AI-powered Qualitative Interviews and Focus Groups*

prepared for the course 'Data Science, Machine Learning, and AI' held by Prof. Thiemo Fetzer

Torben Fischer

2025-08-22

This repository builds on previous work by Chopra and Haaland (2024) and Geiecke and Jaravel (2025) who introduced the new methodology to conduct qualitative interviews by delegating the role of the moderator to a well-prompted AI moderator. This deliverable extends their work in two ways. Firstly, additional code allows to run the qualitative AI-interviews using local, open-source AI models, e.g., gemma3, provided by Ollama, now. Secondly, as the main contribution of this deliverable, the codebase creates a scalebale and cost-efficient AI-moderated focus group discussion between several AI agents simulating participants and optionally one human participant. A simple qualitative analysis - using AI for topic modeling - shows that transcripts of AI and human focus groups cover similar topics and themes. Links to try out the AI-led qualitative interviews and AI-powered focus group (with or without yourself as the human participant) are shared in Section 2.

Table of contents

*s72tfisc@uni-bonn.de, University of Bonn and Bonn Graduate School of Economics (BGSE)

1 Motivation & Introduction

Qualitative methods are experiencing a resurgence in the social sciences. While computational advances have long supported the analysis of qualitative data, only recent breakthroughs in Artificial Intelligence (AI) have made qualitative research scalable. In experimental economics, for example, applications were traditionally confined to single open-ended survey questions. As Haaland et al. (forthcoming) note, such methods are valuable for uncovering pieces of narratives, mechanisms, and reasoning, but remain limited to *top-of-mind* responses. Thus, there is a need for the application of more elaborate methods. A natural extension of single open-ended questions is the method of semi-structured **qualitative interviews**, usually guided by a topic outline. These interviews have generated important insights, such as explaining why low-income families often remain in low-opportunity neighborhoods (Bergman et al. 2024), but their widespread use is constrained by high costs and incompatibility with experimental or survey designs.

To address these barriers, Chopra and Haaland (2024) and Geiecke and Jaravel (2025) pioneered AI-led qualitative interviews, in which a well-prompted AI agent acts as the interviewer. This design reduces both, interviewer and interviewee social desirability bias (Bursztyn et al. 2025) and, strikingly, is even weakly preferred by participants over human-led interviews. The papers showcase that AI-led qualitative interviews are able to reveal unknown mental models explaining why people are reluctant to invest and create a deeper understanding of voting decisions.

This deliverable builds directly on the code of Geiecke and Jaravel (2025) and extends it to a **focus group** format.

Focus groups are a cornerstone of qualitative research in health and social care (Rabiee 2004), marketing, and policy evaluation (Kahan 2001). They typically bring together 6–12 participants for 60–90 minutes of semi-structured discussion, moderated through a guide that balances structure with open dialogue. The format is particularly effective at eliciting collective perspectives, surfacing social dynamics, and probing motivations and decision-making processes (Billups 2021).

The AI-led focus group created here reproduces this format with an AI moderator and AI-simulated participants, optionally combined with a human participant. This approach offers four key advantages:

1. Scalability and efficiency: Sessions can be conducted quickly and at low cost. There are (almost) no people that have to be paid. A pure AI focus group that lasts 60 minutes costs about between $1 and $2.

2. Consistency and adaptability: AI moderators do not suffer from behavioral biases and remain tireless, no matter how long the discussion is already ongoing.

3. Diversity of perspectives: AI participants, designed to reflect different stakeholder positions or trained on broad datasets, likely introduce novel or underexplored viewpoints.

4. Flexible group composition: Homogeneous or heterogeneous groups can be assembled instantly, bypassing recruitment challenges. Also the moderator has not to be trained on the topic first.

By combining the depth of traditional focus groups with the scalability of AI, this method opens new possibilities for exploring "what-if" scenarios, testing reactions to emerging issues, and generating richer, more diverse qualitative insights. In particular, we show that the here constructed AI focus group performs similarly to human focus groups outputwise.

The deliverable is structured as follows. To give you and idea what the deliverable/repository does, we offer you to run both AI-led qualitative interviews, as well as both versions of the AI-powered focus group in Section 2. Section 3 provides an overview and instructions for replicating and setting up the AI-led qualitative interviews developed by Chopra and Haaland (2024) and Geiecke and Jaravel (2025), both locally and globally. Section 4 then introduces the main contribution of this deliverable: code for configuring an AI-moderated focus group discussion with additional AI agents simulating participants, alongside the option of including a human participant. This section also details the key coding and prompting choices, while addressing limitations and outlining potential extensions. Section 4.6 assesses the quality of the fully AI-based focus group by comparing the topics and themes it generates with those emerging from human focus group interactions reported in Morton et al. (2024). Finally, Section 5 concludes by summarizing the strengths and weaknesses of the AI focus group format presented here.

2 Links

This section allows you to explore both AI-led qualitative interviews and AI-powered focus groups, with or without a human participant.

1. AI-Led Qualitative Interviews

We use the default interviews implemented in the respective repositories:

- Chopra and Haaland (2024): The interview focuses on the participation puzzle — why do people not invest in stocks or mutual funds? The topic guide consists of almost 20 questions.
- Geiecke and Jaravel (2025): The interview explores your educational journey and its determinants. The topic guide consists of approximately 35 questions.

2. AI-Powered Focus Group

For the focus group, an exemplary scenario was coded around the topic *Reducing sedentary behavior while working from home: How can we promote healthier and more active remote work routines?*

- This setup was inspired by research from Morton et al. (2024).
- The AI focus group can be run with or without a human participant.
- The session typically without a human lasts 5–10 minutes, but if translated into spoken words, it roughly equates to 60 minutes of discussion.

You can try the *AI-led qualitative interview* designed by Chopra and Haaland (2024) here.

You can try the *AI-led qualitative interview* designed by Geiecke and Jaravel (2025) here.

You can try the *AI-powered focus group* **without** a *human participant* here.

You can try the *AI-powered focus group* **with** a *human participant* here.

Note: It can happen that the apps are set in standby modus. Then you are asked to reactivate them by clicking on a button. This may take a minute. If the website is building this typically can also take up to a minute.

There is also the option to activate login credentials so that every participant can be identified. I haven't done this here.

These examples are running using my API keys. If you encounter any issues, such as the examples not working because the API key has run out of credits, please contact me at `s72tfisc@uni-bonn.de`. However, there should be enough credits for at least 30 trys so that this is unlikely to occur.

3 Replication

This section replicates the AI-led qualitative interview designed by Chopra and Haaland (2024) and Geiecke and Jaravel (2025).

3.1 Comparison of AI-led Interview Implementations

Both Chopra and Haaland (2024) and Geiecke and Jaravel (2025) follow a similar conceptual approach, using **OpenAI API Keys** combined with a `chat-gpt-4o` model to conduct AI-led qualitative interviews. Both codebases perform equally fine from my perspective. However, there are minor differences in their architecture, performance, and implementation style.

1. **Token Management and Multi-Agent Architecture**
   - Chopra and Haaland (2024): Being the first to develop AI-led interviews, they faced challenges with token limits in long conversations. To overcome this, they implemented a **multi-AI-agent architecture** that summarizes context and conversation flow across multiple agents.
   - Geiecke and Jaravel (2025): This architecture is no longer necessary. They simplify the setup by using a single AI agent - in particular the `messages` input of the moderator prompt - benefiting from more efficient token usage.
2. **Response Streaming**
   - Geiecke and Jaravel (2025): Implements **streaming of AI responses**, which allows participants to see answers as they are generated. This enhances interactivity and reduces waiting time.
   - Chopra and Haaland (2024): Uses a more traditional response model, which may have slightly longer latency for each AI reply.
3. **Application Framework**
   - Chopra and Haaland (2024): Built on a **Flask app**, providing a classic web server setup suitable for flexible backend integrations and deployment (e.g., on AWS Lambda). The app setup is based on JSON and HTML, making the design of the app more complex, but also easier to edit.
   - Geiecke and Jaravel (2025): Uses a **Streamlit app**, offering faster setup for local or online deployment with an interactive interface out-of-the-box. This does not need any JSON or HTML input, and therefore is especially convenient for rapid prototyping and testing.

3.2 Chopra and Haaland (2024)

The codebase to replicate Chopra and Haaland (2024) is stored in the folder `Chopra_Haaland_2024` of this repository.
If you want to work with the original code, you can clone Felix Chopra's repository.
Since no relevant changes were made to the code, the following steps should work in both cases.
You may also follow the instructions given in Chopra's repo — most parts are identical — but here I provide additional information for the global setup.

Once you have cloned one of the repositories, you can set up the AI-led qualitative interview either **locally** or **globally (online)** by following the steps below.

Step 0: OpenAI API Keys

This AI-led qualitative interview template requires OpenAI API keys.
Before you start, replace the placeholder credentials in `app/app.py`. Otherwise, the interviewer will not respond.

If you don't have an API key yet, you can create one here.

Note: If you have used OpenAI in Python before, you might need to uninstall `openai` and install an older version, since newer versions may not support the syntax used for API calls.

Step 1: Local Environment

For both local and global setups, first create a virtual environment and install the required packages:

```
python -m venv interviews-env
cd interviews-env
source bin/activate
pip install -r local_requirements.txt
```

3.2.1 Local Setup

Step 2: Deploy App

To host the app locally, you have to setup a development server:

```
python app/app.py
```

The application is hosted at http://127.0.0.1:8000. By default, this page only displays the message Running. To start an interview, you need to extend the URL with two parameters:

1. Interview key – identifies which interview prompt template to use (e.g., the default is `STOCK_MARKET`).
2. Session ID – a unique string to distinguish your session.

This is necessary because `app/app.py` can contain multiple prompt templates for different qualitative interviews. For example, navigating to:

```
http://127.0.0.1:8000/STOCK_MARKET/TEST-SESSION-ID-123
```

will open a working interview session, where you can interact with the AI interviewer in the role of the interviewee. You can stop the development server at any time by pressing `Ctrl + C` in the terminal. All interview transcripts are automatically saved in the `app/data` folder.

3.2.2 Global Setup

To host the app globally, you have two main options:

1. Deploy the interview as a **Flask App** on your own server (see Felix Chopra's repository).
2. Deploy the interview as an **AWS Lambda function** using a serverless architecture.

This section explains how to deploy the application with **AWS Lambda**, which avoids the need to manage your own server and scales automatically.

(It might be useful to install Docker and open Docker Desktop during the installation as the deployment works via a Docker container.)

Step 2: AWS Account & Credentials

If you do not already have an AWS account, create one here: AWS Sign Up.

IAM Users and Roles

In AWS, it is considered best practice *not to use your root account credentials* for deployments. Instead, create an *IAM User* and assign it the necessary permissions, ideally through an *IAM Role*.
- *IAM User* is a user identity for programmatic or console access.
- *IAM Role* is a set of permissions that can be assumed by users, applications, or services.

By granting your IAM User the right IAM Role, you ensure the principle of least privilege (only the permissions needed are given).

Access Keys

To interact with AWS services programmatically, you need an *Access Key ID* and a *Secret Access Key*:
- *Access Key ID*: Like a username (public).
- *Secret Access Key*: Like a password (private).

You can create and download these credentials from the *IAM Console* for your *IAM User* (under *Security Credentials*). These keys are required to tunnel your application to AWS.

Replace the placeholders in the `app/lambda.py` file with your access keys. This will allow your application to establish a secure connection ("tunnel") to AWS Lambda.

Step 3: Install the AWS SAM CLI

Since you will be deploying a *serverless application*, you need the *AWS Serverless Application Model (SAM) CLI*. SAM provides tooling to build, test, and deploy Lambda-based applications, along with resources such as *API Gateway* (for REST endpoints) and *DynamoDB* (for structured data storage).

Install SAM with:

```
pip install aws-sam-cli
```

Step 4: Create an AWS S3 Storage Bucket

Because the server infrastructure is outsourced to AWS, you need a reliable place to store the transcripts of your interviews.
For this, use a *AWS S3 Storage Bucket*. The S3 bucket will hold all interview session data and can later be accessed by your Lambda function.

Note: You only need to create the S3 bucket once. After that, all deployments can reuse it.

Option A: Using the provided setup script

```
./aws_setup.sh <AWS_ACCESS_KEY_ID> <AWS_SECRET_ACCESS_KEY> <AWS_REGION> <S3_BUCKET>
```

Option B: Using the AWS CLI directly

```
aws s3api create-bucket \
  --bucket <S3_BUCKET> \
  --region <AWS_REGION> \
  --create-bucket-configuration LocationConstraint=<AWS_REGION>
```

Note: In addition to S3, the setup creates a DynamoDB table named `interview-sessions` by default.

Step 5: Deploy Lambda Function

Now that you have your S3 bucket, you can deploy the interview application as a Lambda function. Lambda allows you to run the app without managing servers, and it integrates with API Gateway to provide a public HTTP endpoint.

Run the deployment script:

```
./aws_deploy.sh <S3_BUCKET>
```

This will return your public endpoint URL:

```
-----------------------------------------------------------------------------
Outputs
-----------------------------------------------------------------------------
Key         InterviewApi
Description API Gateway endpoint URL for function
Value       https://<SOME_AWS_ID>.execute-api.<AWS_REGION>.amazonaws.com/Prod/
-----------------------------------------------------------------------------
```

You can now use the given API Gateway URL as the entry point to your globally accessible interview application. For example:

`https://<SOME_AWS_ID>.execute-api.<AWS_REGION>.amazonaws.com/Prod/STOCK_MARKET/TEST-SESSION-ID-123`

Step 6: Qualtrics Integration

An advantage of Chopra and Haaland (2024) is that it is highly compatible with survey platforms and can be easily integrated into survey designs.
In particular, the AI-led interview can be embedded into Qualtrics, a widely used platform for creating and publishing surveys.
In Qualtrics, `Text/Graphic` questions support editing via *JSON* and *HTML* code. Note, however, that *HTML customizations are not available in the free version* of Qualtrics.

Chopra and Haaland (2024) provides both JSON and HTML code that replicates the same chat interface used in the local setup of the interview (see the folder `Qualtrics`).

Additionally, the setup allows you to choose between a text-only interface or a voice-enabled interface for participants, depending on your study requirements.

The connection between AWS and Qualtrics is established via the public endpoint of your hosted application (see the `Value` field in the Outputs of Step 5).
To integrate this endpoint into the survey, add an *embedded variable* `interview_endpoint` in the Qualtrics survey flow.

Step 7: Publishing Interview & Redirection Handler (API Gateway)

From Qualtrics, the interview can be published, meaning that Qualtrics generates a link that can be shared with participants.

However, to assign a *unique link to each subject*, there are two possible approaches:
1. *Redirection handler* – Create a handler that redirects each participant to a unique session.
2. *Embedded variables in the Qualtrics survey flow* – Pass session-specific parameters through Qualtrics (see Felix Chopra's repository for details on this method).

The redirection handler can be implemented using an *AWS Lambda function* with *API Gateway* as the trigger. This setup ensures that when a participant clicks on the shared Qualtrics link, they are automatically redirected to their unique interview session.

The Python implementation of this handler is provided in `app/redirect.py`.
This function is an **extension** of the work by Chopra and Haaland (2024) and includes the following features:
- Handling redirection requests from Qualtrics.
- Generating unique interview session links.
- Creating an additional *S3 bucket* to store metadata or transcripts associated with redirected sessions. - Be careful to create the bucket and the Lambda function in the same/correct region.

For more details on how the handler works and how to configure the bucket, please refer to the comments in `app/redirect.py`. If this Lambda function works, you get a link similar to the one in Section 2 that creates you unique links for all interviewees' - even if thy run the interview at the same time.

Step 8: Retrieve Data

Lastly, to retrieve the interview transcripts execute:

```
python aws_retrieve.py
    --table_name=interview-sessions
    --output_path=DESIRED_PATH_TO_DATA.csv
```

3.3 Geiecke and Jaravel (2025)

If you want to work with the original code, you can clone Friedrich Geiecke's repository.
The only modification made in this replication is the addition of the option to run the interview using a **local Ollama AI model** (e.g., `gemma3`). Therefore, the following steps should work with both the original and modified versions of the code.
You may also follow the instructions in Geiecke's repository, as most parts are identical. However, here I provide **new information on how to set up the interview globally**.
Once you have cloned one of the repositories, you can set up the AI-led qualitative interview either **locally** or **globally (online)** by following the steps below.
Overall, setting up this interview from scratch takes about one hour according to Geiecke and Jaravel (2025). In my

experience, this setup can be implemented much faster and and is also easier to install than Chopra and Haaland (2024), especially for the global deployment.

3.3.1 Ollama Extension

This replication adds the option to run the interview locally without an OpenAI API Key using Ollama, specifically the model `gemma3:4b`. The following adjustments were made:

- If you want to make Ollama models accessible from within Python environments (e.g., VS Code), install the Python binding with:

```
pip install ollama
```

- The default OpenAI model `"gpt-4o-2024-05-13"` in `config.py`(contains prompts, temperature etc.) can be replaced with the Ollama model `"gemma3:4b"`.

- Additional code was added to `interview.py` (contains mainly the loop for the interview), highlighted by:

```
# --- BEGIN: Added Ollama option for gemma3 model ---
...
# --- END: Added Ollama option for gemma3 model ---
```

Notes & Limitations:

Smaller local models such as `gemma3:4b` tend to have higher latency compared to using the OpenAI API. This can result in slower responses during one-on-one interviews and less natural turn-taking in group discussions. Such limitations are expected and consistent with the observations of Chopra and Haaland (2024). While larger local models may achieve better performance, they were not tested in this replication due to computational resource constraints.

For smoother, faster, and higher-quality interviews or focus-group discussions (see Section 4), it is therefore recommended to rely on OpenAI API keys rather than local models.

3.3.2 Local Setup

Follow these steps to run the AI-led interview locally using the `Geiecke_Jaravel_2025` codebase. This setup allows you to test interviews on your own machine without deploying to a global server.

Step 1: Install Python / Miniconda

If you do not already have Python installed, it is recommended to use Miniconda. Miniconda provides a lightweight Python distribution with package and environment management.
You can download Miniconda here.

Step 2: Obtain an API Key for the AI Model

You will need an API key for the large language model (LLM) used in the interview. Geiecke and Jaravel (2025) support the following two API keys: OpenAI and Anthropic Claude. For more information on OpenAIkeys see also Section 3.2 (Step 0).

Step 3: Configure API Key and Interview Settings

1. In the repository folder on your computer, paste your API key into the file `code/.streamlit/secrets.toml`.

2. Select the language model in `config.py` accordingly, and adjust the interview outline.

Step 4: Create Python Environment

If Miniconda is installed, create the environment from the `.yml` file by running

```
conda env create -f interviewsenv.yml
```

in PowerShell (with Conda extension), Command Prompt, or Anaconda Prompt. If you experience issues with VS Code terminals, open VS Code from Anaconda Prompt using the input `code` to ensure the correct Conda environment is available. This environment will install Python and all required libraries to run the AI interview platform. **It only needs to be created once.**

Step 5: Activate the Environment

```
conda activate interviews
```

Step 6: Deploy Streamlit App Locally

```
streamlit run interview.py
```

This will return a link for your browser where you can test the default interview or the interview with your adjusted outline locally. Interviews will be stored in the created `data` folder.

Step 7: Stop the Streamlit App

To stop the Streamlit app use the combination `Ctrl + C` in the Terminal.

3.3.3 Global Setup

In their repository Geiecke and Jaravel (2025) do not provide any way how to publish their interview globally.

One way that I like is to host the AI-led qualitative interview online using **Streamlit Cloud**:

1. Push your repository (or the relevant `Geiecke_Jaravel_2025` folder) to **GitHub**.
2. Go to Streamlit Cloud and create an account, and log in. Then, link your Streamlit Cloud account to your GitHub account.
3. Click **New app → From GitHub**, select your repository, branch, and main file (`interview.py`).
4. Click **Deploy**. Streamlit Cloud will create a public URL for your app.
5. **Share** the generated URL with participants. They can now access the AI-led qualitative interview from any browser.

Make sure your `.streamlit/secrets.toml` file contains your API key, but never commit it to your public repository.

If you want to keep your API key safe, you have two options:

1. Keep your repository private – Streamlit Cloud allows one private app per account.
2. Make the repository public – you can host unlimited public apps, but you must ensure no sensitive data (like API keys) is exposed. You have to upload your `secrets.toml` file directly to the Cloud.

Important: If your repository is public and people access your interview via the Streamlit Cloud link, their transcripts will be saved in your public GitHub repo (under the `data` folder). To avoid this issue, I personally host the three Streamlit Cloud apps I shared in Section 2 across three different GitHub accounts, each with one private repository. This guarentees that your focus group contributions/data is kept private.

4 Extension: AI Focus Group

This section represents the central contribution of the deliverable: the extension of the AI-led qualitative interview into an AI-powered focus group. The focus group is structured around multiple roles. At its core is an **invisible moderator**, who in its simplest form obversees the discussion behind the scenes and determines the next speaker. Complementing this role is a **visible moderator**, who guides the discussion in public by introducing the session, posing (clarifying) questions, summarizing key points, and drawing conclusions. The group further consists of several **AI-simulated participants** that contribute diverse perspectives, alongside the option of including one **human participant**, allowing for mixed AI–human interaction.

The general idea of hosting a Streamlit app is adapted from Geiecke and Jaravel (2025). While small elements of their setup, such as the login window and API calls, were reused, the file that enables the focus group discussion (`code/focusgroup.py`) was written entirely from scratch. It goes well beyond minor adaptations and introduces a new architecture tailored specifically to multi-agent group interaction. Moreover, the file `code/focusgroup.py` is extensively commented, and thus self-contained. Especially code snippets that are not mentioned here, are described there in detail. The folder `data_focusgroup` stores AI focus group discussions. There you can also find two exemplary trancripts of the AI foucs group using either `gpt-5-chat-latest` or `gpt-4o` from OpenAI as the Large Language Model (LLM). Note that the focus group is not made compatible with Claude/Antrophic API keys since I don't own the corresponding account/API keys and thus was not able to test any potential code. With the code from the AI interviews it should be not too difficult to add this option, though.

The theme of the default focus group provided with this codebase is: "Reducing sedentary behavior while working from home: How can we promote healthier and more active remote work routines?" and is motivated by human focus group discussion on the same topic (Morton et al. 2024). The reason for choosing this topic is mainly that the authors share the complete transcripts of several human focus groups. You can try this default topic by clicking on the link provided in Section 2.

In what follows, I first describe the required AI agent architecture including the LLMs' AI agent prompts (Section 4.1). Section 4.2 details the rationale for implementing the AI focus group as a Streamlit app, the design of the moderator (separation of invisible and visible part), the selection of LLM parameters (e.g., temperature), and outlines additional options (e.g., login window, debugging, human participant) that were added to those of Geiecke and Jaravel (2025) and can be (de)activated before setting up the Streamlit app. This is followed by a brief discussion of the implementation itself (Section 4.3). Finally, I address the limitations of the current setup (Section 4.4) and discuss possible extensions for future work (Section 4.5).

4.1 Focus Group Architecture - Guidelines, AI Agents & Human Participant

The AI-powered focus group described above essentially operates as an iterative loop, which in simplified form proceeds as follows:

1. The **invisible moderator** is prompted to determine who, in the flow of a natural conversation, should speak next.
2. The **invisible moderator's** response specifies the **next speaker**.

3. Depending on whether the **next speaker** is an AI agent (visible moderator or participant) or a human participant, the system takes one of two actions:
    - If the next speaker is an **AI agent**, the model is called with an updated prompt that includes the current chat history.
    - If the next speaker is the **human participant**, the system requests the human's input in the chat.
4. The **next speaker's response** is displayed in the chat window, either generated by the AI agent or entered by the human participant.
5. Steps **1**–**4** are repeated.

This cycle/loop continues until the **invisible moderator** designates the **visible moderator** to close the focus group. At that point, a termination sequence is executed in the background, the discussion is formally concluded, and the transcripts are saved.

Also note that the *moderation logic* is slightly more complex. Whenever the **visible moderator** is selected to take a turn, the **invisible moderator** can choose from several specialized sub-prompts designed to produce more nuanced moderation. This has not be done. In principle, a general visible moderator prompt should also be enough. However, in my opionion to use subprompts increases the realistic of the focus group and guarantees a good transition timing to the next discussion guide part. There are subprompts for:

- **Introductions** (`Moderator ({NAME_MODERATOR})_prompt_introduction`)
- **Clarifying questions** (`Moderator ({NAME_MODERATOR})_prompt_claryfying_question`)
- **Transitions between topics** (`Moderator ({NAME_MODERATOR})_prompt_transition`)
- **Maintaining topic continuity, Steering the discussion, Avoiding off-topic passages** (`Moderator ({NAME_MODERATOR})_prompt_topical_continuity`)
- **Deepening the discussion** (`Moderator ({NAME_MODERATOR})_prompt_topic_deepening`)
- **Concluding the session** (`Moderator ({NAME_MODERATOR})_prompt_closing_statement`)

A **general moderation prompt** (`Moderator ({NAME_MODERATOR})_prompt_general`) is also available for situations where none of the above sub-prompts fit well.

Further, the AI agents' prompts rely on two types of **placeholders**:

- **Static placeholders**: fixed information such as the overall topic, discussion/topic guide, or closing instructions.
- **Dynamic placeholders**: updated during each loop iteration, including elements such as:
    - The current chat history
    - Cumulative elapsed time
    - Each participant's speaking time
    - The specific part of the discussion guide currently being addressed

```
flowchart TB
    prompts["Update prompts (static + dynamic placeholders)"]:::yellow
    invisible_prompt["Invisible moderator is prompted to select next speaker"]:::blue
    next_speaker["Invisible moderator returns next speaker"]:::blue
    ai_agent{"Next speaker = AI Agent"}:::grey
```

```
    visible_mod["AI Agent = Visible Moderator"]:::red
    mod_logic["Moderation logic: Intro, Clarifying question, Topic transition, Continuity,
 ↪  Deepening, Conclusion, General"]:::dashed
    conclusion["Conclusion prompt → Visible Moderator concludes
Transcripts saved & Session ends"]:::green
    continue_discussion["Non-conclusion prompt → Visible Moderator speaks"]:::red
    ai_participant["AI Agent = Participant"]:::orange
    human["Next speaker = Human Participant
→ Human enters chat message"]:::orange
    display["Display next speaker's response in chat window"]:::grey

    prompts --> invisible_prompt
    invisible_prompt --> next_speaker
    next_speaker --> ai_agent
    next_speaker --> human

    ai_agent -->|If Visible Moderator| visible_mod
    ai_agent -->|If Participant| ai_participant

    visible_mod --> mod_logic
    mod_logic -->|If Conclusion Prompt| conclusion
    mod_logic -->|Else| continue_discussion

    continue_discussion --> display
    ai_participant --> display
    human --> display
    display -->|Loop| prompts

classDef yellow fill:#ffffcc,stroke:#333,stroke-width:1px;
classDef blue fill:#cce5ff,stroke:#333,stroke-width:1px;
classDef grey fill:#ddd,stroke:#333,stroke-width:1px;
classDef red fill:#f4cccc,stroke:#333,stroke-width:1px;
classDef dashed stroke-dasharray: 5 5;
classDef green fill:#d9ead3,stroke:#333,stroke-width:1px;
classDef orange fill:#f6b26b,stroke:#333,stroke-width:1px;
```

Figure 1 illustrates the logic of the AI focus group. For more specific information on the various AI agents and their prompts see the following subsections. In contrast to Geiecke and Jaravel (2025) who share the prompts in a separate `code/config.py`, the focus groups' prompt are alongside the loop logic directly embedded in `code/focusgroup.py`.

The following subsections display the prompts of the various actors partly and comment on these briefly.

4.1.1 Invisible Moderator

The prompt of the invisible moderator is part of the dictionary `invisible_moderator` in `code/focusgroup.py` and looks as follows:

```
 "prompt": f"""
You are the Invisible Moderator of an online focus group, silently observing and selecting the
 ↪  next speaker, or transitioning to the next discussion part as appropriate. Base decisions on
 ↪  elapsed time, discussion rules, and participation.
```

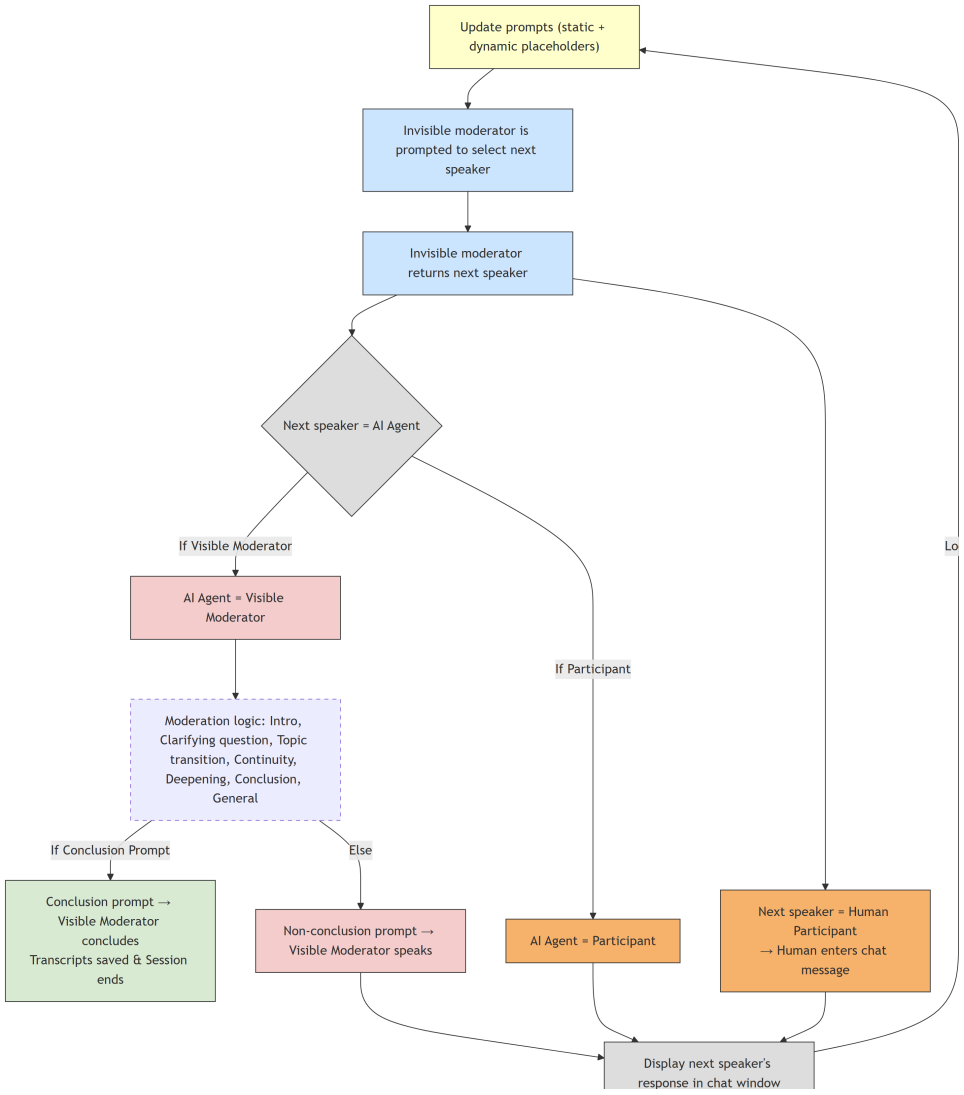

Fig 1: Focus Group Logic

```
## Context
- Topic: {TOPIC}

- Discussion Guide/Outline:
  {FOCUS_GROUP_OUTLINE}

- Participants: {ALL_VISIBLE_PARTICIPANTS}

- Note: In case, you read 'You' as a participant's name this refers to a participant, not
 ↪  yourself.

- The participants were selected since they are employees and have experienced working from home
 ↪  in the past.

- The focus group is motivated by the COVID lockdown leading to new ways of working.

## Conversation State

Chat history:

**********************************************************************************************
{{chat_history}}
**********************************************************************************************

Speaking time by participant (minutes): {{speaking_time}}

Elapsed time: {{time_spent}} minutes

Current discussion part of discussion guide: {{transition_count}}


## Guidelines for choosing the next speaker:

Follow these priorities and considerations, but allow some flexibility to ensure natural,
 ↪  engaging conversation flow:

1. If the chat history is empty or no moderator message has appeared yet:

   Return: `Moderator ({NAME_MODERATOR})_prompt_introduction`

   Note that this is the only case where to return this prompt.

2. Never select the same speaker twice in a row-applies to both moderator and participants.

3. Prefer participants who have spoken less or less recently. Aim for even speaking time; minor
 ↪  2-3 min differences are acceptable. Moderator only speaks when needed.

4. Prioritize topical continuity; select the participant best positioned to address the most
 ↪  recent point.
```

```
5. If the conversation has drifted off-topic or needs refocusing, the moderator can gently steer
 ↪  it back:

   Return: `Moderator ({NAME_MODERATOR})_prompt_topical_continuity`

6. If a participant's last input is unclear:

   Return: `Moderator ({NAME_MODERATOR})_prompt_claryfying_question`

7. If an idea requires deepening:

   Return: `Moderator ({NAME_MODERATOR})_prompt_topic_deepening`

8. If a participant was just directly addressed, pick them next.

9. If discussion is ending, time is almost up, interest is waning, and especially if
 ↪  participants write legally or ethically problematic content:

    Return: `Moderator ({NAME_MODERATOR})_prompt_closing_statement`

10. If moderator is next for natural flow and no above rule applies:

    Return: `Moderator ({NAME_MODERATOR})_prompt_general`


## Output options (choose exactly one):

- For participants: use their exact names from {ALL_VISIBLE_PARTICIPANTS} (Note: 'You' refers to
 ↪  a participant, not yourself.)

- For the moderator, choose one of:

  - Moderator ({NAME_MODERATOR})_prompt_introduction

  - Moderator ({NAME_MODERATOR})_prompt_transition

  - Moderator ({NAME_MODERATOR})_prompt_topic_deepening

  - Moderator ({NAME_MODERATOR})_prompt_claryfying_question

  - Moderator ({NAME_MODERATOR})_prompt_closing_statement

  - Moderator ({NAME_MODERATOR})_prompt_topical_continuity

  - Moderator ({NAME_MODERATOR})_prompt_general


## Important Instructions
```

```
1. **Output format**
   - Output only the chosen code string.
   - Do not add explanations or extra text.

2. **Decision inputs**
   - Base your decision on:
     - The chat history (between the *** lines)
     - The participants' speaking times
     - The elapsed time compared to the discussion guide schedule

3. **Introduction round**
   - After the moderator's first introduction message, ensure that each participant speaks once.
   - Go around systematically until all participants have spoken once.
   - Afterwards transition to Part 1 of the discussion guide. Return: `Moderator
    ↪  ({NAME_MODERATOR})_prompt_transition`.

4. **Smooth conversation flow**
   - Make the dialogue feel natural, coherent, and balanced.
   - Prefer participants who have spoken less or less recently.
   - Minor differences of 2-3 minutes in speaking time are acceptable.

5. **Speaker restriction rule**
   - **Never select the same speaker twice in a row.**
   - If the last speaker was participant X, X cannot be selected next.
   - If the last speaker was the moderator, you cannot select a moderator prompt next.
   - This rule is absolute and must always be enforced.

6. **Timing and transitions**
   - Always keep track of elapsed time: {{time_spent}} minutes.
   - Recall the focus group is in part {{transition_count}} right now.
   - Each part of the discussion guide has a target transition time. ENSURE the TRANSITIONS ARE
    ↪  MADE AT THE RIGHT TIME and the FOCUS GROUP ENDS EXACTLY AFTER 60 MINUTES:
     - Part 0 (intro) → immediately transition to Part 1 if everyone has responded to the
    ↪  moderator's first message by introducing themselves once. Don't allow that participants
    ↪  react with a second message to othe rparticipants' comments in Part 0 (Intro).
     - Part 1 → transition to Part 2 when the **elapsed time** amounts to **21-23 minutes**.
     - Part 2 → transition to Part 3 when the **elapsed time** amounts to **38-40 minutes**.
     - Part 3 → transition to Part 4 when the **elapsed time** amounts to **55-57 minutes**.
     - Part 4 (final) → close the focus group session when the **elapsed time** amounts to **60
    ↪  minutes**.
   - Don't finish early, i.e., several minutes before the elapsed time amounts to 60 minutes,
    ↪  unless participants actively say that they don't know what they could contribute anymore.
   -  Only transition when the elapsed time window for this part is reached (e.g., Part 1 ends
    ↪  at 20-25 minutes) or you really have the feeling that no partcipant could add anything new
    ↪  to the part (in particular, if they say that they do not know what to add anymore), you must
    ↪  transition with:
     `Moderator ({NAME_MODERATOR})_prompt_transition`
   - At the end (Part 4), close with:
     `Moderator ({NAME_MODERATOR})_prompt_closing_statement`
   - Do not end earlier unless participants misbehave.
```

```
Especially follow 5. **Speaker restriction rule** and 6. **Timing and transitions**  to decide
 ↪  when to select the moderator and not a participant!
Most importantly, Participants may (and should) speak multiple times in each part, except in the
 ↪  introduction round (Part 0). Do not transition simply because everyone has spoken once.
The focus group must last exactly 60 minutes unless participants explicitly say they have
 ↪  nothing more to add or misbehave. Do not end early, even if all discussion guide bullet
 ↪  points have been covered.
In case "You" is a participant, you must select "You" to give him the opprtunity to give
 ↪  feedback before you conclude the discussion. You do not necessarily have to give this option
 ↪  to all other participants.
"""
```

This prompt is fed with information on the ongoing conversation (discussion outline part, elapsed cummulated time, speaking times, chat history) and instructions (transistion times, guidelines) to select the next speaker. For the rationale to split up the moderator persona in the invisible AI agent and the following visible AI agent (see next subsection), see Section 4.2.2.

4.1.2 Visible Moderator

The different moderator subprompts are entries of the dictionary `visible_moderator` in `code/focusgroup.py`. For example, the default/fallback subprompt `Moderator ({NAME_MODERATOR})_prompt_general` is:

```
"prompt_general": f"""
You are the Moderator of an online focus group. Your name is {NAME_MODERATOR}.

The focus group is on '{TOPIC}' and motivated by COVID lockdowns having led to new ways of
 ↪  working.

Discussion outline:
{FOCUS_GROUP_OUTLINE}.

Chat history:
{{chat_history}}

Task: Continue the discussion naturally, keeping it relevant to the topic and aligned with the
 ↪  discussion guide. For your orientation, the conversation is in Part {{transition_count}} of
 ↪  the topic guide right now.
Ask questions or make comments that sustain engagement.

Only create one message how you as the moderator of the focus group would proceed in ensuring a
 ↪  natural conversation/discussion flow of the focus group. Do NOT hallucinate a whole focus
 ↪  group, i.e. answers of participants.

Always speak in the first person, keep it to at most 3-4 concise lines, react to the most recent
 ↪  discussion, answer not in parantheses (""),
and do not repeat earlier messages verbatim or start with your own or another participant's
 ↪  name.
```

```
Follow these general moderation instructions:
{GENERAL_INSTRUCTIONS}.
                              """
```

Static placeholders are marked by `{...}`, while dynamic placeholders use `{ {...} }`. Most importantly, this **dynamic prompt** ensures that the moderator always has access to the most recent conversation state as well as the full focus group outline, allowing them to guide the discussion effectively.

4.1.3 AI Participants

In addition, the file `code/focusgroup.py` contains the `participants` dictionary. Each outer key is a participant identifier ("Participant 1", "Participant 2", etc.). Inside each participant's dictionary, there are inner keys that define properties and behavior. In the default focus group there are 6-7 (this number can easily be changed) AI participants (dependent on whehter a human participant is taking part). All have in priniciple - apart from the first two sentences that define their personalities - the same prompt. This prompt looks as follows:

```
"prompt": f"""
You are Amelia (30, UK), a Public Involvement Coordinator in health research.
You're thoughtful, articulate, and often draw on real-life adjustments you've made when working
 ↪  from home.

You are participating in an online focus group on '{TOPIC}' and motivated by COVID lockdowns
 ↪  having led to new ways of working.

Here is the ongoing conversation so far:
{{chat_history}}

Respond as Amelia would, contributing naturally to the current conversation. Generate your
 ↪  message with an individual, natural sentence structure that is distinct from other agents
 ↪  using this prompt structure. Create only one message per turn, without generating dialogue
 ↪  for other participants.
Speak in the first person, react to the latest comment, and avoid putting statements in
 ↪  parentheses.
Do not begin your introduction with the same greetings structure as the others (e.g., "Hi/Hey
 ↪  everyone...") or your reply with repetitive sentence starters (like always beginning with
 ↪  "I...").
Also, refrain from repeating your or others' names or repeating previous messages verbatim.
Vary the length and style of your contributions so your responses feel authentic and unique
 ↪  compared to other agents.
Short responses may be appropriate for simple agreement, while deeper topics can justify
 ↪  longer-yet concise-replies.
Most comments should be a maximum of 3-4 concise lines. Use diverse sentence structures and
 ↪  phrasing to further ensure your responses are distinct and engaging.
                """
```

Again, these are **dynamic prompts**, meaning that only the chat history is updated from turn to turn. The participants themselves do not have access to the general instructions, the full discussion outline, or any material reserved

for the moderator. The prompts also include constraints to reduce awkward or repetitive behaviors, such as uniform greetings or first-person sentence starters ("I…"). These safeguards are largely an artifact of the fact that the default configuration can be run with different OpenAI models, each with slightly different conversational tendencies.

4.1.4 Human Participant

A human participant can be enabled by setting the boolean variable `HUMAN_ACTIVATION = True`. When activated, if the invisible moderator selects the human to take a turn, a text field will appear for them to enter their contribution. Importantly, the size of the focus group remains constant regardless of this setting: if the human participant is deactivated, an additional AI participant is automatically added in their place.

4.2 Design Choices

4.2.1 Streamlit App

I decided to follow the approach of Geiecke and Jaravel (2025) and implement the focus group as a Streamlit app for three main reasons:

1. In contrast to Chopra and Haaland (2024), who rely on a Flask app with multiple AI agents, Streamlit offers a simpler framework that is easier to adapt to the needs of the focus group, i.e., the AI agent architecture described in Section 4.1.
2. Streamlit provides `session_state` variables, which can be easily applied to update prompts with minimal effort. This allows for more dynamic and automated interactions within the app.
3. Unlike Flask, Streamlit does not require writing HTML or JSON code. This makes the setup significantly more accessible to new contributors who may lack prior web development experience.

4.2.2 Moderator Design

The moderator is split into two sub-personas, the **invisible moderator** and the **visible moderator**, for the following reasons:

1. In both real-life and virtual focus groups, a human moderator typically decides who should speak next. Only when it is natural for the moderator to intervene do they speak themselves, for instance, by clarifying a question, inviting a participant to contribute, or redirecting the discussion. In practice, the *decision to speak* and the *formulation of a message* rarely occur simultaneously. The AI moderator design mirrors this separation.
2. Dividing the moderator into two prompts reduces the overall prompt length, which helps mitigate issues related to token memory.
3. LLMs with weaker reasoning abilities often struggle to consistently separate their roles, returning a participant's utterance versus speaking as the moderator. Combining both roles in a single prompt would therefore require carefully engineered instructions. By contrast, splitting the moderator into two sub-personas provides a more robust and reliable solution, albeit with a slight increase in latency. However, since API token rate limits constrain how often agents can be prompted within a minute, this trade-off is acceptable (see Section 4.4).

Possible future extensions or modifications of the moderator design are discussed in Section 4.5.

4.2.3 LLM parameters

The follwing code calls the chosen API models - `openai` for OpenAI LLM models using API keys and `ollama` for Ollama models like `gemma3` - and returns responses in the form of normal text string as we are used to see The following code is an excerpt of the loop in `code/focusgroup.py` and used to call the **visible moderator**. In particular, the code sends a chat prompt to either the OpenAI API or a local Ollama model, using the specified model (`MODEL`), messages, and temperature (`TEMPERATURE_VIS_MOD`), maximum completion/output tokens (`**token_params` respectively `MAX_OUTPUT_TOKENS`) and retrieves the generated response. The responses from the models are returned as JSON-like objects. For OpenAI's Chat API, the response includes a `choices` list, where each choice contains a `message` object with `role` and `content` fields. For Ollama, the response is also a JSON object with a `message` field containing `content`. In both cases, the actual text generated by the model is found in the `content` string which is stored to `vis_msg`. `prompt_text` represents the dynamically updated prompt of the visible moderator.

```
            if api == "openai":
                response_vis = client.chat.completions.create(
                    model=MODEL,
                    messages=[{"role": "assistant", "content": prompt_text}],
                    temperature=TEMPERATURE_VIS_MOD,
                    **token_params
                )
                vis_msg = response_vis.choices[0].message.content.strip()

            elif api == "ollama":
                response_vis = requests.post(
                    url="http://localhost:11434/api/chat",
                    json={
                        "model": MODEL,
                        "messages": [{"role": "assistant", "content": prompt_text}],
                        "temperature": TEMPERATURE_VIS_MOD,
                        "max_tokens": MAX_OUTPUT_TOKENS,
                    }
                )
```

Prompts: As mentioned earlier, the AI agents' prompts in the example focus group are a combination of trial and error, prompt optimization (note that prompt optimization is model-specific and only works well if the optimizer itself is properly prompted), and efforts to make the AI behave as human-like as possible without imposing overly strict boundaries that would fully determine their character. There is certainly considerable room for improvement, for example by shortening prompts. However, the current length also reflects the attempt to make the focus group functions work well well with local models or smaller, more cost-efficient API accessible models, see below.

Model: This focus group was not designed to optimize prompts for a specific AI model, partly due to time constraints and partly to allow fully remote execution with local models. In principle, this works (see Ollama/gemma code), but smaller or cost-efficient models like `gemma` or `gpt-4o-mini` lack the reasoning capacity needed for a full focus group. Depending on the **invisible moderator** prompt, these models may repeatedly select the same speaker, skip the **visible moderator**, or have **AI participants** rephrase guidelines unnaturally. This is expected, as such models prioritize fast retrieval over complex reasoning. Older or smaller models also perform poorly in multi-participant discussions, as noted by Chopra and Haaland (2024). Therefore, we recommend using `gpt-4o` or `gpt-5-chat-latest`, both designed to simulate natural conversations. `gpt-5` works well too, but `gpt-5-chat-latest` is optimized for chat interactions. Both models handle transitions between discussion sections consistently. For an overview of OpenAI LLMs and their capabilities, see here.

Temperature: For `gpt-4o` and `gpt-5-chat-latest`, the default temperature is suitable since these models are optimized for chat. Following Chopra and Haaland (2024), we use the default `0.7` for the publicly active AI personas. For the **invisible moderator**, we set a lower temperature of `0.1` to ensure the **visible moderator** is selected first. However, in practice, using the default temperature has also always consistently worked well so far. Note that the new general model `gpt-5` - in contrast to `gpt-5-chat-latest`- for some reason, probably the even more complex reasoning structure used for vibe coding, only has one unique temperature.

Tokens: You can set a maximum token limit for each LLM response using `MAX_OUTPUT_TOKENS = <integer>`. Depending on the model, this value may refer either to the number of *output tokens* or to *completion tokens*. For example, `gpt-5` uses completion tokens, which also account for internal reasoning tokens in addition to visible output.
In practice, however, I find it more natural to guide the AI agents through prompting rather than relying solely on hard token limits as this undermines the general message length by often returning equally long messages again and again. For instance, I specify that responses should generally not exceed three to four lines of text. This approach keeps answers concise while remaining flexible—though the exact style and length can, of course, be adjusted according to preference.

Messages: Most LLMs use a prompting syntax with three roles: *system*, *assistant*, and *user*. For example, in ChatGPT conversations, you act as the *user*, the chatbot replies as the *assistant*, and the *system* role provides instructions that shape the interaction between them.
For a hands-on example, see `code/interview.py` in Geiecke and Jaravel (2025). Their implementation appends each message after a question so that, even when prompted with a shorter input, the AI can still recall the entire interview context. While effective for interviews between two persons/agents, this approach is not feasible for a multi-agent focus group.
Some alternatives exist, such as in Zhang et al. (2024), where several AI participants speak through the *user* role, with prompts instructing the model to *forge* its previous identity and adopt a new one. However, I follow a more structured approach: instead of appending all prior messages, I provide the updated conversation history directly. In other words, every response of an AI agent can be seen as the first LLM response in a conversation you just have started. This ensures clarity and role separation, though it comes at the cost of a larger input token count.

4.2.4 Additional Contributions: Options & Functions

In addition to features already available in the original repository of Geiecke and Jaravel (2025) (e.g., the optional login window), I implemented several new options and functions:

1. **Human Activation:**
   By setting `HUMAN_ACTIVATION = True` and sharing the focus group link, an external participant can join the discussion via a text input field alongside the AI agents.

2. **Debugging:**
   When developing or testing new prompts, it is often useful to inspect how the invisible moderator selects the next speaker and whether the prompts are updated correctly. If `DEBUGGING = True`, the app displays the AI agents' prompts and JSON outputs, making it easier to analyze issues while running the focus group.

3. **Prompt Updating & Chat History:**
   I implemented functions such as `inv_filled` that update the prompts of the AI agents (e.g., the invisible moderator). In addition, I defined functions like `chat_history` that retrieve the current conversation history from the Streamlit `session_state` variables. This history is then embedded into the prompt-updating function, ensuring that the agents respond with full awareness of the ongoing discussion.

4. **Speaking Time:**
   I implemented a function (`minutes_spoken`) that helps to track tracks both, the speaking time of each participant and the cumulative elapsed time. The invisible moderator relies on this information to determine the

next speaker based on several criteria: ensuring equal participation across agents, adhering to predefined time slots or intervals for each discussion segment, and transitioning smoothly at the appropriate moments (see the instructions, i.e., prompt, for the invisible moderator in Section 4.1.1).

Important: Following Zhang et al. (2024), I apply a transformation where 100 written words correspond to 1 minute of speaking time.

4.3 Implementation - Local & Global Setup

The implementation works analogously to Section 3.3. In principle, you only have to replace `code/interview.py` with `code/focusgroup.py` in the instructions there. The interview transcripts, time stamps, and backups are saved in the folder `code/data_focusgroup`. Further, the file `code/config.py` is integrated in `code/focusgroup.py`, and thus, no longer needed for the AI focus group.

4.4 Limitations

I encountered several limitations when writing the codebase for the focus group. These are the most relevant:

- **Token rate and memory constraints:**
  A key limitation lies in token rate and token memory constraints, which restrict how much context can be processed in real time and slow down multi-agent interactions. These limits make it challenging to provide rich prompts or maintain detailed conversation histories without hitting API boundaries. In particular, if you are relatively new to OpenAI and only on the first tier, the token rate limit for `gpt-4o` and `gpt-5` models is about 30,000 tokens per minute (TPM), which is low enough that you can only make 1–2 API calls per minute. On the third tier (800,000 TPM), this issue is less critical. To accommodate lower tiers, I implemented a short 6-second pause between loop iterations (`time.sleep(6)`), allowing up to 20 prompts per minute. This setup works well for the default focus group (60 minutes / ~6,000 words). You can check your current tier, limits, and upgrade options here.
- **Model differences:**
  `gpt-5` tends to follow instructions more precisely than `gpt-4`, but this can also lead to subtle behavioral differences between agents. Balancing role separation, reasoning complexity, and natural conversation flow across different models remains a challenge.
- **Uniform prompts:**
  Further, using the same base prompt for all participants ensures replicability but results in repetitive phrasing. While acceptable in purely AI-driven discussions, it feels unnatural in mixed groups with human participants. To counter this, I prompt agents not to copy each other's sentence structure (see prompt in Section 4.1.3). The tradeoff lies between very specific prompts and more superficial ones. In particular the question is how many characteristics should be given to every AI agent simulating a realistic person.
- **Timing and transitions:**
  Moreover, leaving topic shifts to the LLM often produces unpredictable results, with some discussions running much shorter or longer than intended. For instance, prompting the LLM to transistion to the next part of the topic guide, when it has the feeling that the goals of the current topic being covered and be able to end the focus group without hurrying exactly on time are fulfilled equally, might cause a huge variation in the decisions of the invisible moderator. To prevent this, I introduced time windows that guide the invisible moderator in triggering transitions (see prompt in Section 4.1.1). The tradeoff is between giving the LLM more flexibility and ensuring all planned topics are covered within the available time.

- **Natural Human Integration:**
  Lastly, a major challenge is balancing LLM contributions so that the focus group resembles a discussion among humans if the human participant is activated. The key limitation is the absence of non-verbal communication, which plays a central role in human interaction. In real groups, participants rely on facial expressions, gestures, and eye contact to gauge agreement, disagreement, or readiness to speak. These cues are missing in the hybrid setting of AI and human participants, which can make the conversation feel less natural. Since this deliverable mainly focuses on a pure AI focus group design such aspects have been ignored so far. However, Section 4.5 discusses potential extensions to address this limitation.

4.5 Extensions

Since this deliverable represents only a minimal working example of an AI or hybrid focus group, there is considerable room for future improvements and extensions. Below are several directions I considered while coding the prototype:

- **Natural Human Integration:**
  To make the experience more realistic for human participants, several enhancements could be explored:
  - Allow both AI agents and human participants to indicate their willingness to speak (e.g., via a "raise hand" button). This could reduce frustration when someone wants to contribute but is not selected.
  - Introduce voice input for human participants, enabling them to speak their contributions directly. For instance, Chopra and Haaland (2024) provide a voice interface for AI-led interviews. Likewise, AI responses could be read aloud to human participants.
  - Enable multiple human participants by allowing more than one login and linking each login to a separate participant role. A simple condition can then decide whose input is accepted at a given turn.
  - Make the closing stage more natural. In real groups, participants often shake their heads or use gestures to indicate they have nothing more to add. In a chat interface, this absence feels unnatural. One solution would be to prompt all AI participants to respond with at least a brief acknowledgment (e.g., "No further comments."), while ensuring the human participant is always invited to provide feedback.
  - Use AI-generated video avatars for AI participants, creating the impression of a virtual face-to-face discussion.
- **Classification AI Agent:**
  An additional invisible agent could be introduced to summarize transcripts, classify statements, or highlight interesting research questions - similar to the summary agent within the multi-agent setup in Chopra and Haaland (2024).
- **Temperature and Model Variations:**
  Varying the temperature setting across AI participants could simulate different personalities, making some more subjective, skeptical, or prone to conspiracy thinking, while others remain more neutral. This would allow the study of polarization dynamics in focus groups. It would also be useful to test which LLMs perform best in the invisible moderator role, since it requires strong analytical reasoning, and whether different temperature settings improve reliability.

4.6 Evaluation

This section provides a **preliminary, suggestive evaluation** of whether the pure AI focus group achieves its intended purpose: to generate insights comparable to those obtained from human focus groups. In particular, we are interested in the topics that both human and AI-led discussions can reveal.

To conduct this comparison, we need a *"gold standard"* against which the AI-generated topics can be measured. Here, we use the default focus group discussion described earlier, which closely imitates well-documented human focus groups reported by Morton et al. (2024). They provide three full transcripts (see `Morton et al - Material/Transcripts/Employee`) from 60-minute sessions with seven employees and a moderator discussing sedentary behavior. We compare these with two AI-generated transcripts recorded using `gpt-4o` and `gpt-5` (see `Geiecke_Jaravel_2025/data_focusgroup/transcripts`). The default prompts for the AI simulations were partially enriched with participant information from Morton et al. (2024) to achieve as close a comparison as possible.

For transcript comparison, topic modeling is a natural choice. Methods such as LDA or TF-IDF combined with clustering are commonly used. In our analysis, we employ BERTopic, developed by Grootendorst (2022). BERTopic *"leverages transformers and c-TF-IDF to create dense clusters, allowing for easily interpretable topics while preserving important words in topic descriptions."* This method combines several concepts covered in this course, including embeddings (BERT), dimensionality reduction (UMAP), clustering (KMeans), and weighting schemes (c-TF-IDF). BERTopic is a well-established algorithm, previously applied by Chopra and Haaland (2024) to extract subjective mental models from qualitative interviews. For general guidance on thematic analyses with or without AI, see Braun and Clarke (2021) and Olawade et al. (2025).

Before starting with the analysis, it is important to note that analyzing only five transcripts is insufficient for traditional topic modeling, which usually requires hundreds or thousands of documents. To mitigate this, we divide each transcript into smaller chunks, allowing BERTopic to assign a **topic representation** to each chunk. An excerpt of the results is shown in the **table below**. While it is theoretically possible to cluster and name these topics, doing so here is not meaningful. Some chunks may contain deep insights, while others may consist solely of moderator questions. Additionally, treating chunks as independent documents ignores the structured interview guidelines, which help prevent repetitive topic representations. Therefore, we do not further classify the topic representations in this evaluation.

```
# --- Step 1: Import required libraries ---
from bertopic import BERTopic
import umap
import hdbscan
import pandas as pd
import os
from IPython.display import display
import plotly.io as pio
import plotly.express as px
import numpy as np

# --- Step 1a: Configure Plotly for inline rendering ---
pio.renderers.default = "notebook_connected"

# --- Step 2: Specify transcript files ---
transcript_files = [
    r"Morton et al - Material/Transcripts/Employee/FG_employee_1_pseudo.txt",
    r"Morton et al - Material/Transcripts/Employee/FG_employee_2_pseudo.txt",
```

```
    r"Morton et al - Material/Transcripts/Employee/FG_employee_3_pseudo.txt",

    ↪ r"Geiecke_Jaravel_2025/data_focusgroup/transcripts/testaccount_gpt-5-chat-latest_NO_HUMAN_2025

    ↪ r"Geiecke_Jaravel_2025/data_focusgroup/transcripts/testaccount_gpt-4o-2024-05-13_NO_HUMAN_2025
]

# --- Step 3: Load transcripts into memory ---
transcripts = []
file_names = []

for file_path in transcript_files:
    if os.path.exists(file_path):
        with open(file_path, "r", encoding="utf-8") as f:
            transcripts.append(f.read())
            file_names.append(os.path.basename(file_path))
    else:
        print(f"Warning: File not found: {file_path}")

#print(f"Loaded {len(transcripts)} transcripts.")

# --- Step 4: Chunk transcripts into smaller segments ---
def chunk_text(text, chunk_size=80):
    """Split a text into chunks of approximately `chunk_size` words."""
    words = text.split()
    return [" ".join(words[i:i+chunk_size]) for i in range(0, len(words), chunk_size)]

chunks = []
doc_map = []
for doc_id, transcript in enumerate(transcripts):
    for chunk in chunk_text(transcript, chunk_size=80):
        chunks.append(chunk)
        doc_map.append(doc_id)

#print(f"Total chunks created: {len(chunks)}")

# --- Step 5: Configure BERTopic model ---
umap_model = umap.UMAP(
    n_neighbors=2,
    n_components=2,
    min_dist=0.1,
    metric="cosine",
    random_state=42
)

hdbscan_model = hdbscan.HDBSCAN(
    min_cluster_size=2,
    min_samples=1,
    metric='euclidean',
    cluster_selection_method='eom'
```

```
)

topic_model = BERTopic(
    umap_model=umap_model,
    hdbscan_model=hdbscan_model,
    min_topic_size=2,
    verbose=True
)

# --- Step 6: Fit BERTopic to the chunks ---
topics, probs = topic_model.fit_transform(chunks)

# Create dataframe mapping chunks, docs, and topics
df = pd.DataFrame({
    "chunk": chunks,
    "doc_id": doc_map,
    "topic": topics
})

# --- Step 7: Inspect discovered topics ---
topic_info = topic_model.get_topic_info()
display(topic_info)
```

```
C:\Users\torben\Documents\Bonn\Uni\Courses\Data Sciene, Machine Learning, AI\Deliverable\deliverable_s

IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/st

2025-08-24 07:32:48,925 - BERTopic - Embedding - Transforming documents to embeddings.
Batches:   0%|          | 0/15 [00:00<?, ?it/s]Batches:   7%|         | 1/15 [00:00<00:08,  1.58it/s]
2025-08-24 07:33:00,501 - BERTopic - Embedding - Completed
2025-08-24 07:33:00,503 - BERTopic - Dimensionality - Fitting the dimensionality reduction algorithm
2025-08-24 07:33:13,408 - BERTopic - Dimensionality - Completed
2025-08-24 07:33:13,409 - BERTopic - Cluster - Start clustering the reduced embeddings
2025-08-24 07:33:13,442 - BERTopic - Cluster - Completed
2025-08-24 07:33:13,448 - BERTopic - Representation - Fine-tuning topics using representation models.
2025-08-24 07:33:14,142 - BERTopic - Representation - Completed
```

	Topic	Count	Name	Representation
0	-1	12	-1_tired_could_build_time	[tired, could, build, time, youre, now, your,
1	0	7	0_lunch_here_just_easy	[lunch, here, just, easy, buy, eat, healthily,.
2	1	5	1_dog_computer_nice_chairs	[dog, computer, nice, chairs, horizontally, t
3	2	5	2_cancer_surprised_james_thank	[cancer, surprised, james, thank, seeing, co
4	3	5	3_monotony_improvements_discomfort_maintaining	[monotony, improvements, discomfort, mai
...	...	...	...	...
153	152	2	152_additionally_desks_standing_lets	[additionally, desks, standing, lets, traditio.
154	153	2	153_refreshed_distinct_areas_physically	[refreshed, distinct, areas, physically, relax.
155	154	2	154_virtual_step_challenge_fantastic	[virtual, step, challenge, fantastic, social, ...

	Topic	Count	Name	Representation
156	155	2	155_shared_virtual_step_connected	[shared, virtual, step, connected, remotely,
157	156	2	156_discussed_share_solutions_transforming	[discussed, share, solutions, transforming, l

```
# --- Step 8: Standard BERTopic visualizations ---
fig_heatmap = topic_model.visualize_heatmap(top_n_topics=20)

# Keep only the main heatmap (optional)
fig_heatmap.update_layout(showlegend=True)

# Return the figure for Quarto rendering
fig_heatmap.write_image("fig_heatmap.png", width=800, height=600)

# --- Step 9: Chunk-level scatterplot with D1/D2 axes ---

# Step 9.1: Compute embeddings for each chunk
embeddings = topic_model.embedding_model.embed(chunks)

# Step 9.2: Reduce embeddings to 2D
umap_2d = umap.UMAP(
    n_neighbors=5,
    n_components=2,
    min_dist=0.1,
    metric="cosine",
    random_state=42
).fit_transform(embeddings)

# Step 9.3: Add D1/D2 and Transcript labels
df["D1"] = umap_2d[:, 0]
df["D2"] = umap_2d[:, 1]
df["Transcript"] = df["doc_id"].map(lambda i: file_names[i])

# Step 9.4: Compute topic frequencies for marker size
topic_counts = df.loc[df["topic"] != -1, "topic"].value_counts().to_dict()
df["Frequency"] = df["topic"].map(lambda t: topic_counts.get(t, 1))

# Step 9.5: Create scatter plot
fig = px.scatter(
    df,
    x="D1",
    y="D2",
    color="topic",         # color = Topic
    symbol="Transcript",   # symbol = Transcript
    size="Frequency",      # size = Topic frequency
    size_max=25,
    hover_data={
        "chunk": True,
        "topic": True,
        "Transcript": True,
```

```
        "Frequency": True,
        "D1": False,
        "D2": False
    },
    title=""
)

# Step 9.6: Adjust legend
fig.update_layout(
    legend=dict(
        orientation="h",
        yanchor="top",
        y=-0.25,
        xanchor="center",
        x=0.5,
        font=dict(size=10)
    ),
    legend_title_text="Transcript"
)

# Clarify size meaning
fig.add_annotation(
    x=0.5,
    y=-0.3,
    xref="paper",
    yref="paper",
    text="",
    showarrow=False,
    font=dict(size=10),
    xanchor="center"
)

fig.write_image("fig_chunks.png", width=800, height=600)
```

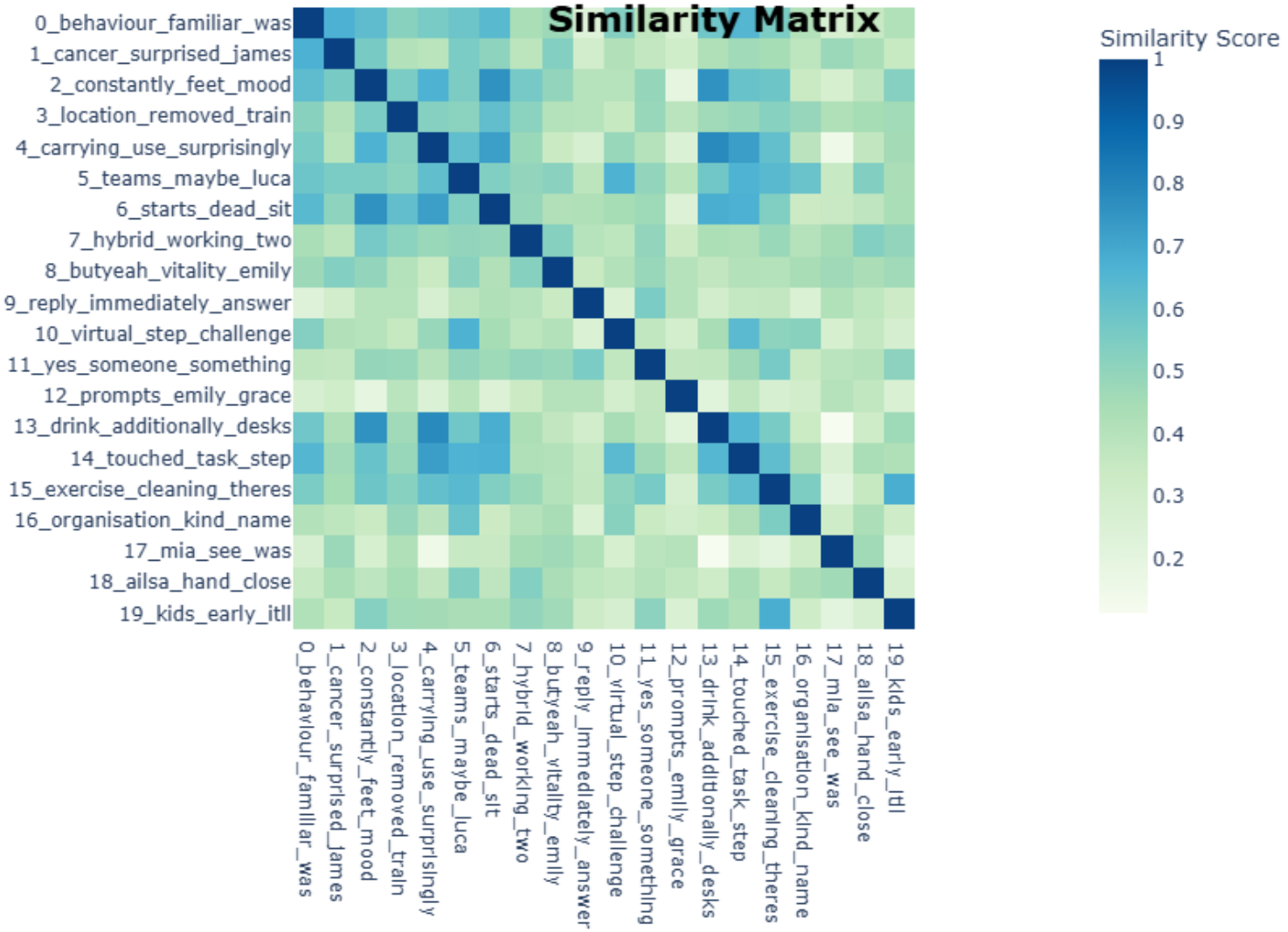

Fig 2: Heatmap - Semantic Cosine Similarity Across 20 Top Topics

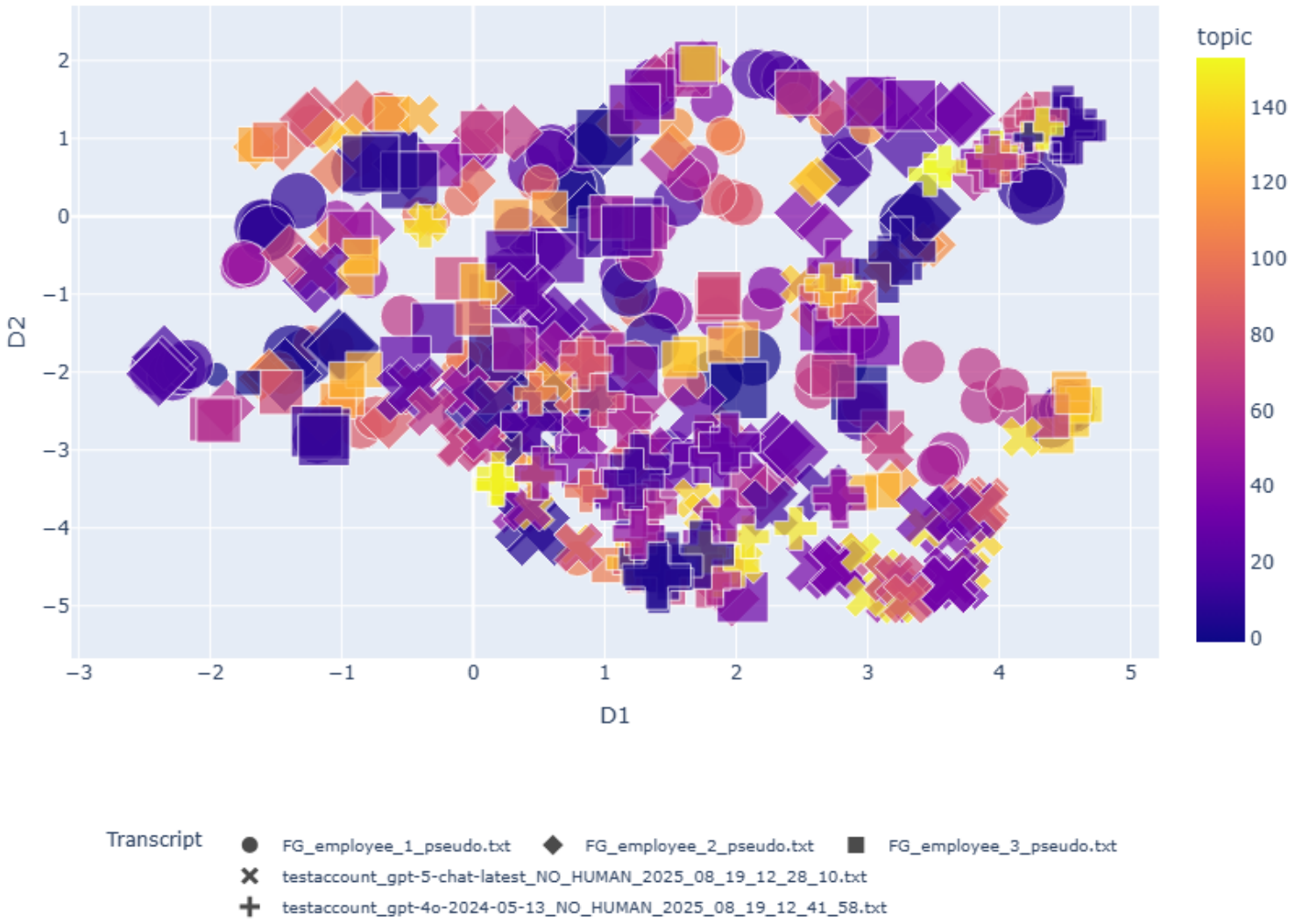

Fig 3: Transcripts - Topics and Topic Frequency

The **heatmap** visualizes the **top 20 topic representations** and their **cosine similarity** to each other. This provides a preliminary overview, demonstrating that the most frequent topics cover distinct themes.

The **scatterplot** reduces the topic dimensionality to two dimensions, illustrating how chunk topics and transcript topics are distributed in a 2D vector space. Note that the HTML format, allows the interactive inspection of individual data (topic) points. Overall, no transcript appears clearly separable from the others, suggesting that the AI focus group captures topics that humans also perceive as important when it comes to sitting behavior. Further, both AI transcripts tend to discuss similar topics, which is expected since only the LLM model varied while prompts remained constant. Interestingly, AI-generated topic representations are more densely clustered in the bottom-right region of the plot. This could indicate an additional perspective introduced by the AI, limitations in prompt replication to simulate exact human disscusion partipants, or improvable LLM temperature settings. Several other factors - like models being trained on things that have happened after the human focus group was conducted - may also contribute to this observation.

Ultimately, we cannot definitively evaluate whether these patterns are positive or negative due to the small dataset. Furthermore, the two AI focus groups were not moderated identically, and prompts were not fully optimized to replicate human interactions. In similar research, Zhang et al. (2024) found that AI agents and humans contribute very

similar, though not identical, content when moderated by AI. Their descriptive analyses produced results comparable to those observed here.

5 Conclusion

In summary, while further evaluation is needed, this **flexible, end-to-end implementable** AI focus group - and by extension, the hybrid format - appears capable of generating human-like outcomes, making this deliverable at least partially successful. A key improvement is the enhanced transition logic, which addresses a limitation observed in previous AI-moderated focus groups: the participant selection mechanism. Earlier implementations often relied on bidding competitions, either imagined by the moderator or triggered by participant signals (e.g., hand-raising), as discussed in Zhang et al. (2024). In contrast, the **invisible moderator** role introduced here provides a more robust and natural approach to determining speaking order.

The development of this format is particularly relevant for research exploring group dynamics under varying conditions. For example, it enables the study of human reactions to exogenously induced polarized, radical, or conspiracy-oriented positions, dependent on whether these are homogeneous or heterogeneous within a discussion group. Moreover, the hybrid focus group format is advantageous when it is difficult to gather all desired social groups simultaneously or when specific representatives are unavailable.

Finally, it is important to acknowledge the current limitations of the design (see Section 4.4). Future extensions could include video *"deepfakes"* of AI participants to enhance the visual and emotional realism of the conversation. Such improvements would allow both the invisible moderator and human participants to exercise more natural autonomy in deciding when to contribute, further bridging the gap between AI simulations and real-world focus group dynamics.

References

Bergman, Peter, Raj Chetty, Stefanie DeLuca, Nathaniel Hendren, Lawrence F. Katz, and Christopher Palmer. 2024. "Creating Moves to Opportunity: Experimental evidence on barriers to Neighborhood Choice." *American Economic Review* 114 (5): 1281–1337.

Billups, Felice D. 2021. "Focus Group Moderator Guides." *Focus Group Moderator Guides*, 97–132.

Braun, Virginia, and Victoria Clarke. 2021. "Thematic Analysis: A Practical Guide."

Bursztyn, Leonardo, Ingar K Haaland, Nicolas Röver, and Christopher Roth. 2025. "The Social Desirability Atlas." National Bureau of Economic Research.

Chopra, Felix, and Ingar Haaland. 2024. "Conducting Interview with AI." *Working Paper*.

Geiecke, Friedrich, and Xavier Jaravel. 2025. "Conversations at Scale: Robust AI-led Interviews." *Working Paper*.

Grootendorst, Maarten. 2022. "BERTopic: Neural Topic Modeling with a Class-Based TF-IDF Procedure." *arXiv Preprint arXiv:2203.05794*.

Haaland, Ingar, Chris Roth, Stefanie Stantcheva, and Johannes Wohlfahrt. forthcoming. "Understanding Economic Behavior Using Open-ended Survey Data." *Journal of Economic Literature*, forthcoming.

Kahan, James P. 2001. "Focus Groups as a Tool for Policy Analysis." *Analyses of Social Issues and Public Policy* 1 (1): 129–46.

Morton, Sarah, Claire Fitzsimons, Divya Sivaramakrishnan, Ruth Jepson, and Ailsa Niven. 2024. "'Are We Working (Too) Comfortably?': A Focus Group Study to Understand Sedentary Behaviour When Working at Home and Identify Intervention Strategies." *BMC Public Health* 24 (1): 1516.

Olawade, David B, Deborah Omeni, Manisha Nitin Gore, and Manizha Hadi. 2025. "Enhancing Qualitative Research Through Virtual Focus Groups and Artificial Intelligence: A Review." *International Journal of Medical Informatics*, 106004.

Rabiee, Fatemeh. 2004. "Focus-Group Interview and Data Analysis." *Proceedings of the Nutrition Society* 63 (4): 655–60.

Zhang, Taiyu, Xuesong Zhang, Robbe Cools, and Adalberto Simeone. 2024. "Focus Agent: Llm-Powered Virtual Focus Group." In *Proceedings of the 24th ACM International Conference on Intelligent Virtual Agents*, 1–10.