Proceedings of the 13th Conference on Language Resources and Evaluation (LREC 2022), pages 4069–4078
Marseille, 20-25 June 2022

The *slurk* Interaction Server Framework: Better Data for Better Dialog Models

Jana Götze, Maike Paetzel-Prüsmann, Wencke Liermann, Tim Diekmann, David Schlangen
Foundations of Computational Linguistics, Department Linguistics
University of Potsdam, Germany
{firstname.lastname}@uni-potsdam.de, diekmann.tim@gmail.com

Abstract

This paper presents the *slurk* software, a lightweight interaction server for setting up dialog data collections and running experiments. *slurk* enables a multitude of settings including text-based, speech and video interaction between two or more humans or humans and bots, and a multimodal display area for presenting shared or private interactive context. The software is implemented in Python with an HTML and JAVASCRIPT frontend that can easily be adapted to individual needs. It also provides a setup for pairing participants on common crowdworking platforms such as Amazon Mechanical Turk and some example bot scripts for common interaction scenarios.

Keywords: dialog, data collection tool, interaction, chat, spoken dialog, multimodal dialog, crowdsourcing

1. Introduction

Much of NLP's breakthroughs in recent years is based on data-driven learning methods. Data-hungry machine learning algorithms to model Visual Question Answering (Das et al., 2017) or Vision and Language Navigation Tasks (Krantz et al., 2020) are fed with crowdsourced data, which is fast and affordable to obtain, as crowdworkers do not have to be brought to the lab for common tasks like labeling, captioning images or producing navigation instructions. Data for the subfield of dialog modelling requires at least two crowdworkers to be involved, something that is a non-standard use case for the most popular crowdsourcing platforms as it requires coordinating two or more workers to find a common timeslot to work on a task.

The *slurk* tool adds to a body of frameworks and tools (Healey et al., 2003; Manuvinakurike and DeVault, 2015; Miller et al., 2017; Schlangen et al., 2018) that facilitate data collection for training and testing dialog models where it is often necessary for researchers to set up their own data collection that satisfies their specific needs, e.g., to cover some specific domain or dialog phenomenon. *slurk* allows to create experiments for human-human or human-machine interaction, with no limitation on the number of participants for a given setup. Possible interaction channels include text as well as audio and video. Dialog games – consisting of an interaction setting such as a text or audio channel, a certain context to refer to, and a task to solve – can be created to include multimodal context such as images or interactive tools in which participants can manipulate the context together.

In this paper, we describe the *slurk* software, a modular tool for collecting multimodal dialog data that is integratable with crowdworking platforms to pair up participants on demand. We explain how *slurk* can be set up to constrain the interaction channel in a number of ways as well as to manipulate the visual context for each participant. Section 2 details the purpose of the software, Section 3 describes related frameworks, and Section 4 introduces the *slurk* architecture and system features in detail. In Section 5 we demonstrate how these features can be used to set up data collections and experiments, in the lab or via crowdsourcing.

2. Goals

The main purpose of *slurk* is to provide a framework that is flexible and modular to set up a variety of dialog tasks in order to both collect data from human conversations as well as test existing dialog models with human evaluators. Dialog context, such as images or interactive buttons, as well as the interaction channel can be manipulated in a number of ways as we outline in this paper. Figure 1 shows an example interface for a dialog task, with the chat area on the left, showing the dialog history, and what we call the display area on the right, providing the visual context.

Developing new dialog models for different settings often requires researchers to first collect data of humans performing both sides of the task in order to fully understand the parameters of what they are modelling, e.g., what does the conversation look like when participants have different roles, when they do not share the visual context or when the task formulation changes slightly? Then, once a model has been developed, it is imperative to test it with human users. While many metrics exist that evaluate dialog along a number of dimensions, interactions with and judgments from humans are vital to understand a model's scope and detect its limitations.

slurk aims to be useful for both understanding human-human conversations as well as evaluating dialog models and is designed to be extended for new settings. For example, we can use the display area to

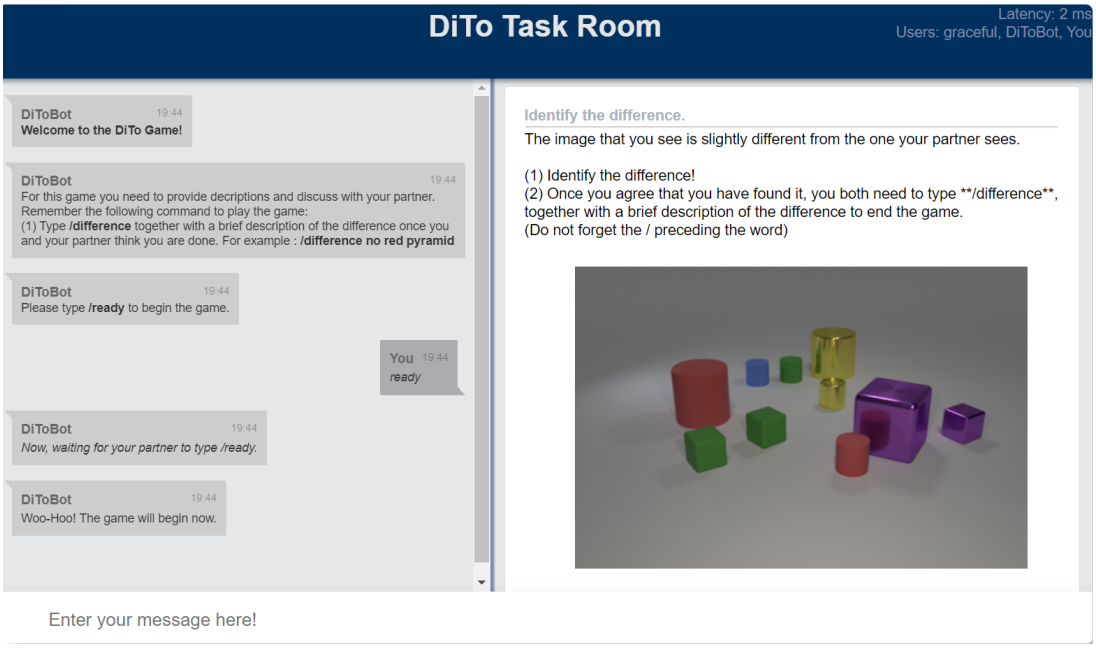


Figure 1: One participant's browser view in the DiTo task. The browser view is separated into the *chat area* on the left and the *display area* on the right. The other participant in this task sees a different image. The top right corner shows who is present in the room. Images are taken from the CLEVR dataset (Johnson et al., 2017).

connect users to external tools and send information via the `slurk` server, thus synchronizing all information and logs in one place.

3. Related work

The three platforms that are most similar to `slurk` are the parlAI tool (Miller et al., 2017), the DiET toolkit (Healey et al., 2003) and the Pair Me Up Web Framework (Manuvinakurike and DeVault, 2015).

The parlAI platform (Miller et al., 2017) was built with the goal of supporting the creation of general open-domain chatbots in mind. It is a text-based dialog platform that lets developers of language models test their trained models on many different tasks to which the platform provides seamless access. The framework can also be used for training models and sharing datasets. parlAI has been used mainly for open-ended dialog, e.g., to study what factors make for a good conversation (See et al., 2019).

The Dialogue Experimentation Toolkit (DiET) (Healey et al., 2003) focusses on studying human-human text-based dialog. The toolkit aims at studying manipulations to the interaction settings, e.g., by deleting, changing, or adding turns. It also provides a GUI interface for defining such interventions. Some DiET features can be run via the open-source chat messenger Telegram, making it easily accessible for non-lab participants. The toolkit has been used for example to study the role of laughter in chats by inserting artificial laughter tokens into turns (Maraev et al., 2020).

The Pair Me Up Web Framework (PMU) (Manuvinakurike and DeVault, 2015) has originally been developed for collecting human-human spoken conversations over the web and was later extended to allow for autonomous bots to be paired with human conversation partners as well. Although it has so far been used only in one particular interactive setting (RDG-Image (Paetzel et al., 2014)) and is not actively maintained anymore, the general technology developed for pairing participants as well as recording their audio and synchronizing it with events in the game interface could be applied to other domains.

While the first two platforms focus on two very different goals – building an open-ended chatbot vs. studying human-human interactions – they share some features. parlAI and DieT are text-based and comprise a display area to present visual context to the human dialog partners in the form of images. Image displays are configurable in both, but in contrast to `slurk`, no interactive elements such as buttons can be part of the interaction context. The third platform, PMU, like `slurk`, allows for an interactive visual context, the pairing of both human and artificial conversation partners as well as the potential for spoken conversations; it has however not been developed into a general purpose tool (and is not in active development), and lacks the flexibility that `slurk` offers.

parlAI dialogs follow a strict turn-taking regime in which participant and chatbot turns strictly alternate. Turns are displayed in full as the focus is on the generated language. DiET is more flexible with respect to turn-taking. It allows free turn-taking between participants and implements, in addition to sending messages turn by turn, a WYSIWYG mode that simulates

	parlAI	DiET	PMU	*slurk*
Interaction				
via text	✓	✓		✓
via audio			✓	✓
via video				✓
flex. turn-taking		✓	✓	✓
human-human		✓	✓	✓
Dialog context				
images	✓	✓	✓	✓
config. layout	✓		(✓)	✓
interact. elements			✓	✓
Data collection				
AMT integration	✓		✓	✓
Telegram integr.		✓		
Models included	✓			

Table 1: Features of the dialog platforms parlAI (Miller et al., 2017), DiET (Healey et al., 2003), Pair Me Up (Manuvinakurike and DeVault, 2015) and *slurk*.

incrementality by sending typed characters to the other participant immediately, and lets them fade after some time. *slurk* implements a similar mode called *live-typing* that can send messages character by character but does not let them fade from the display area. PMU allows for communication via spoken language only and does not enable experimental restrictions on turn-taking. *slurk* also implements a number of additional options to control the interaction setting, such as incorporating mouse clicks or annotation elements and an audio and video channel that can be restricted, e.g., by enforcing a push-to-talk turn-taking setting.

For setting up data collections or experiments, parlAI provides an AMT interface, and DiET an interface to the Telegram chat messenger. *slurk* and PMU provide a pairing setting that can be used with either a crowdsourcing platform like AMT or via a desktop browser directly. Table 1 summarizes some of the four tools' features, providing an overview of the main differences between them.

Other tools exist that focus on manipulating only a specific part of the interaction channel, e.g., changing the speech signal to adjust for emotional cues (Rachman et al., 2018) or changing gestures recorded via virtual reality tools to produce fake gestures (Gurion et al., 2018). Furthermore, specific environments have been created to present rich interaction contexts to participants while keeping the interaction channel fixed, usually in a turn-by-turn setting. An example is the Minecraft virtual environment in which communication happens between a player and a bot to build objects together (Gray et al., 2019).

A previous version of the *slurk* software (Schlangen et al., 2018) has already been used for research, recruiting participants via AMT. For example, Ilinykh et al. (2019) set up a task in which participants use commands to navigate a network of rooms, represented by changing images; Attari et al. (2019) presented images for participants to discuss; Galetzka et al. (2020) manipulated the status of messages, controlling whether information is shared or private; and Chiyah Garcia et al. (2020) used a dialog task in which the window layout differs depending on the participant's role in the interaction, including buttons for certain actions. Haber and Poesio (2020) described their setup for presenting participants with static images without recruitment via a crowdsourcing platform. In this paper, we describe the existing and new features in more detail. In the new software version, we have improved the API and extended it to handle audio and video data and implemented a number of new plug-ins and example bots that exemplify the described behaviors.

4. Architecture and features

slurk runs as a server that provides the interface to communicate with clients, which can be either human dialog participants or software "bots", cf. Figure 2. Interactions happen in "rooms" to which the clients log on. A permission system controls what a client can see and do. Room layouts are divided into a *display* area that can be used for providing task-based content such as images, and a *chat* and *input* area that shows the chat history, input field, and the video if desired, see Figure 1.

4.1. Core *slurk* concepts

The core concepts in *slurk* are the following:

- **Room**: A room is the space in which users interact with each other, with a bot, or with material presented to them, e.g., an image.
- **User**: A user is a participant in the interaction who has certain permissions that may restrict her access. Permissions may also change during an interaction, e.g., at the end of a task we may want to prevent users from making further contributions. Both human participants and bots are users. A human user can only be in one room at a given time. Bots can participate in several rooms at once.
- **Task**: A task defines what a room looks and behaves like. We can define the number of users to be assigned to a task. Rooms can then be opened based on task information, so that several rooms can have interactions about the same task in parallel.
- **Token**: User permissions are encoded via tokens. Tokens also carry information about which task a user is assigned to.
- **Layout**: The layout defines what the users can see, e.g., what type of chat history, and what the context looks like, e.g., whether an image or buttons are visible. The visible context can change during an interaction.

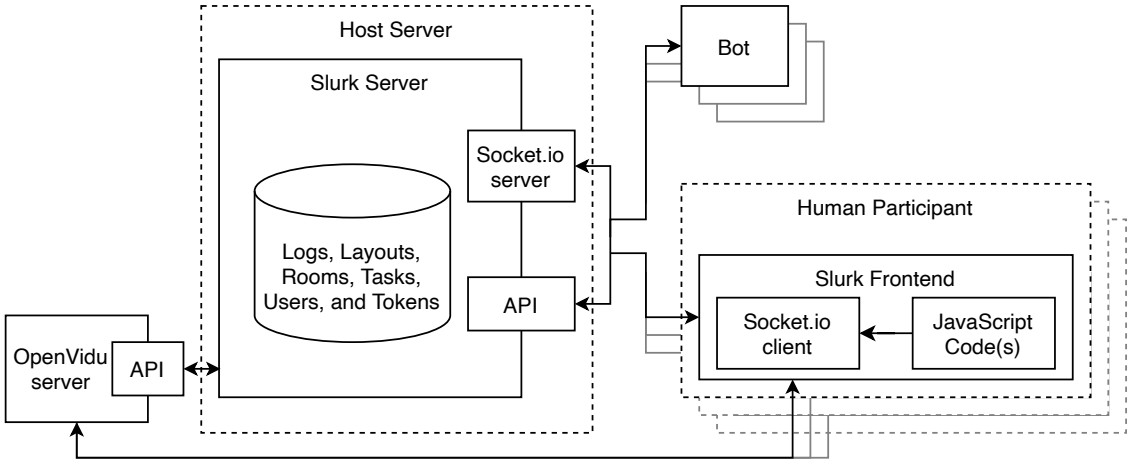


Figure 2: The `slurk` main architecture. The `slurk` server is deployed on a host machine. Clients – bots or participant frontends – connect via the `slurk` API and socket.io. All communication between clients happens via the server. Participants log into rooms using a generated token to see the frontend of their assigned room.

- **Events**: Both server and clients emit events that bots may react to for defining the logic of an interaction. For example, bots may react to a user entering a room or on a text message that was sent in a certain room.

A typical data collection setup involves defining a layout and task and using a script that creates rooms on demand (the Concierge Bot, cf. Section 5.2.1), whenever the necessary number of participants has entered a waiting room. Other bots can then join this room. Human users log in to `slurk` using a url link that encodes their token and with it the permissions they have for carrying out actions like sending messages or images. We describe some example bots and settings in Section 5.

4.2. Technology

Figure 2 shows the overall server-client architecture of `slurk`. The system is built in Python, on top of flask[1] and flask-socketio.[2] The `slurk` server communicates with a separate video server via https for audio and video communication (cf. Section 4.3). The video server can be deployed on a different machine and is configured via the `slurk` REST API, thus making everything configurable in one place.

Bots use the `slurk` API via socket.io[3] to act in a particular setup: They can create rooms on the fly, send users into a room or disconnect them, or they can associate a video session to a room. Bots use the API also to act as overt or hidden dialog participants. They can react on server events, e.g., when human users send text messages, commands or images, or when they click on an element in the display area. Bots emit events themselves when they message a user or when they modify the display area of one or more users.

Human users are served an HTML-page that connects to the `slurk` server via flask-socketio. A token specifies the permissions they have, e.g., to send private messages or commands or to participate in a video session. Permissions are specified by the experimenter depending on the task. The frontend that users see is written in JAVASCRIPT and configures some functionality via plug-ins, e.g., the command syntax and the syntax for private messaging.

4.3. Audio & video interaction

Adding video to remote dialog allows us to bridge the gap between a face-to-face setting and a pure audio setting with shared visual context. Video analysis of facial gestures in face-to-face conversation has for some time played a role in studying dialog (van Son et al., 2008; Oertel et al., 2013). Adding the video modality to remote conversation can give us new insights into the role of the facial gesture modality. Note that it is also possible to use `slurk` in settings where there is only one user to record, e.g., to collect acted gesture performances (video) or spoken descriptions of images or other stimuli (audio only).

For allowing users to interact via audio and video, we have added the possibility to connect to an *OpenVidu*[4] server session directly via `slurk`. *OpenVidu* is an open-source software for streaming live video and audio based on the highly compatible WebRTC framework for streaming multimedia data. It is licensed under Apache License v2 and the free version of the software includes streaming and recording videos, as well as self-hosting, making it possible to keep full control of the data.

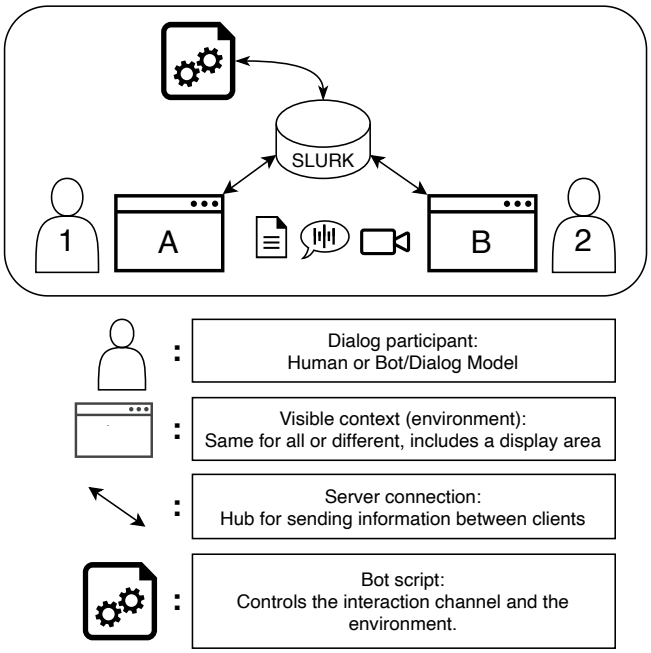
Connections to *OpenVidu* sessions can be set up directly via the `slurk` API. For a room to include a video or audio session, it has to be associated with the respective *OpenVidu* session. Users in the room must

[1] `http://flask.pocoo.org`

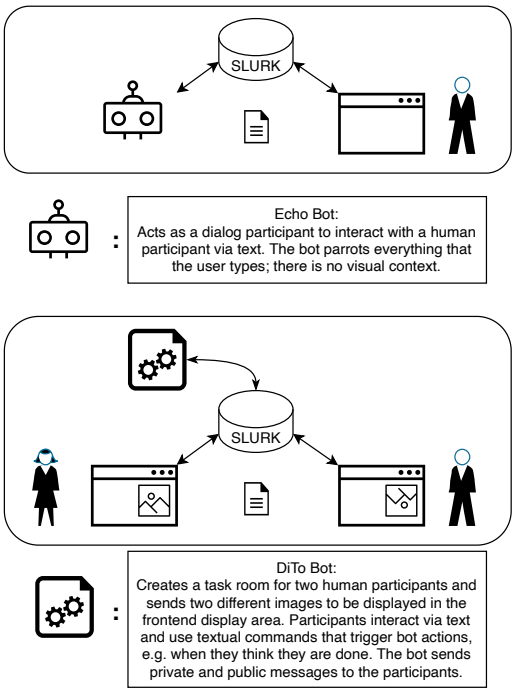
[2] `https://flask-socketio.readthedocs.io`

[3] For example using Python or any other socket.io client.

[4] `https://openvidu.io`



(a) Abstract depiction of possible information flow: Human participants access the server via their browser clients, bot scripts can access the server via the API. In practice, bot scripts and bot participants are implemented as one object.



(b) Setups for the Echo Bot with one participant (top) and the DiTo Bot with two participants (bottom).

Figure 3: Schema and two instantiations of interaction setups.

be granted permissions (via their tokens) to publish or subscribe to the *OpenVidu* session. In the future, we plan to further explore the quality of this data that we can obtain using crowdsourcing and the privacy implications of this type of data collection.

4.4. Bots

Bots are client programs that can combine different roles in order to control the interaction. Bots can take the role of a full dialog agent, i.e., we can use them to test any dialog model by letting human users interact with the model. Bots can also control the interaction as hidden agents, e.g., they can interpret commands, modify contributions, or suppress contributions. This gives us the possibility to control an ongoing interaction. For eyample, a bot might react to certain words and send a private message to a user in response, or impersonate another user by modifying their turn.

Bots may also implement background logic, i.e., they interpret button presses and other events happening in the display area, react on incoming users, or track time since the last contribution. For example, in the MeetUp! corpus (Ilinykh et al., 2019), users navigate a room network by typing commands such as `/n` for "go north". The bot interprets the command and changes the image in the display area accordingly. Bots can also administrate the main settings, e.g., they can move users between rooms or change their permissions. Bots can be triggered to end an interaction, e.g. via a timeout or a user command, making it possible to define custom logic for data collections.

4.5. Configuring the visual context

A growing body of research today is concerned with building models that can integrate language and visual information, for example to reason about spatial relations of objects in an image (Bisk et al., 2016). *slurk*'s display area is configurable to show dialog participants custom images or other visual material. The configuration of various visual aspects is done via a JSON file from which HTML is built. The bot script can then determine (and throughout the interaction change) what each user can see. For example, in the DiTo Bot setting, two participants are trying to find the difference between the images that they see (cf. Figure 3b). A screenshot of what one participant's browser window looks like can be seen in Figure 1.

4.6. Logging

All parts of an interaction are logged on the *slurk* server in a JSON format. Figure 4 shows example log entries for common events. JSON data[5] is human-readable and can easily be parsed automatically and converted to other data formats for annotation or analysis. Every log entry contains a timestamp, allowing synchronization with all data sources, including the video and audio recordings. Logging happens continuously, so that logs retrieved at a certain time reflect the interaction up to that point.

[5] https://www.json.org

```
{
  "data": {
    "broadcast": false,
    "html": false,
    "message": "hello!"
  },
  "date_created": "2021-11-22T16:42:30",
  "date_modified": null,
  "event": "text_message",
  "id": 5,
  "receiver_id": null,
  "room_id": 1,
  "user_id": 2
}
```


A


```
{
  "data": {
    "coordinates": {
      "bottom": 0.6728571428571428,
      "left": 0.6385,
      "right": 0.9605,
      "top": 0.45571428571428574
    },
    "type": "add"
  },
  "date_created": "2022-01-04T17:02:23",
  "date_modified": null,
  "event": "bounding_box",
  "id": 26,
  "receiver_id": null,
  "room_id": 2,
  "user_id": 3
}
```


B

Figure 4: Example log entries for different events: (A) A user has sent a text message. (B) A user has drawn a bounding box in the display area.

4.7. License, download and development

slurk is available at `https://github.com/clp-research/slurk` under a BSD-3-Clause License. The repository contains instructions about how to download, install and deploy the server, as well as a tutorial for setting up an example room. A second repository contains example bots at `https://github.com/clp-research/slurk-bots`. Both repositories are public for others to contribute to or file issues for support and improvement.

5. Studying dialog with *slurk*

The *slurk* tool is intended to allow fine-grained manipulation of interaction behaviors in order to study dialog phenomena such as creating common ground in shared or private settings. A specific dialog experiment will have to define the interaction channel, e.g., how many participants are interacting and whether the interaction is written or spoken, the context that is available to the participants, i.e., what they can see and how they can interact with entities in the context, and the rules of the interaction, e.g., when the interaction is successful. We describe in the following how the settings for these building blocks can be manipulated in *slurk*.

5.1. Controlling interaction settings

Figures 3a and 5 schematically show how the settings of the chat and display area can be adjusted. The default mode of interaction is text, shown in the interaction area in the left part of the window. The chat area contains the dialog history as is common for similar chat tools. By default, users can see their own and others' messages, ordered by recency. In addition, an audio and video connection can be established. Any visual dialog context is shown in the display area in the right part of the browser window (cf. Figure 1).

Incrementality: Two plug-ins are available to show to users how the dialog is evolving: The `typing-users` plug-in shows who is currently typing, the `live-typing` plug-in sends messages directly as they are being typed, without the typer having to hit the "send" button. Plug-ins are specified as part of the room layout as shown in Figure 6a.

Turn-taking: Turn-taking can be manipulated along a scale to either follow a strict turn-handover regime in which the addressee cannot talk until they are explicitly given the turn, or be fully incremental. By default, anyone can type or speak at any time. Bots can temporarily prevent users from typing by changing their permissions based on custom logic. For example, a bot can keep track of turns and enforce a round-robin format in which each participant needs to wait until it is their turn to contribute.
Similarly, in spoken interaction, we can control whether the audio channel is always open for everyone to speak or enforce stricter turn-taking by using a "push-to-talk" setting in which speakers signal when they are done speaking in order to open the channel for someone else.[6]

Intercepting the channel: Communication between the clients happens via the server and it is therefore possible to modify any messages before sending them on to their addressee (cf. Figures 2 and 5). Bots can change, insert or delete text messages and pretend to be another user. In the same way, it is possible to intercept the audio and video channels, although we are leaving such tests to future work.

Multimodal context: The display area serves as dialog context, controlled by JAVASCRIPT, and can contain arbitrary HTML elements. For example, the display area can present images, buttons, or pre-recorded audio and video elements, or embed interactive tools. The position of the live video can be freely adjusted. We also make available sample plug-ins that track mouse movement and let participants draw bounding boxes. Any element in the display area can be modified programmatically by bots during the interaction so that the context can change, e.g., images may become visible or invisible to one or more users. An example is shown in Figures 6b and 7.

[6] Note that it is possible to mix written and spoken dialog.

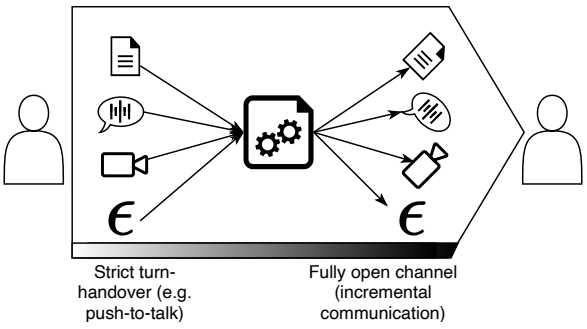


Figure 5: The interaction channel between users (humans or bots) can be intercepted and modified in a number of ways, including removing and inserting elements (ϵ). The bot can either be an overt participant or be invisible to some or all others. Human user clients communicate via the *slurk* server that emits events to the bot who can then manipulate the channel. Turn-taking regimes can set restrictions on how early a message is sent and when the addressee can answer.

Shared vs. private information: Permissions control a range of possible user behavior, including what each participant can see in the context, i.e., whether all participants see the same context or different contexts. Privately sent text is only visible to the respective addressee. In this way, bots can send administrative messages to single participants, e.g., to encourage them to contribute or remind them of the task rules.

5.2. Data collection in practice

5.2.1. Pairing participants

When collecting human-human dialog data, participants need to be recruited and paired based on the specific task. The biggest challenge in connection to collecting dialog data from crowdworkers is often the synchronous nature of the task. For most other tasks, crowdworkers perform tasks in their own time, with no dependencies on other workers. This section shows how we address this challenge.

For data collections, we set up a task bot that is responsible for the core of the dialog game users are asked to play. This bot might first present some general instructions to users, repeat the dialog task, or remind a user to engage with a particular part of the context. For example, the DiTo Bot in Figure 1 (a variant of the bot used by Attari et al. (2019)) presents the task and makes sure both users are ready to start the dialog.

When participants log in to *slurk*, they are not immediately sent to a room with their task bot, but instead they see a waiting room and another bot that is in the room with them. This bot, the *Concierge Bot*, monitors incoming users and their tasks and keeps track of the task requirements: Once enough users for a task have entered, the Concierge Bot creates a new task room and sends the users to the new room. The task bot then joins this task room as well. Any bot can also track time,

```
{
  "title": "Room",
  "scripts": {
    "incoming-text": "display-text",
    "incoming-image": "display-image",
    "submit-message": "send-message",
    "print-history": "plain-history",
    "typing-users": "typing-users"
  }
}
```

(a) A minimal room layout defining what JAVASCRIPT plugins to use for different tasks. In this example, text, images and history are displayed, as well as who is currently typing.

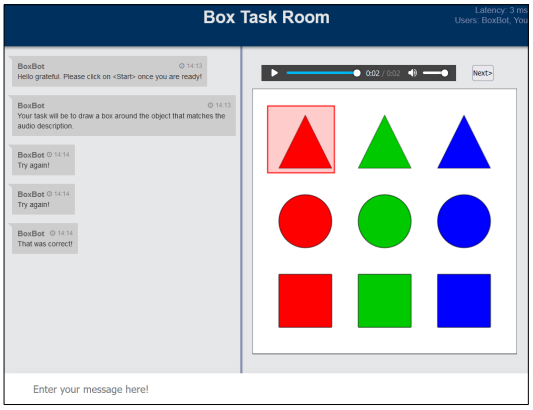
```
{
  "title": "Box Task Room",
  "html": [
  {
    "layout-type": "div",
    "style": "text-align: center;",
    "layout-content": [{
      "layout-type": "audio controls",
      "id": "audio-file",
      "src": "",
      "autoplay": "true",
      "style": "height:30px;"
    }]
  },
  {
    "layout-type": "div",
    "style": "text-align: center;",
    "layout-content": [{
      "layout-type": "image",
      "id": "drawing-area"
    }]
  }],
  "scripts": {
    "incoming-text": "markdown",
    "incoming-image": "display-image",
    "submit-message": "send-message",
    "print-history": "markdown-history",
    "plain": "bounding-boxes"
  }
}
```

(b) A room layout specifying HTML elements for an audio player and an image. The bot will later insert the respective source files. The image element is labeled with `"id":"drawing-area"` so that the bounding-box plug-in can access the element. The plug-in is specified in the `scripts` block.

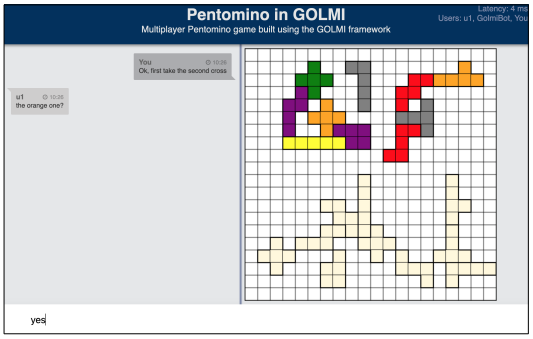
Figure 6: Example JSON room layouts.

so that participants can receive a small reimbursement even for their waiting time and are sent back to the crowdsourcing platform in case no other participants should appear within a given timeframe, e.g., five minutes (cf. Section 5.2.2).

For the DiTo task, two participants are required, so the Concierge Bot forwards incoming users to a newly cre-



(a) Display area containing an audio player, a button next to it, an image, and a bounding box that a user has drawn.



(b) Display area containing an interactive Pentomino game.

Figure 7: Example display areas.

ated task room once this number is reached. A third and fourth participant would trigger the Concierge Bot to create a second task room, so that the same task can be carried out in parallel in different rooms. Figure 1 shows the task room as seen by participants. Figure 3b schematically shows the task room setup that configures the interaction channel and display area in which users have separate visual contexts and a bot is present to administrate the interaction by tracking time and counting contributions.

For running Wizard-of-Oz data collections, either the same pairing mechanism can be used, or a bot can send participants directly into a room with a specified wizard. The display area of the `slurk` window can be configured to include any tools that the wizard might need, such as buttons for predefined speech or text, images, or task text.

5.2.2. Connecting with crowdsourcing platforms

Another requirement for integrating a `slurk` data collection on a crowdworking platform such as Amazon Mechanical Turk is that workers need to leave the platform for performing the task and after completing it need to return to it so that their profile can be associated to the correct `slurk` logs in order to evaluate and reimburse them correctly.

When directing participants from the crowdworking or experiment platform to their `slurk` room, it is possible to include all necessary information into the url they need to follow. In particular, the url contains their individual access token and we can automatically assign them a name that keeps them anonymous. This has the advantage of reducing the steps they would otherwise need to follow (choosing an access name and copy/pasting the token).

In order to connect their dialog data (present in the `slurk` server) to their crowdworking profile (present in the provider's system), a mechanism can be included in a bot that provides an individual code to each participant once they have completed their task. Participants need to enter the code on the crowdworking platform once they have returned, so that their profile can be linked with the correct dialog logs.

5.2.3. Collecting video data via crowdsourcing

Collecting dialog data remotely involves pairing people that usually do not know each other. While random pairing has been done for both text and audio, e.g., for the Switchboard corpus (Godfrey et al., 1992), the video channel adds another dimension of privacy concerns that needs to be accounted for.

Video data has been collected via AMT before, e.g., by Sigurdsson et al. (2016), but it is a different matter whether a participant shares video with researchers that can vouch for data security or whether to share live video with another participant. Special care also needs to be exacted in deploying the video server to safeguard it from unauthorized access.

The *OpenVidu* platform allows us to record video on our own servers and adhere to local data protection laws. We explicitly want to stress that when using the audio and video features, participants need to be informed that their data is recorded in these modalities and that their explicit consent is needed.

6. Conclusion

We have described `slurk`, a lightweight interaction server to experiment with dialog. The infrastructure can be used to collect dialog data or test dialog models by letting human participants interact with them. `slurk` allows a range of manipulations to the interaction channel as well as user-defined dialog context, such as images or interactive elements. Among the planned work for the future are a bot that allows collaborative manipulation of a Pentomino game board (Zarrieß et al., 2016), cf. Figure 7b; an integration for a text messenger such as Telegram; and experimenting further with the audio and video channel. We invite the community to use the tool for their purposes and are open for suggestions for further features. We hope that the tool can contribute to making it easier to collect better dialog data that in turn lead to better models.

7. Acknowledgements

This work was partially funded by DFG project 423217434 ("recolage").

8. Bibliographical References

Attari, N., Heckmann, M., and Schlangen, D. (2019). From Explainability to Explanation: Using a Dialogue Setting to Elicit Annotations with Justifications. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 331–335, Stockholm, Sweden. Association for Computational Linguistics.

Bisk, Y., Marcu, D., and Wong, W. (2016). Towards a Dataset for Human Computer Communication via Grounded Language Acquisition. In *AAAI Workshop: Symbiotic Cognitive Systems*.

Chiyah Garcia, F. J., Lopes, J., Liu, X., and Hastie, H. (2020). CRWIZ: A framework for crowdsourcing real-time wizard-of-oz dialogues. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 288–297, Marseille, France, May. European Language Resources Association.

Das, A., Kottur, S., Gupta, K., Singh, A., Yadav, D., Moura, J. M. F., Parikh, D., and Batra, D. (2017). Visual Dialog. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 326–335.

Galetzka, F., Eneh, C. U., and Schlangen, D. (2020). A Corpus of Controlled Opinionated and Knowledgeable Movie Discussions for Training Neural Conversation Models. In *Proceedings of the 12th Language Resources and Evaluation Conference*, Marseille, France / online.

Godfrey, J. J., Holliman, E. C., and McDaniel, J. (1992). SWITCHBOARD: Telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, IEEE International Conference On*, pages 517–520. IEEE Computer Society, March.

Gray, J., Srinet, K., Jernite, Y., Yu, H., Chen, Z., Guo, D., Goyal, S., Zitnick, C. L., and Szlam, A. (2019). CraftAssist: A Framework for Dialogue-enabled Interactive Agents. *arXiv:1907.08584 [cs]*, July.

Gurion, T., Healey, P. G., and Hough, J. (2018). Real-time testing of non-verbal interaction: An experimental method and platform. In *Proceedings of the 22nd Workshop on the Semantics and Pragmatics of Dialogue - Poster Abstracts*, Aix-en-Provence, France. SEMDIAL.

Haber, J. and Poesio, M. (2020). Classification of Low-Agreement Pronouns Through Collaborative Dialogue: A Proof of Concept. In *Proceedings of the 24th Workshop on the Semantics and Pragmatics of Dialogue - Full Papers*, Virtually at Brandeis, Waltham, New Jersey. SEMDIAL.

Healey, P. G. T., Purver, M., King, J., Ginzburg, J., and Mills, G. J. (2003). Experimenting with Clarification in Dialogue. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 25.

Ilinykh, N., Zarrieß, S., and Schlangen, D. (2019). MeetUp! A Corpus of Joint Activity Dialogues in a Visual Environment. In *Proceedings of the 23rd Workshop on the Semantics and Pragmatics of Dialogue (SemDial 2019 / LondonLogue)*, London, UK.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. (2017). CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997, Honolulu, HI, July. IEEE.

Krantz, J., Wijmans, E., Majumdar, A., Batra, D., and Lee, S. (2020). Beyond the Nav-Graph: Vision-and-Language Navigation in Continuous Environments. *arXiv:2004.02857 [cs]*, May.

Manuvinakurike, R. and DeVault, D. (2015). Pair Me Up: A Web Framework for Crowd-Sourced Spoken Dialogue Collection. In G.G. Lee, et al., editors, *Natural Language Dialog Systems and Intelligent Assistants*, pages 189–201. Springer International Publishing, Cham.

Maraev, V., Mazzocconi, C., Mills, G., and Howes, C. (2020). "LOL what?": Empirical study of laughter in chat based dialogues. *Laughter and Other Non-Verbal Vocalisations Workshop: Proceedings (2020)*, October.

Miller, A., Feng, W., Batra, D., Bordes, A., Fisch, A., Lu, J., Parikh, D., and Weston, J. (2017). ParlAI: A Dialog Research Software Platform. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 79–84, Copenhagen, Denmark, September. Association for Computational Linguistics.

Oertel, C., Cummins, F., Edlund, J., Wagner, P., and Campbell, N. (2013). D64: A corpus of richly recorded conversational interaction. *Journal on Multimodal User Interfaces*, 7(1):19–28, March.

Paetzel, M., Racca, D. N., and DeVault, D. (2014). A Multimodal Corpus of Rapid Dialogue Games. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4189–4195, Reykjavik, Iceland, May. European Language Resources Association (ELRA).

Rachman, L., Liuni, M., Arias, P., Lind, A., Johansson, P., Hall, L., Richardson, D., Watanabe, K., Dubal, S., and Aucouturier, J.-J. (2018). DAVID: An open-source platform for real-time transformation of infra-segmental emotional cues in running speech. *Behavior Research Methods*, 50(1):323–343, February.

Schlangen, D., Diekmann, T., Ilinykh, N., and Zarrieß, S. (2018). Slurk – A Lightweight Interaction Server For Dialogue Experiments and Data Collection. In *Short Paper Proceedings of the 22nd Workshop on the Semantics and Pragmatics of Dialogue (AixDial / Semdial 2018)*, Aix-en-Provence, France.

See, A., Roller, S., Kiela, D., and Weston, J. (2019). What makes a good conversation? How controllable attributes affect human judgments. In *Proceedings of the 2019 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 1702–1723, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Sigurdsson, G. A., Varol, G., Wang, X., Farhadi, A., Laptev, I., and Gupta, A. (2016). Hollywood in Homes: Crowdsourcing Data Collection for Activity Understanding. *ECCV*.

van Son, R., Wesseling, W., Sanders, E., and van den Heuvel, H. (2008). The IFADV Corpus: A Free Dialog Video Corpus. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

Zarrieß, S., Hough, J., Kennington, C., Manuvinakurike, R., DeVault, D., Fernández, R., and Schlangen, D. (2016). PentoRef: A Corpus of Spoken References in Task-oriented Dialogues. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 125–131, Portorož, Slovenia, May. European Language Resources Association (ELRA).