DIAGRAPH: An Open-Source Graphic Interface for Dialog Flow Design

Dirk Väth* **Lindsey Vanderlyn*** **Ngoc Thang Vu**
University of Stuttgart, Germany
{vaethdk|vanderly|thang.vu}@ims.uni-stuttgart.de

Abstract

In this work, we present DIAGRAPH, an open-source [1] graphical dialog flow editor built on the ADVISER toolkit. Our goal for this tool is threefold: 1) To support subject-experts to intuitively create complex and flexible dialog systems, 2) To support rapid prototyping of dialog system behavior, e.g., for research, and 3) To provide a hands-on test bed for students learning about dialog systems. To facilitate this, DIAGRAPH aims to provide a clean and intuitive graphical interface for creating dialog systems without requiring any coding knowledge. Once a dialog graph has been created, it is automatically turned into a dialog system using state of the art language models. This allows for rapid prototyping and testing. Dialog designers can then distribute a link to their finished dialog system or embed it into a website. Additionally, to support scientific experiments and data collection, dialog designers can access chat logs. Finally, to verify the usability of DIAGRAPH, we performed evaluation with subject-experts who extensively worked with the tool and users testing it for the first time, receiving above average System Usability Scale (SUS) scores from both (82 out 100 and 75 out of 100, respectively). In this way, we hope DIAGRAPH helps reduce the barrier to entry for creating dialog interactions.

1 Introduction

Dialog systems have gained much attention in recent years as they offer a convenient way for users to access information in a more personalized manner, or accomplish tasks through an intuitive natural language interface. Traditionally, they need to understand user input, track information across multiple dialog turns and choose a system response. These tasks can either be performed in a modular manner or as an end-to-end approach where user input is directly mapped to system output. However, regardless of approach, state-of-the-art approaches to dialog systems largely rely on neural methods (Chen et al., 2017). While these methods have generally shown improvements to dialog performance and generalizability across multiple dialog domains, they rely on the availability of sufficient training data which can be work-intensive and expensive to collect. Additionally, their decision-making often remains a black-box, which can make them unsuitable for highly sensitive domains, e.g., medical or legal contexts, where it is critical that dialog designers can maintain careful control over the system's behavior.

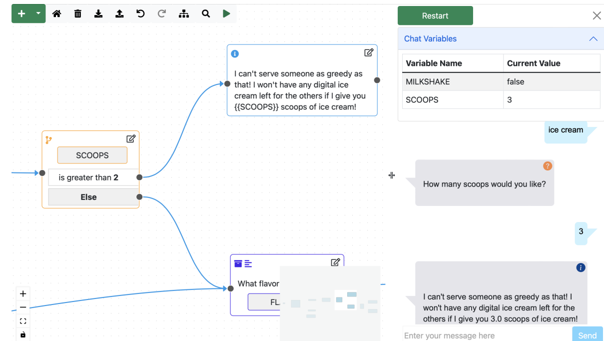

Figure 1: Left: Dialog editor with tutorial graph; Right: Debugging window with chat and variable explorer.

Although multiple toolkits have been developed to speed up the creation of dialog systems (Bohus and Rudnicky, 2009; Lison and Kennington, 2016; Ultes et al., 2017; Zhu et al., 2020; Li et al., 2020), and this has accelerated progress in the field, to the best of our knowledge all toolkits currently used for research require users to be able to write code in order to customize pre-implemented models for new domains or datasets.

However, this overlooks the fact that technical experts are not the only parties interested in creating dialog systems. Domain experts, or researchers from other disciplines, who may not have a technical background, may also be interested in using

*Both authors contributed equally
[1] https://github.com/DigitalPhonetics/diagraph

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 3: System Demonstrations, pages 136–143
July 10-12, 2023 ©2023 Association for Computational Linguistics

dialog systems for a variety of goals. For example, domain experts might want to design a controlled production system. Researchers, e.g. psychologists or linguists, might want to design pilot studies or easily conduct research on interactions with a dialog system. Additionally, an interface to quickly setup data collection, e.g., to bootstrap an AI based dialog system or quickly iterate on user feedback, can accelerate dialog research or deployment.

To this end, we propose DIAGRAPH (figure 1): an open-source, graphical software for designing dialog flow graphs which are automatically converted into dialog systems. In this way, we hope it will serve as a good alternative to closed-source commercial options. Our goal for this tool is threefold: 1) To support subject-area experts to intuitively create and deploy complex and flexible dialog systems, 2) To support users, e.g., researchers, to easily and rapidly prototype and evaluate dialog systems or user behaviour, and 3) To provide a hands-on test bed for students learning about dialog systems. We evaluate DIAGRAPH with all three user groups and demonstrate its usability and practical usefulness by 1) Working with the department for business travel a University, 2) Performing a usability study where participants were asked to design or alter a dialog system in less than 30 minutes, and 3) Teaching a workshop on dialog systems and programming concepts to high school students.

2 Related Work

2.1 Dialog System Toolkits

In recent years, several toolkits have been developed to aid in the creation of dialog systems. Toolkits like RavenClaw (Bohus and Rudnicky, 2009), provide basic functionality, letting developers focus solely on describing the dialog task control logic. More recent toolkits include OpenDial (Lison and Kennington, 2016) – incorporating probabilistic rules and integration of external modules – and PyDial (Ultes et al., 2017) – a multi-domain dialog toolkit, for building modular dialog systems. Additionally, ConvLab (Lee et al., 2019; Zhu et al., 2020) and ADVISER (Ortega et al., 2019; Li et al., 2020) are modern toolkits, incorporating state of the art models in their implementations. In its recent update (Zhu et al., 2020), the developers also integrated evaluation and analysis tools to help developers debug dialog systems.

However, while these toolkits have accelerated dialog system research, their code-based interfaces can be prohibitively complex for non-technical users and do not lend themselves to quick prototyping or education. While the ADVISER toolkit still uses a code-based interface for designing dialog systems, we chose to use it as the backend for DIAGRAPH due to the low overhead and flexibility of the toolkit.

2.2 Dialog Flow Designers

Recently, there has been research into efficiently navigating dialogs based on flow diagrams, such as work by Raghu et al. (2021), who investigate learning dialog policies from troubleshooting flowcharts and Shukla et al. (2020) who learn a dialog policy from a dialog flow tree designed using Microsoft's graphic design tool. However, we are not aware of any well-fleshed out open-source tools for creating such graphs. Even though such dialog designer tools have become popular in industry with companies including Microsoft[2], Google[3] and Amazon[4] offering such services, the lack of open source alternatives impedes research around such graph/flow-based systems. Therefore, by providing a freely available alternative, we hope to make dialog system creation easier for researchers, domain experts, and educators.

To the best of our knowledge, the only open-source graphic-based dialog designer was created by Koller et al. (2018) as an educational tool to interface with Lego Mindstorms robots. While the authors show that it was well received by school and university students, its narrow scope does not address the needs of users such as subject-area experts or researchers. To this end, we create and publish DIAGRAPH: a general-purpose, open-source, graphical dialog designer built on the ADVISER toolkit.

3 Design Principles

The goal of the dialog designer is to allow for the intuitive creation of dialog systems with any level of technical knowledge. To this end, we design DIAGRAPH around the following principles:

User Friendliness To accommodate all three user groups, the software needs to be intuitive to operate without any previous programming experience. Thus, we try to keep interactions simple,

[2] https://powervirtualagents.microsoft.com

[3] https://cloud.google.com/dialogflow?hl=de

[4] https://aws.amazon.com/de/lex/chatbot-designer/

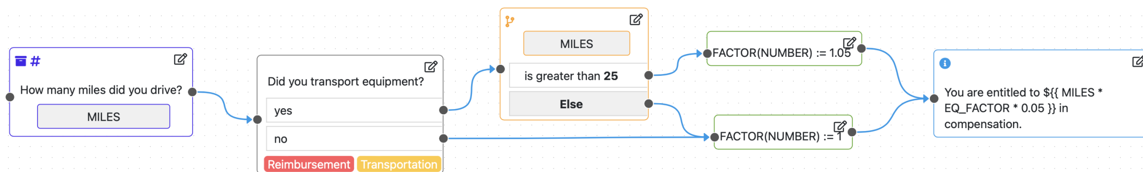

Figure 2: Example of each type of node. From left to right Variable Node (purple outline), User Response Nodes (black outline) Logic Node (orange outline), Variable Update Nodes (green outline), and Information Node (blue outline). The information node displays an example of the template syntax, using two variables in a mathematical expression to output a personalized message

e.g., dragging a connection from one node to the next to define dialog flow. We additionally try to only use icons and keyboard/mouse shortcuts commonly used in other programs, e.g., right clicking to get a context menu or typing `ctrl+f` to search. Finally, we include several features to help keep an overview even in complex graphs, e.g., a mini-map, text search and tags.

Flexibility To meet the needs of all user groups, we provide features for designing arbitrarily complex dialog systems. To personalize dialog outputs, we provide a template language which can be used to generate expressions based on previous user inputs and/or external data tables. To control dialog flow, we allow the storage of user inputs in variables, the creation of hidden variables, e.g. for loop counters, and blocks to split the dialog flow based on conditional logic.

Transparency Finally, we design DIAGRAPH to be transparent for all user groups, making it easy to understand the dialog structure and debug unintended behaviors. To this end, we provide a debugging view (figure 1) which can be opened in parallel to the dialog graph. Here, users can test out the different branches of their dialog as well as verify that the content of variables is correct at every turn. In this way, students can gain a deeper understanding of how the dialog system processes and researchers/subject-area experts can ensure that their dialog systems provide the correct outputs to end-users.

4 DIAGRAPH Software

DIAGRAPH is an open-source graphical software for designing dialog systems. The software consists of a web frontend and a python backend built on the ADVISER (Li et al., 2020) dialog system toolkit. DIAGRAPH enables users to design dialog systems by representing each turn as a node in a graph of the dialog flow. For each node, the dialog designer can define the text which the dialog system will give to the end-user and where applicable, the possible end-user responses. Nodes can then be connected to form complex dialog flows. Each node or answer can only be connected to a single follow-up node, but a single follow-up node may be reached by multiple previous nodes. Additionally, cycles (loops) can be created by connecting a node to a node earlier in the graph, allowing for more complex dialog logic.

4.1 Nodes

The dialogs created with DIAGRAPH are built using five types of nodes (see Fig. 2), which can define even complex system behavior, such as storing variables, accessing data tables, and performing logical operations.

User Response Nodes are the fundamental building blocks of the dialog graph and allow branching dialog flow. These nodes provide a dialog system utterance and a finite set of possible end-user answer prototypes. During runtime, a node of this type expects end-user input, which will then be matched against the list of its answer prototypes. The prototype most similar to the end-user's input will then be selected as the user intent, and the dialog will progress to the node connected to that answer. In case the intent recognition fails for some end-user inputs, designers may update a user response node to include answer synonyms that connect to the same follow-up node as the original answer prototype.

Information Nodes give information to the end-user without asking for input, acting as linear dialog flow. They are useful for presenting information, such as hints, to end-users when a decision from the end-user is not necessary. In this way, they can be used to split up long system answers into shorter, easier to read chunks to avoid overwhelming the end-user. Since Information Nodes

do not require end-user input, they can be directly connected to a single follow-up node.

Variable Nodes allow asking for user input and storing it in variables. They are similar to User Response Nodes in that they allow a dialog designer to define a system utterance and that they expect a response from the end-user. However, in contrast to User Response Nodes, the dialog designer does not define a set of prototypical end-user responses, but rather the general type of expected answer and the name of the variable which will store the value. Supported types include number, text, and Boolean. The user response is then stored inside this variable and can be used either to fill a template (see 4.1.1) or as part of more complex logical control. Variable nodes, like Information Nodes, can only be connected to a single follow-up node, but the values stored within the variable can be accessed at any point later in the dialog.

Variable Update Nodes are a way for dialog designers to either update the value of existing variables or create hidden variables which can be used to control the dialog flow, e.g., as loop counters. They do not provide output to the end-user.

Logic Nodes are purely used for dialog flow control, allowing to branch to follow-up nodes based on the values of variables, e.g. a number exceeding a certain threshold. This node does not provide any output to the end-user. Given a variable, Logic Nodes allow dialog designers to define a series of logical conditions based on the value of a variable. Each of the conditions can then be connected to a different follow-up node, personalizing the dialog based on previous end-user input. For more complex logic, Logic Nodes can also be connected to each other to define branches that require more than one condition or more than one variable.

4.1.1 Node Editor and Template System

In order to allow dialog designers to change the dialog system output, we incorporated the TinyMCE editor[5]. This allows dialog designers to visually format dialog system text, and include tables, links, or images. Additionally, the interface of TinyMCE is similar to that of common word processing software, reducing the barrier of such formatting.

We also provide a minimal template syntax (see Fig. 3) which can be used within a node to personalize system output based on previous input from an

[5]https://www.tiny.cloud

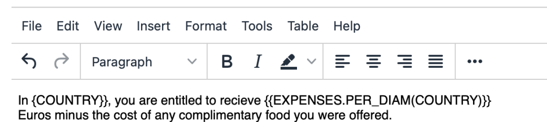

Figure 3: An example of the node editor, using the template syntax, which can be used to personalize output.

end-user and/or values from external tables which can be re-uploaded as information changes. Using the template syntax, for example, could allow dialog designers to create a single node, linked to an uploaded table which returns the per diam for a user based on the length of stay and the country of travel they have given. This can greatly simplify dialog graphs, both in terms of reducing the total number of nodes and answers needed (a single node instead of one for each country and duration) as well as in the ease of updating the graph as policies change (uploading a new table instead of editing nodes). The template syntax allows for the following operations, which can combined together to create arbitrarily complex templates: 1) referencing the value of a variable, 2) performing mathematical operations, and 3) referencing a value from an uploaded table.

4.2 Navigation Features

As dialog graphs can become quite large for complicated domains, one important aspect of our tool is helping users to maintain an overview of the whole graph as well as to find individual nodes.

Mini-Map To keep track of how the section of graph they are working on at the moment fits into the bigger picture of the entire dialog, we provide a mini-map of the whole graph in the bottom right corner of the editor, highlighting the portion currently visible.

Search We provide a search panel with fuzzy matching to help find and update specific nodes in the graph. The user can click on a result to jump to the corresponding node, which will be highlighted and placed at the center of their screen. Additionally, as weblinks may change or need updating with more frequency than other types of information, we provide a similar panel where all weblinks are listed alphabetically.

Tags and Filtering Additionally, dialog designers can create tags and add them to any node in the dialog graph. Each tag will be assigned a color text and displayed as text at the bottom of the node

Name	# of Nodes	Sharable Link to Chat	Actions
Tutorial	57	Link	Edit Graph Rename Delete Log
CTS	1	Link	Edit Graph Rename Delete Log

Create a new graph Feedback

Figure 4: Graph management dashboard. Users can create/delete, edit, and share their dialog systems.

(see figure 2). The color coding helps to visually tell what category or categories a node belongs to which can help with grouping. Users can filter which tags are visible in the graph at a time, allowing them to hide graph sections not relevant to what they are currently working on.

4.3 Debugging

To test a dialog graph before releasing it, we provide a parallel debug window next to the dialog graph editor with an interactive instance of the corresponding dialog system. Dialog designers can then directly try out different inputs and verify whether the dialog flow works as intended. To simplify understanding of the dialog flow, the editor window will pan to and focus on the node currently displayed in the chat window. Any changes to the dialog flow in the editor will be immediately available in the chat, starting from the next dialog turn. Additionally, dialog designers can use the debug panel to view the values of all variables active in the dialog and ensure their correctness at every turn.

4.4 Managing Graphs

Dialog designers can manage their graphs from a central page 4. Here they can choose create new dialog systems or edit/delete existing ones. Additionally, they can download user interaction log data or get a link to their finished dialog system. This link, which can be distributed or embedded, points to a non-editable chat window (figure 5) where end-users can interact with the current version of the dialog system.

5 Implementation Details

5.1 Dialog System

Once the dialog graph has been defined, it can instantly be used as a fully functioning dialog system. The dialog system communication backend is based on the ADVISER toolkit (Li et al., 2020), which defines an abstract service class from which dialog modules can inherit. All modules which inherit from this service class can communicate with each other using a publish-subscribe framework. In

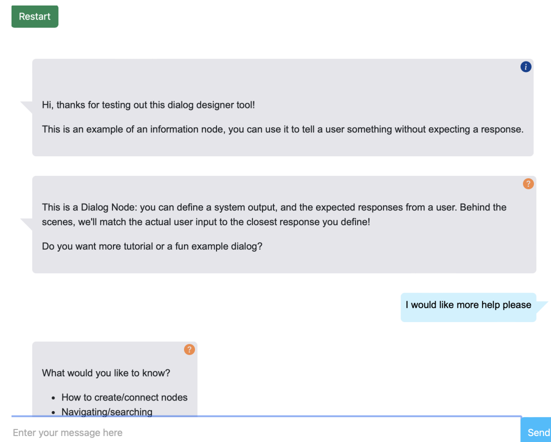

Figure 5: Embeddable/shareable chat window.

comparison to the communication protocol implemented in ADVISER, the backend of DIAGRAPH has been modified by attaching a user id to each message sent. In this way, the dialog systems created with DIAGRAPH can support an arbitrary number of end-users concurrently. The full system consists of two new modules: a policy and a natural language understanding unit (NLU). The policy navigates users through a dialog graph, choosing a next step based on the NLU output, which maps user input to one of the pre-defined answer candidates. As the dialog nodes themselves define the system output, there is no need for a natural language generation unit.

Currently, DIAGRAPH was designed for creating text-based dialogs. However, it would also be possible to create a spoken dialog system, e.g. by incorporating ADVISER's text-to-speech and automatic speech recognition modules, which can communicate with the other DIAGRAPH services. This functionality is not included in the default DIAGRAPH distribution.

5.1.1 Natural Language Understanding

The natural langauge understanding unit (NLU) is based on a state-of-the-art, multilingual similarity model (Reimers and Gurevych, 2019) for User Response Nodes and regular expressions for Variable Nodes. As User Response Nodes have a fixed set of prototypical answers and the goal is to match the user input to one of these answers, a large language model performs well. For variable nodes, however, the input space must be restricted according to the variable's type: e.g., a boolean variable should not be allowed to assume values other than *true* or *false*. Therefore, we use regular expressions to guarantee

that variable values conform to their specified type.

5.1.2 Dialog Policy

The dialog system's behaviour, the policy, is primarily defined by the dialog designer based on how they construct their dialog graph. DIAGRAPH is by default configured to use a rules-based policy that will traverse the provided graph turn-wise, beginning from the start and taking the following actions depending on the current node type:

- User Response Nodes: The policy will output the system utterance, wait for user input, and then consider which prototypical answer best matches the user input(as determined by the NLU module). Finally, it will proceed to the node connected to that answer.
- Information Node: The policy will output the system utterance and transition to the connected node.
- Variable Node: The system will output the system utterance, wait for user input, store it in the associated variable, and then traverse to the connected node.
- Variable Update Node: The system will not output any text, but update the associated variable according to the specified rule. Then, it will move on to the connected node.
- Logic Node: The system will not output any text, but evaluate each logical condition based on the value of the associated variable. The system will then proceed to the node connected to the matched condition.

As an alternative to the handcrafted policy, the graphs generated from DIAGRAPH could be exported and directly used to train a reinforcement learning (RL) policy, as proposed in our earlier work (Väth et al., 2023). In comparison to the handcrafted policy included in the standard distribution of DIAGRAPH, the RL policy adapts to the amount of information in a given user query to either navigate the end-user through the dialog tree node by node or skip extraneous nodes once the user intent can be inferred.

5.2 Editor

The DIAGRAPH frontend is implemented in React, using the React Flow [6] library to help smoothly render nodes and edges as they are moved around the screen. As graphs can be quite large, we focused on efficiency of rendering for the frontend, trying to keep both the amount of memory needed to run the editor and the amount of external libraries to a minimum. In this way, the editor can seamlessly support graphs with hundreds of nodes, in terms of creating or updating nodes, as well as in terms of fluid navigation. The dialog nodes created in the frontend are automatically stored in the backend database, which is updated every time information about the node (position, text, connections, associated answers, etc.) is changed. Keeping the frontend store and the backend database synced ensures that the handcrafted policy is always up to date with the state of the graph displayed in the editor. For handling the database connections, user authentication, (re)starting the dialog system, and serving a compiled version of the front end, we use the python toolkit Django [7].

6 Evaluation

To evaluate the usefulness of our software, we tested its usability in three different scenarios 1) to create a real-world dialog system in a complex domain, 2) to rapidly prototype dialog systems, 3) to teach students about programming and dialog systems.

6.1 Complex Real-World Domain

As a first test, we worked with three subject-area experts from a university travel reimbursement department, to create a dialog system to help employees navigate travel planning and reimbursement. None of the experts had previous experience with chatbots, but hoped to offload common questions to the chatbot in order to have more time for complex cases and processing reimbursements.

Given the complexity of the domain and the importance of providing legally correct information, it served as a good test of DIAGRAPH's full functionality. To create the dialog system, the subject-area experts first generated a set of frequently asked questions and then worked with us to sort them into categories and dialog sequences. Once they had a clear picture of how they wanted to group information, they were provided with an interactive tutorial and user manual for the system. After a collaborative phase to implement the first version of the dialog graph, the experts were left alone to expand and update it, resulting in a final version with

[6] https://reactflow.dev

[7] https://www.djangoproject.com

194 nodes and a maximum depth of 32. The dialog system defined by this graph was then released for university employees as an additional option for answering travel related questions. During the initial test phase, approximately 2000 dialogs were conducted by university employees, each lasting roughly 5.9 turns

At the end of the collaboration, the experts were asked provide feedback about the experience via the System Usability Scale (Brooke, 1996), a ten item Likert scale for measuring user interfaces. They were also asked to give free-form feedback about their positive and negative impressions. DIAGRAPH received largely positive feedback with an average SUS score of 82 (highest possible 100; average 69 (Bangor et al., 2009)). Experts appreciated its user friendliness, how well dialogs could be specified, and the freedom for creative design the tool promoted.

This use-case highlights that the dialog designer can provide the level of control needed for highly complex and sensitive domains and be deployed in real-world scenarios.

6.2 Rapid Dialog System Design

In a second test, we investigated DIAGRAPH as a tool for rapidly creating dialog systems. We collected 19 participants and asked them to take 15-30 minutes to create and test a new dialog system of their own design. Participants were provided with a tutorial in the form of an interactive dialog graph and an example dialog of a digital ice cream seller. In contrast to the previous scenario, participants were not provided any in person instruction. Despite the lack of additional instruction and short interaction time, all participants were able to successfully develop a variety of dialog systems.

After interacting with DIAGRAPH, participants were also asked to fill out the 10 item SUS questionnaire and provide free-form feedback on things they liked or disliked about the tool. When evaluating the survey results, DIAGRAPH was given an average SUS score of 75 (out of a possible 100) indicating above average usability. This was also reflected in the comments, where the software was described as “fun!" and “intuitive to use", although the tutorial dialog was generally seen as too long.

As results from the SUS can be considered to generalize when tested with at least 12 participants(Brooke, 2013), and because all users were able to create dialog systems in such a short time, our results confirm that DIAGRAPH provides an intuitive interface which allows for rapid prototyping of dialog systems.

6.3 Educational Tool

Finally, we held a workshop on dialog systems with a group of six high school aged students to explore how DIAGRAPH could be used in an educational setting. Students were given a 45 minute long introduction to dialog systems and programming concepts. Following the theoretical introduction, they were given a 30 minute interactive tutorial – on how to create a chatbot for selling icecream with the DIAGRAPH tool – and then allowed to create and test their own dialog systems. The experience was rated fun and engaging by all participants (1.5 on a six-point Likert scale from very engaging to not at all engaging). All participants who left free-form feedback further indicated that they enjoyed the experience and/or felt that they learned a lot from it. Although not all participants had previous coding experience, all students were able to successfully create their own dialog graphs by the end of the half hour, each of which involved some type of logical operation or loop condition.

This experiment suggests that in addition to being easy to use, DIAGRAPH has potential as a teaching tool for dialog systems and for programming concepts.

7 Conclusion and Future Work

In this paper we have presented DIAGRAPH: an open-source graphic interface for designing dialog systems supporting either rules-based or RL-based dialog graph navigation. DIAGRAPH provides an intuitive way for subject-area experts to create complex dialog systems, users to rapidly prototype dialog interactions, and students to learn about dialog systems – regardless of the user’s level of technical background. Our user evaluation shows that DIAGRAPH was considered easy to use for all three use cases and users generally considered working with the tool an intuitive and fun experience.

In the future, we hope to extend our tool with the ability to query web APIs and to allow dialog designers to define expected inputs for variable nodes using custom regular expressions in order to increase flexibility even further. By releasing this software as open-source, we hope to make dialog design more accessible and to spark more research in controllable dialog policies.

References

Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123.

Dan Bohus and Alexander I. Rudnicky. 2009. The ravenclaw dialog management framework: Architecture and systems. *Computer Speech & Language*, 23(3):332–361.

John Brooke. 1996. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7.

John Brooke. 2013. Sus: a retrospective. *Journal of usability studies*, 8(2):29–40.

Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. 2017. A survey on dialogue systems: Recent advances and new frontiers. *SIGKDD Explor. Newsl.*, 19(2):25–35.

Alexander Koller, Timo Baumann, and Arne Köhn. 2018. DialogOS: Simple and Extensible Dialogue Modeling. In *Proc. Interspeech 2018*, pages 167–168.

Sungjin Lee, Qi Zhu, Ryuichi Takanobu, Zheng Zhang, Yaoqin Zhang, Xiang Li, Jinchao Li, Baolin Peng, Xiujun Li, Minlie Huang, and Jianfeng Gao. 2019. ConvLab: Multi-domain end-to-end dialog system platform. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 64–69, Florence, Italy. Association for Computational Linguistics.

Chia-Yu Li, Daniel Ortega, Dirk Väth, Florian Lux, Lindsey Vanderlyn, Maximilian Schmidt, Michael Neumann, Moritz Völkel, Pavel Denisov, Sabrina Jenne, Zorica Kacarevic, and Ngoc Thang Vu. 2020. ADVISER: A toolkit for developing multi-modal, multi-domain and socially-engaged conversational agents. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 279–286, Online. Association for Computational Linguistics.

Pierre Lison and Casey Kennington. 2016. OpenDial: A toolkit for developing spoken dialogue systems with probabilistic rules. In *Proceedings of ACL-2016 System Demonstrations*, pages 67–72, Berlin, Germany. Association for Computational Linguistics.

Daniel Ortega, Dirk Väth, Gianna Weber, Lindsey Vanderlyn, Maximilian Schmidt, Moritz Völkel, Zorica Karacevic, and Ngoc Thang Vu. 2019. ADVISER: A dialog system framework for education & research. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 93–98, Florence, Italy. Association for Computational Linguistics.

Dinesh Raghu, Shantanu Agarwal, Sachindra Joshi, and Mausam. 2021. End-to-end learning of flowchart grounded task-oriented dialogs. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4348–4366, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics.

Swadheen Shukla, Lars Liden, Shahin Shayandeh, Eslam Kamal, Jinchao Li, Matt Mazzola, Thomas Park, Baolin Peng, and Jianfeng Gao. 2020. Conversation Learner - a machine teaching tool for building dialog managers for task-oriented dialog systems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 343–349, Online. Association for Computational Linguistics.

Stefan Ultes, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gašić, and Steve Young. 2017. PyDial: A multi-domain statistical dialogue system toolkit. In *Proceedings of ACL 2017, System Demonstrations*, pages 73–78, Vancouver, Canada. Association for Computational Linguistics.

Dirk Väth, Lindsey Vanderlyn, and Thang Vu Ngoc. 2023. Conversational tree search: A new hybrid dialog task. In *[forthcoming]Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. ACL Anthology.

Qi Zhu, Zheng Zhang, Yan Fang, Xiang Li, Ryuichi Takanobu, Jinchao Li, Baolin Peng, Jianfeng Gao, Xiaoyan Zhu, and Minlie Huang. 2020. ConvLab-2: An open-source toolkit for building, evaluating, and diagnosing dialogue systems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 142–149, Online. Association for Computational Linguistics.