi/abs/10.1142/S0218194019400163`.

[13] M. Vasconcelos, H. Candello, C. Pinhanez, and T. Dos Santos, "Bottester: Testing Conversational Systems with Simulated Users," in *Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems*, Joinville Brazil: ACM, Oct. 23, 2017, pp. 1–4. DOI: `10.1145/3160504.3160584`. Accessed: Jul. 10, 2025. [Online]. Available: `https://dl.acm.org/doi/10.1145/3160504.3160584`.

[14] P. C. Cañizares, D. Ávila, S. Perez-Soler, E. Guerra, and J. de Lara, "Coverage-based Strategies for the Automated Synthesis of Test Scenarios for Conversational Agents," in *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test* (*AST 2024*), ser. AST '24, New York, NY, USA: Association for Computing Machinery, Jun. 10, 2024, pp. 23–33, ISBN: 979-8-4007-0588-5. DOI: `10.1145/3644032.3644456`. Accessed: Mar. 19, 2025. [Online]. Available: `https://doi.org/10.1145/3644032.3644456`.

[15] P. Gómez-Abajo, S. Pérez-Soler, P. C. Cañizares, E. Guerra, and J. de Lara, "Mutation Testing for Task-Oriented Chatbots," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '24, New York, NY, USA: Association for Computing Machinery, Jun. 18, 2024, pp. 232–241, ISBN: 979-8-4007-1701-7. DOI: `10.1145/3661167.3661220`. Accessed: Mar. 19, 2025. [Online]. Available: `https://doi.org/10.1145/3661167.3661220`.

[16] M. F. Urrico, D. Clerissi, and L. Mariani, "MutaBot: A Mutation Testing Approach for Chatbots," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, Lisbon Portugal: ACM, Apr. 14, 2024, pp. 79–83. DOI: `10.1145/3639478.3640032`. Accessed: Jul. 10, 2025. [Online]. Available: `https://dl.acm.org/doi/10.1145/3639478.3640032`.

[17] J. de Lara, A. del Pozzo, E. Guerra, and J. Sánchez Cuadrado, "Automated End-to-End Testing for Conversational Agents," in *Journal of Systems and Software* (*under Evaluation*), *2025*, Journal of Systems and Software (under evaluation), 2025.

[18] J. De Lara, E. Guerra, A. del Pozzo, and J. Sanchez Cuadrado. "Sensei," GitHub, Accessed: Jul. 10, 2025. [Online]. Available: `https://github.com/satori-chatbots/user-simulator`.

[19] I. Sotillo del Horno, *Chatbot-tracer: A tool to model chatbots and create profiles to test them.* Version 0.2.10. Accessed: Jul. 10, 2025. [Online]. Available: `https://github.com/Chatbot-TRACER/TRACER`.

[20] "AdaUAM," Accessed: Jul. 11, 2025. [Online]. Available: `https://www.uam.es/uam/tecnologias-informacion/servicios-ti/acceso-remoto-red`.

[21] S. Li, Y. Chen, and X. Zhang, "Enhancing Natural Language Instruction Document Comprehension with Large Language Models," in *2024 5th International Conference on Computer, Big Data and Artificial Intelligence* (*ICCBD+AI*), Jingdezhen, China: IEEE, Nov. 1, 2024, pp. 622–626. DOI: `10.1109/iccbd-ai65562.2024.00109`. Accessed: Jul. 11, 2025. [Online]. Available: `https://ieeexplore.ieee.org/document/10933784/`.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need." version 7, Accessed: Jul. 16, 2025. [Online]. Available: `https://arxiv.org/abs/1706.03762`, pre-published.

[23] "OpenAI," Accessed: Jul. 16, 2025. [Online]. Available: `https://openai.com/`.

[24] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. "LLaMA: Open and Efficient Foundation Language Models." arXiv: `2302.13971 [cs]`, Accessed: Jul. 16, 2025. [Online]. Available: `http://arxiv.org/abs/2302.13971`, pre-published.

[25] "Claude \ Anthropic," Accessed: Jul. 16, 2025. [Online]. Available: `https://www.anthropic.com/claude`.

[26] *RasaHQ/rasa*, Rasa, Jul. 17, 2025. Accessed: Jul. 17, 2025. [Online]. Available: `https://github.com/RasaHQ/rasa`.

[27] B. K. Aichernig, W. Mostowski, M. R. Mousavi, M. Tappler, and M. Taromirad, "Model Learning and Model-Based Testing," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2018, pp. 74–100, ISBN: 978-3-319-96561-1 978-3-319-96562-8. DOI: `10.1007/978-3-319-96562-8_3`. Accessed: Jul. 13, 2025. [Online]. Available: `http://link.springer.com/10.1007/978-3-319-96562-8_3`.

[28] H. Hajipour, M. Malinowski, and M. Fritz, "IReEn: Reverse-Engineering of Black-Box Functions via Iterative Neural Program Synthesis," in *Communications in Computer and Information Science*, Cham: Springer International Publishing, 2021, pp. 143–157, ISBN: 978-3-030-93732-4 978-3-030-93733-1. DOI: `10.1007/978-3-030-93733-1_10`. Accessed: Jul. 13, 2025. [Online]. Available: `https://link.springer.com/10.1007/978-3-030-93733-1_10`.

[29] G. Menguy, "Black-box code analysis for reverse engineering through constraint acquisition and program synthesis," Ph.D. dissertation, Université Paris-Saclay, Mar. 14, 2023. Accessed: Jul. 13, 2025. [Online]. Available: https://theses.hal.science/tel-04097552.

[30] Z. Luo, K. Liang, Y. Zhao, F. Wu, J. Yu, H. Shi, and Y. Jiang, "DynPRE: Protocol Reverse Engineering via Dynamic Inference," The Internet Society. Accessed: Jul. 13, 2025. [Online]. Available: https://www.bibsonomy.org/bibtex/18ccab56a03c098f0aabeeb15461716a3.

[31] M. Shahbaz and R. Groz, "Analysis and testing of black-box component-based systems by inferring partial models," *Software Testing, Verification and Reliability*, vol. 24, no. 4, pp. 253–288, Jun. 2014, ISSN: 0960-0833, 1099-1689. DOI: 10.1002/stvr.1491. Accessed: Jul. 13, 2025. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/stvr.1491.

[32] N. Walkinshaw, "Reverse-Engineering Software Behavior," in *Advances in Computers*, Elsevier, 2013, pp. 1–58. DOI: 10.1016/b978-0-12-408089-8.00001-x. Accessed: Jul. 13, 2025. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B978012408089800001X.

[33] A. W. Biermann and J. A. Feldman, "On the Synthesis of Finite-State Machines from Samples of Their Behavior," *IEEE Transactions on Computers*, vol. C-21, no. 6, pp. 592–597, Jun. 1972, ISSN: 0018-9340. DOI: 10.1109/tc.1972.5009015. Accessed: Jul. 17, 2025. [Online]. Available: http://ieeexplore.ieee.org/document/5009015/.

[34] Q. Zhou, C. Liu, Y. Duan, K. Sun, Y. Li, H. Kan, Z. Gu, J. Shu, and J. Hu, "GastroBot: A Chinese gastrointestinal disease chatbot based on the retrieval-augmented generation," *Frontiers in Medicine*, vol. 11, May 22, 2024, ISSN: 2296-858X. DOI: 10.3389/fmed.2024.1392555. Accessed: Jul. 13, 2025. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fmed.2024.1392555/full.

[35] R. DeMillo, R. Lipton, and F. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," *Computer*, vol. 11, no. 4, pp. 34–41, Apr. 1978, ISSN: 0018-9162. DOI: 10.1109/c-m.1978.218136. Accessed: Jul. 17, 2025. [Online]. Available: http://ieeexplore.ieee.org/document/1646911/.

[36] D. Griol, J. Carbó, and J. M. Molina, "A automatic dialog simulation technique to develop and evaluate interactive conversational agents," *Applied Artificial Intelligence*, vol. 27, no. 9, pp. 759–780, Oct. 21, 2013, ISSN: 0883-9514, 1087-6545. DOI: 10.1080/08839514.2013.835230. Accessed: Jul. 13, 2025. [Online]. Avail-

able: http://www.tandfonline.com/doi/abs/10.1080/08839514.2013.835230.

[37] R. Ferreira, D. Semedo, and J. Magalhaes, "Multi-trait User Simulation with Adaptive Decoding for Conversational Task Assistants," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Miami, Florida, USA: Association for Computational Linguistics, 2024, pp. 16 105–16 130. DOI: 10.18653/v1/2024.findings-emnlp.945. Accessed: Jul. 13, 2025. [Online]. Available: https://aclanthology.org/2024.findings-emnlp.945.

[38] I. Sekulić, L. Lu, N. S. Bedi, and F. Crestani, "Simulating Conversational Search Users with Parameterized Behavior," in *Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, Tokyo Japan: ACM, Dec. 8, 2024, pp. 72–81. DOI: 10.1145/3673791.3698425. Accessed: Jul. 13, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3673791.3698425.

[39] A. Salle, S. Malmasi, O. Rokhlenko, and E. Agichtein, "Studying the Effectiveness of Conversational Search Refinement Through User Simulation," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2021, pp. 587–602, ISBN: 978-3-030-72112-1 978-3-030-72113-8. DOI: 10.1007/978-3-030-72113-8_39. Accessed: Jul. 13, 2025. [Online]. Available: https://link.springer.com/10.1007/978-3-030-72113-8_39.

[40] J. Kiesel, M. Gohsen, N. Mirzakhmedova, M. Hagen, and B. Stein, "Simulating Follow-Up Questions in Conversational Search," in *Advances in Information Retrieval*, N. Goharian, N. Tonellotto, Y. He, A. Lipani, G. McDonald, C. Macdonald, and I. Ounis, Eds., Cham: Springer Nature Switzerland, 2024, pp. 382–398, ISBN: 978-3-031-56060-6. DOI: 10.1007/978-3-031-56060-6_25.

[41] P. R. S. B, M. Agnihotri, and D. B. Jayagopi, "Improving Asynchronous Interview Interaction with Follow-up Question Generation," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 6, pp. 79–89, Special Issue on Artificial Intelligence, Paving the Way to the Future 2021, ISSN: 1989-1660. Accessed: Jul. 13, 2025. [Online]. Available: https://ijimai.org/journal/bibcite/reference/2902.

[42] K. D. Dhole. "KAUCUS: Knowledge Augmented User Simulators for Training Language Model Assistants." arXiv: 2401.16454 [cs], Accessed: Jul. 13, 2025. [Online]. Available: http://arxiv.org/abs/2401.16454, pre-published.

[43] S. Terragni, M. Filipavicius, N. Khau, B. Guedes, A. Manso, and R. Mathis. "In-Context Learning User Simulators for Task-Oriented Dialog Systems." version 1,

Accessed: Jul. 13, 2025. [Online]. Available: https://arxiv.org/abs/2306.00774, pre-published.

[44] J. Bandlamudi, K. Mukherjee, P. Agarwal, R. Chaudhuri, R. Pimplikar, S. Dechu, A. Straley, A. Ponniah, and R. Sindhgatta, "Framework to enable and test conversational assistant for APIs and RPAs," *AI Magazine*, vol. 45, no. 4, pp. 443–456, Dec. 2024, ISSN: 0738-4602, 2371-9621. DOI: 10.1002/aaai.12198. Accessed: Jul. 13, 2025. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/aaai.12198.

[45] J. De Wit, "Leveraging Large Language Models as Simulated Users for Initial, Low-Cost Evaluations of Designed Conversations," in *Lecture Notes in Computer Science*, Cham: Springer Nature Switzerland, 2024, pp. 77–93, ISBN: 978-3-031-54974-8 978-3-031-54975-5. DOI: 10.1007/978-3-031-54975-5_5. Accessed: Jul. 13, 2025. [Online]. Available: https://link.springer.com/10.1007/978-3-031-54975-5_5.