SUPPLEMENTAL MATERIAL

ROGER: Visualizing Voice Records to Enhance Team Communication Trainings for High-Stress Situations

ANONYMOUS AUTHOR(S)

Contents

Communication Failure	**Description**
Incomplete information	Forgets to mention the floor number.
Failure to follow standardized protocol	Deviates from established communication procedures, causing confusion.
Failure to use standardized terminology	Colleagues use different terms for the same action.
Disadvantageous words	A colleague mentions a weapon malfunction in front of third parties.
Unclear or overly complex messages	Instructions are overly detailed or ambiguous, making them hard to follow.
Excessive use of filler words	Overuse of “um” and “ah” disrupts clarity.
Vague language	Instructions such as ”take care of that” without specifying the task.
Information overload	Too much information is given at once, making it difficult to process.
Emotional tone mismatch	A casual tone is used during a high-risk situation, reducing urgency.
Absence of a conversation leader	Multiple people attempt to give instructions, leading to uncertainty.
Interruptions between colleagues	Two people talk over each other, causing important information to be missed.
Lack of confirmation	No acknowledgment of a critical command leads to uncertainty about whether it was understood.
Failure to speak	A colleague remains silent during a critical moment when input is needed.
Poor timing of message	A question is asked after a decision has already been made.
Long response times	A delayed reply disrupts the flow of action.
Turning away from potential danger	An officer instinctively turns toward a colleague while speaking, putting themselves in a dangerous position.

Table 1.: Exemplary communication failures that were repeatedly noted during interviews with police trainers, who aim to analyze such issues using *ROGER*.

SUP-1. Example Communication Failures

During our requirements analysis, we found that interviewees provided more valuable insights when discussing communication failures and recounting breakdowns rather than specifying interface requirements. Table 1 presents a selection of articulated failures that informed our task abstraction.

SUP-2. Implementation

ROGER uses a client-server architecture which separates the front-end interactive visualizations from back-end data processing. The front end is implemented in JavaScript using Svelte. D3.js is used for custom visualizations and interaction techniques. The back-end is developed in Python using Flask[1], which provides a RESTful API for handling requests and processing data. The integration of LLM-based motif filtering is facilitated by LangChain[2], a Python framework that connects the back-end to the LLM API.

SUP-3. Prompt Template

The prompt is the request sent from the *ROGER* backend to the LLM API. The transcript of messages and the custom motif are dynamically inserted. The LLM processes the request and returns a structured response containing a list of message IDs that match the given motif. These message IDs are then used to filter relevant messages and adjust the visualizations.

Below is our prompt template:

Prompt

```
You are a professional communication training assistant. Analyze the
transcript below to identify messages that match the given user query:
"{MOTIF}". Each row in the transcript is one message and contains the
following columns: id (message ID), spk (speaker ID), message (spoken
content). Return only a list of message IDs that match the query.
Ensure all returned IDs are present in the original transcript.

{TRANSCRIPT}
```

[1]https://flask.palletsprojects.com

[2]https://python.langchain.com