TalkTagger – Can You Guess Who Said That?

A Chat-Based NLP Game

Beray Nil Atabey
AI Lab: CV and NLP
Applied Computer Science and Artificial Intelligence
Sapienza University of Rome

July 2025

Abstract

This project explores the intersection of natural language processing and social interaction by developing a chat-based guessing game that challenges players to identify speakers and styles from both real and AI-generated messages. Real-world chat data (such as WhatsApp and Discord) and a variety of natural language processing (NLP) techniques are used – including sentiment analysis, personality modeling, custom preprocessing, and neural text generation – to simulate and analyze user communication patterns. This approach is driven by the goal of comprehending how personality and style appear in digital conversations. The game leverages existing NLP techniques to capture distinct digital personalities while also offering insights into the subtleties of language, style, and attribution.

All code is available on https://github.com/NilAtabey/TalkTagger.

1 Introduction

Playing party games like Jackbox with friends and looking for a more customized, socially relevant experience served as an inspiration for TalkTagger. Even though current games are entertaining and engaging, they frequently don't directly relate to the personalities and interactions of players in real life. The ability to export chat histories from apps like WhatsApp has become widely available, creating a special opportunity to use real-world digital conversations as the basis for an entirely new type of game. Analyzing chat data for style and personality not only deepens our understanding of social dynamics and digital identity, but also creates a playful context for exploring how we express ourselves in everyday language.

TalkTagger is a multiplayer online game that is similar to Kahoot or Jackbox. This is how the gameplay plays out:

- The game backend is run on a single device, usually a large screen, which also shows questions to all players.
- Users have the option to transfer their exported chat histories to the host device, such as from Discord or WhatsApp. These conversations are processed by the system and gets them ready for gameplay.
- To enable a smooth and easy multiplayer experience, players use their mobile devices to join the game session by entering a special lobby code.
- Players must guess the original author after being shown a series of chat messages, some of which are created by AI and some are real.
- Fun superlatives are shown at the end of the game based on player performance and message analysis, and points are given for accurate attributions.
- Real-time player interaction and simple access from mobile devices are made possible by the system's host-client model.

2 Dataset and Preprocessing

Export chat logs from a preferred messaging app, like WhatsApp or Discord, and serve as TalkTagger's main data sources. Because these two platforms provide exporting solutions, only preprocessing steps are implemented at this time. Users upload these exports through the host interface of the game, and they are saved in the `convos_before/` directory. `convos_after/` contains the cleaned and parsed versions that have been processed.

The project does not currently use any pseudonymization or anonymization techniques for user identities. During preprocessing and gameplay, speaker identifiers and usernames from the original chat exports are retained. This design decision was made to preserve the game's social and personal elements, but it's crucial to remember that using sensitive data could raise privacy issues. To better protect user privacy, future iterations might include hashing or pseudonymization.

Raw chat exports are often noisy and contain extraneous information such as emojis, timestamps, and system messages. The preprocessing pipeline, implemented in scripts such as `chat_preprocessor.py`, `wp_parser.py`, and `dc_parser.py`, performs several cleaning steps:

- Strips out emojis and other non-textual symbols to focus on linguistic content.
- Removes timestamps and metadata that are not relevant to the game or analysis.
- Standardizes text by handling casing, whitespace, and other formatting inconsistencies.

After cleaning, the chat data undergoes further processing:

- Messages are split into sentences or tokens as needed for downstream NLP tasks.
- Each message is tagged with its corresponding speaker, preserving the conversational structure and enabling accurate attribution during gameplay.

The preprocessing pipeline is designed to manage the informal, error-prone nature of real-world chat data, which often includes typos, abbreviations, and unconventional grammar.

3 Methodology

3.1 Personality Modeling

A user's style is captured through a combination of statistical features and linguistic patterns extracted from their real chat messages. Each user profile includes metrics such as most common words, signature words and phrases, average message length, capitalization and punctuation habits, and emoji usage. These features are engineered to reflect both the content and stylistic tendencies of each participant.

Features:

- Topics and n-grams (frequent words/phrases)
- Stylistic markers (e.g., sentence capitalization, punctuation ratios, emoji frequency)
- Message length statistics

```python
# Inside _build_user_profiles
signature_words = sorted(signature_scores.items(), key=lambda
   x: x[1], reverse=True)[:10]
profiles[user] = {
    "most_common_words": [{"word": w, "count": c} for w, c in
        vocab.most_common(10)],
    "signature_words": [{"word": w, "score": s} for w, s in
        signature_words],
    "avg_message_length_words": round(avg_msg_length, 2),
    "capitalized_sentence_start_ratio":
        round(capitalized_starts / msg_count, 3),
    "emoji_count": emoji_count,
    # ... more features ...
}
```

Listing 1: Extracting stylistic features from chat logs

Custom feature engineering for style and personality, using statistical analysis of chat data. User profiles are built from real message data, providing a robust statistical profile for each participant. These profiles are used both for message selection and as conditioning context for text generation.

3.2 Sentiment Analysis

```python
from transformers import pipeline
sentiment = pipeline("sentiment-analysis",
   model="cardiffnlp/twitter-roberta-base-sentiment-latest")
result = sentiment("I love this game!")
```

Listing 2: Sentiment Analysis Placeholder

For sentiment analysis, the pipeline is designed to support both classic models (like VADER) and modern transformer-based models. In particular, the Hugging Face Twitter-roBERTa-base-sentiment model was used, which is specifically trained on social media data and was the most well-suited sentiment model available for Gen-Z text messages. This model classifies each message as negative, neutral, or positive, providing a nuanced view of the emotional tone in conversations. Right now, sentiment scores are not actively used in the main game pipeline (but can be generated upon demand, and in future iterations, can be integrated at the end screen along with the "Superlatives"). The pipeline is designed to accommodate sentiment scoring, with placeholders for model integration.

3.3 Text Generation

Synthetic messages are generated using the Mistral API (a transformer-based large language model) in a zero-shot setting. No local training or fine-tuning is performed; instead, prompts are engineered to condition the model on user-specific style features.

```
# Inside generate_messages_for_user
prompt = self.create_balanced_user_prompt(user, profile,
    sample_messages, count, topics)
response = self.call_mistral_api(prompt)
```

Listing 3: Text Generation: Prompting the Model

Initial attempts had a custom fine-tuned model, however, it slowed gameplay significantly. In the 2nd iteration currently the model is not fine-tuned on user data. Instead, user profiles (statistical features and sample messages) are used to craft prompts that guide the model to generate text in the style of a specific participant. This is done so that the game pipeline is faster, so that players don't have to wait too long before playing the game.

In order to ensure that generated messages closely resemble the communication style of the real user, prompts include signature words, phrases, and stylistic statistics from the user's profile.

The generation process is infused with user style through prompt engineering. The system generates high-quality, context-aware text by utilizing a cutting-edge transformer model (Mistral) through an API. To guarantee style fidelity, generated messages are graded for uniqueness and BERT similarity.

3.4 Similarity Metrics

Metrics used are distinctiveness score and BERT similarity. For the distinctiveness score: a custom, statistical metric that quantifies how closely a message matches a user's unique style, based on engineered features. For the BERT similarity: cosine similarity between BERT (sentence-transformer) embeddings of a message and the average embedding of a user's real messages, expressed as a percentage.

```
def compute_similarity(text1, text2):
    emb1 = get_embedding(text1)
    emb2 = get_embedding(text2)
    return cosine_similarity([emb1], [emb2])[0][0]
```

Listing 4: Similarity Metrics: BERT Similarity

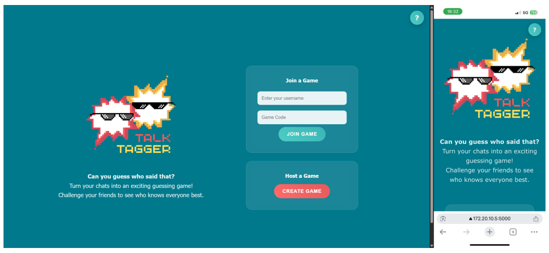
```
# Inside score_message_distinctiveness
```

```
signature_matches = sum(1 for word in message_words if word
    in signature_words)
score += signature_matches * 3.0
if phrase in message.lower():
    score += 5.0  # signature phrase bonus
```

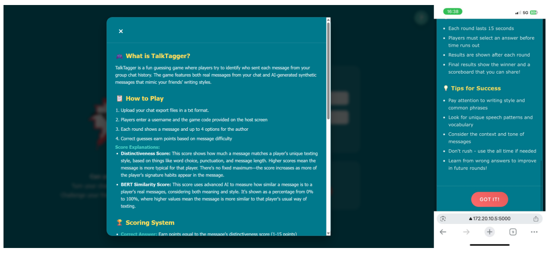
Listing 5: Similarity Metrics: Distinctiveness Score

Both metrics are used to evaluate and display how well real and synthetic messages match a user's style. They are shown to players after each round and can be used for superlative awards or further analysis. Embedding extraction and comparison using the sentence-transformers library. Quantitative evaluation of style similarity, combining interpretable statistical features with modern deep learning embeddings. Both real and synthetic messages are evaluated using these metrics, providing a comprehensive view of style matching.

4 Game Logic and UI


TALK TAGGER
Can you guess who said that?
Turn your chats into an exciting guessing game!
Challenge your friends to see who knows everyone best.
Join a Game
Enter your username
Game Code
JOIN GAME
Host a Game
CREATE GAME
172.20.10.5:5000

(a) Main Page



🎮 What is TalkTagger?

TalkTagger is a fun guessing game where players try to identify who sent each message from your group chat history. The game features both real messages from your chat and AI-generated synthetic messages that mimic your friends' writing styles.

📋 How to Play

1. Upload your chat export files in a txt format.
2. Players enter a username and the game code provided on the host screen
3. Each round shows a message and up to 4 options for the author
4. Correct guesses earn points based on message difficulty

Score Explanations:

- **Distinctiveness Score:** This score shows how much a message matches a player's unique texting style, based on things like word choice, punctuation, and message length. Higher scores mean the message is more typical for that player. There's no fixed maximum—the score increases as more of the player's signature habits appear in the message.
- **BERT Similarity Score:** This score uses advanced AI to measure how similar a message is to a player's real messages, considering both meaning and style. It's shown as a percentage from 0% to 100%, where higher values mean the message is more similar to that player's usual way of texting.

🏆 Scoring System

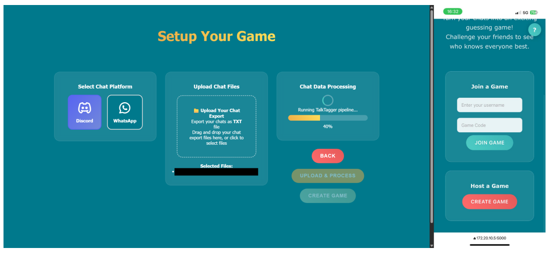
- Correct Answer: Earn points equal to the message's distinctiveness score (1-15 points)

- Each round lasts 15 seconds
- Players must select an answer before time runs out
- Results are shown after each round
- Final results show the winner and a scoreboard that you can share!

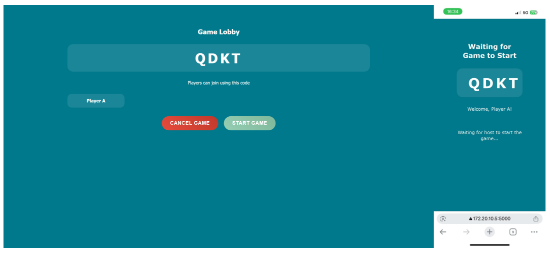
💡 Tips for Success

- Pay attention to writing style and common phrases
- Look for unique speech patterns and vocabulary
- Consider the context and tone of messages
- Don't rush - use the all time if needed
- Learn from wrong answers to improve in future rounds!

(b) How To Play Instructions



(c) Game Setup


Game Lobby
QDKT
Players can join using this code
Player A
CANCEL GAME
START GAME
Waiting for Game to Start
QDKT
Welcome, Player A!
Waiting for host to start the game...
172.20.10.5:5000

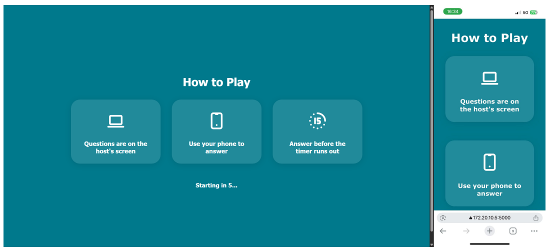
(d) Game Lobby

Figure 1: Screenshots of TalkTagger Pre-Game

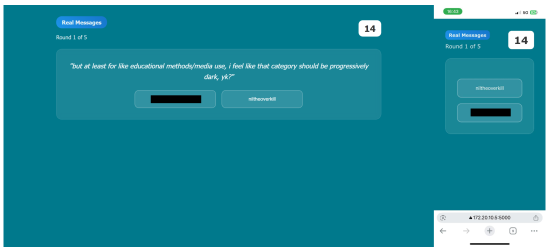
The game is structured in two main phases: real and generated message rounds. In the first phase, players are presented with actual messages from their uploaded chat history and must guess which participant authored each message. In the second phase, the game introduces AI-generated (synthetic) messages crafted to mimic the style of real users. Players are again challenged to attribute each message to the correct person, with the added twist of distinguishing between authentic and AI-generated content.

Players can upload more than one chat log from the same platform (e.g. multiple WhatsApp exports), allowing broader sampling of user messages and enabling more dynamic gameplay.

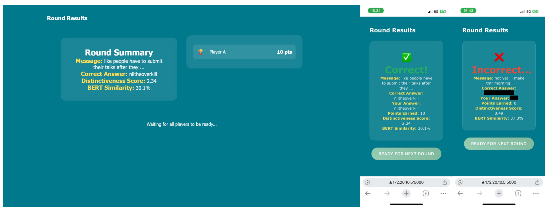
Points are awarded based on the difficulty of each question, primarily using the distinctiveness score of the message. More distinctive messages (those that closely match a user's unique style) are worth more points. There is also a bonus point system for faster correct answers, no points are deducted for incorrect answers. Both distinctiveness and BERT similarity scores are displayed after each round.


How to Play
Questions are on the host's screen
Use your phone to answer
Answer before the timer runs out
Starting in 5...

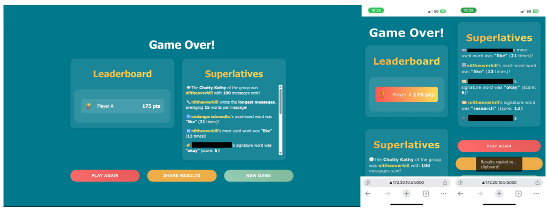
(a) Warnings Before the Game Starts



(b) Questions Page



(c) Results Page



(d) Game Over Page

Figure 2: Screenshots of TalkTagger Gameplay and Post-Game

At the end of the game, fun superlative awards are generated based on player performance and message analysis. These awards use both statistical and embedding-based metrics to highlight unique player traits and add a playful, competitive element to the experience.

The user interface is fully web-based, built with Flask (backend), HTML, CSS, and JavaScript (frontend). The main files include `frontend/app.py` (Flask server and backend logic), `game.html` (game interface), `game.js` (client-side logic and real-time updates), and `style.css` (styling and responsive design). The UI is designed to be accessible from both desktop and mobile devices, allowing players to join games easily via a browser.

Real-time game state and player actions are managed using WebSockets, enabling instant updates and smooth multiplayer gameplay. The backend handles game progression, scoring, and synchronization of player actions, ensuring a seamless experience even with multiple concurrent users. The interface emphasizes clarity, responsiveness, and ease of use, with clear feedback for player actions, visually distinct phases, and engaging animations for transitions and results. The design supports both large-screen (host) and mobile (player) experiences, making the game suitable for group play in various settings.

5 Experiments and Results

To evaluate the effectiveness of the system, I measured how accurately players could attribute both real and AI-generated messages to the correct speaker. Human performance was compared to the model's own style-matching metrics (distinctiveness and BERT similarity). Results showed that players were generally more accurate with real messages, while synthetic messages, especially those with high BERT similarity, were more challenging and sometimes successfully fooled participants. Both categories had a similar correct answer rate, with real messages being guessed correctly 72% of the time and synthetic messages being guessed correctly 78% of the time (participant results from 20 games of 10 questions each).

This also goes to show that the addition of a time bonus was necessary, considering that participant scores were close in terms of the answers given to questions, also somewhat stemming from the number of answer options available (a maximum of 4 options are displayed, this naturally goes down to 2 or 3 when conversations from smaller groups are uploaded).

Informal user testing was conducted with small groups of friends. Feedback indicated that the game was engaging and that the AI-generated messages were often convincingly similar to real ones. Players appreciated the transparency provided by the similarity scores and found the superlatives to be a fun addition. Players also made suggestions on the UI and game design, which later got implemented (the biggest of these suggestions being the addition of a "Share Results" button similar to Wordle). Players also highlighted the ability to upload multiple different chats is very useful.

6 Discussion

High attribution accuracy for authentic messages and convincing AI-generated content demonstrate the system's proficiency in recognizing and imitating distinct texting styles. State-of-the-art performance and interpretability are achieved by combining statistical and embedding-based similarity metrics. But sometimes, especially for users with little data or very different styles, the model generates generic or less convincing synthetic messages.

The quantity of chat data available for each user has a significant impact on the quality of both text generation and personality modeling. Low-quality synthetic messages and less unique profiles can result from small datasets.

Without additional modification, the system might not translate well to other languages or more formal communication styles because it is designed for informal, English-language chat data (such as that found on WhatsApp and Discord). Existing biases in the chat data, such as the overrepresentation of frequent speakers or particular linguistic patterns, run the risk of being reinforced.

Because of their unusual stylistic traits or extremely distinctive vocabulary, some users are consistently easier or harder to guess. The usefulness of these measures is confirmed by the fact that players typically perform better on messages with high distinctiveness and BERT similarity scores. Reflection on digital identity and the subtle clues that differentiate different communication styles is encouraged by the game format. Generic or context-agnostic messages frequently result in misattributions, underscoring the significance of feature selection and data quality. The system's reliance on user-provided data means that fairness and representation are directly tied to the diversity and balance of the input chats.

7 Conclusion and Future Work

TalkTagger offers a new and interactive method for attribution of style and personality modeling in online conversations. The system offers a fun game as well as insight into the subtle aspects of online communication by fusing transformer-based text generation, custom feature engineering, and contemporary similarity metrics.

Future work on TalkTagger will concentrate on:

- Using sentiment analysis to create more complex superlatives and richer message context.
- Adding support for more chat platforms and languages.
- Improving AI-generated messages' quality and variety, possibly by fine-tuning or using bigger datasets.
- Using the game as a scalable web-based platform to test and engage more users.

Beyond party games, the underlying technology has potential applications in chatbots, text stylization tools, authorship attribution, and digital identity research.

-

All code is available on https://github.com/NilAtabey/TalkTagger.