Bot Or Not: Synthetic Text Generation in Social Media Contexts Using Large Language Models

Written by David Holcer[1]*

[1]McGill University
845 Sherbrooke St W
Montreal, Quebec H3A 0G4

Abstract

The rise of bot accounts on social media platforms presents challenges in distinguishing synthetic from authentic user accounts. This research focuses on developing methodologies to generate bot accounts and synthetic text posts for an adversarial competition aimed at evaluating the efficacy of bot detection systems based primarily on the text within a user's post, devoid of additional metadata typically associated with user accounts such as number of likes or number of followers. Using techniques such as Markov chain models, fine-tuned large language models (LLMs), and probabilistic text augmentation techniques, the study explores various strategies for enhancing authenticity in bot-generated content to more reliably deceive bot detection methods. Key techniques include integrating human-like typographical errors, statistical patterns, and probabilistic time distributions into synthetic posts to increase the difficulty of detection. Results indicate that incorporating advanced text augmentation and LLM-based strategies can reduce detection rates. However, continued development is necessary to mimic nuanced human behaviors effectively. Future work can refine these methodologies, explore group bot dynamics, and analyze sentiment-based text patterns to further challenge bot detection systems.

Introduction

With the rise in popularity of social media platforms in the last decade, more users are flocking to these platforms to connect with their social network digitally. Users on these platforms may create accounts to represent their person, their interests, or their pets, and post their thoughts, feelings, the news, or funny messages. However, not all user accounts are created for these reasons. As with all great technologies, they may be used for purposes beyond their original intent. Nefarious actors have arisen and exploited vulnerabilities for personal or organizational gain. Bot or sybil accounts are accounts whose post are synthetically generated often by automated means detached from an individual. They may post to spread false information, promote a political cause, or simply to gain popularity and elicit discourse. With far reaching repercussions from impact on U.S. presidential elections to the rise of extremist ideologies and conspiracy theories, the effect of bots is undoubtedly profound. Despite attempts by many platforms to suspend or restrict access to such accounts on their platforms, differentiating bot accounts from real users is not straightforward. (Hayawi 2024)

Out of the need to better develop, understand, and create metrics for detecting bot accounts and synthetic text-based posts in the context of social media spurred the Bot or Not research project. This research relies on an adversarial, digital competition consisting of two teams competing against each other. On one side the bot team has as its primary goal to inject users and tweet-like posts into the competition environment without detection. Simultaneously, the bot detector's primary objective is to discern the bot accounts from the real user accounts. The digital competition is static and relies on an existing set of posts, textual based data obtained from the primarily text-based social media platform X formerly known as Twitter. The competition is divided into sessions and subsessions within each session, spanning a time period of two days. The bot teams must create a set of users and posts to inject into a session, which consists of four subsessions. Meanwhile, given the totality of the datasets that span all four subsessions, the detector team must discern between real users and artificially generated bot users/posts by outputting a yes or no response to the question 'Is the given user a bot?' for each user in the dataset, accompanied by a score ranging from 0 to 100 representing the confidence in the given decision. The overview of the competition is outlined in Figure 1. below.

The author's work focuses on the development of bot user accounts and the synthetic generation of posts associated with each bot. User accounts are comprised of certain metadata including:

name: The user's full name, which may include emojis or special characters.

username: An alphanumeric string handle uniquely identifying a given user.

description: A brief string description giving a brief overview of the users interests, hobbies, or affinities.

location: An optional string parameter detailing the location of the user or an arbitrary location description if the user is based predominantly online.

User posts in the context of this research had to adhere to the following constraints:

Text-Only: Only text-based posts were considered, excluding content with memes, images, or videos.

*Under the supervision of Derek Ruths, McGill University.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

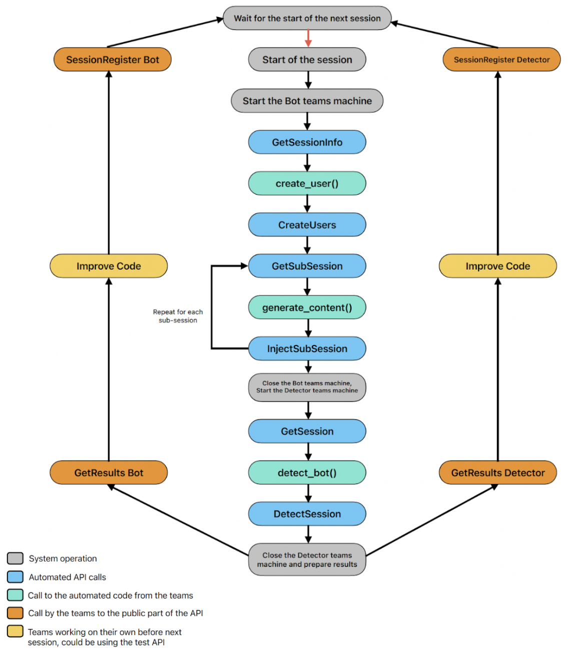
Wait for the start of the next session
SessionRegister Bot
Start of the session
SessionRegister Detector
Start the Bot teams machine
GetSessionInfo
create_user()
CreateUsers
Improve Code
GetSubSession
Improve Code
Repeat for each sub-session
generate_content()
InjectSubSession
Close the Bot teams machine, Start the Detector teams machine
GetSession
GetResults Bot
detect_bot()
GetResults Detector
DetectSession
Close the Detector teams machine and prepare results
⬜ System operation
🟦 Automated API calls
🟩 Call to the automated code from the teams
🟧 Call by the teams to the public part of the API
🟨 Teams working on their own before next session, could be using the test API

Figure 1: Competition Overview whereby both bot and detector teams interact with the competition infrastructure.

No Dynamic Metadata: Likes, followers, comments, replies, and reposts are excluded. This is because real users on X (Twitter) already have this metadata, but newly created bot users would not naturally accumulate it. Allowing random numbers for these metrics would lack credibility and have no way of being verified as on X (Twitter).

No Links: Posts containing links are not allowed because bot detectors could gain information by verifying their authenticity. To address links added by X (Twitter) for posts referencing other posts, images, or links, existing posts were anonymized by converting links to the generic link 'https://t.co/twitter_link'. This link was used as a placeholder in lieu of attached images, memes, and other linked content.

No Mentions: Mentions are anonymized because they can only reference existing users, and bot detectors could gain information by verifying their authenticity. Mentions may be included in posts represented as '@mention'.

User Activity Requirements: Each user in the dataset, including bots, must have between 10 and 100 posts to ensure they appear active.

The author took various approaches when conducting research and generating bots for the competition.

Manual Generation: Initially, synthetic posts were created manually by studying real social media posts. This approach allowed for controlled experimentation with authentic writing styles but lacked scalability and complexity.

Markov Chain Models: Leveraging publicly available articles, this method generated text by sequentially generating words using Markov chain models and an input source article. While useful for mimicking general patterns, this approach produced incoherent sentences and lacked semantic complexity, leading to easy detection within the competition environment.

Large Language Models (LLMs): The author trained Meta's Llama 3.1 LLM on personal text messages to simulate casual conversations with typographical errors and abbreviations. When this method proved insufficient, OpenAI's ChatGPT LLM gpt-4o-mini was employed. ChatGPT generated synthetic users and posts using pre-existing datasets. By prompting the LLM with existing sample outputs and specific prompts outlining instructions for metadata and tweet creation, the credibility of the output text increased, supported by increased performance in the competition environment.

Text Augmentation and Enhancement: To combat patterns typical of LLM outputs, text was modified using functions for typographical errors, case transformations, random spacing, and hashtag or mention additions. These adjustments were guided by statistical insights derived from real user and post data.

Probabilistic Time Distribution: Post timestamps were assigned based on weighted probabilities obtained from a given session's metadata, better reflecting real user activity and enhancing the realism of the given bot's activity.

These methodologies were implemented using Python code under a competition framework designed by Émile Ducharme and Aviv Shlomi. All source code the author developed throughout this research may be found in the cited Github repository. (Holcer, 2024)

Methodologies

Initial Approach – Manual Generation

The author's initial approach for post generation involved manually creating synthetic Twitter users and tweets by studying and adapting real posts from the platform. By examining how real users tweet and interact, the generated tweets remained realistic but included artificial profiles and content. This approach provided control over the dataset and allowed for a rapid first method to test the competition infrastructure. The method focused on capturing authentic writing styles and semantic patterns consisting of typographic errors or incomplete sentence structures. The synthetic content reflected genuine social media interactions. After generating a set of 12-13 posts for 2 users, an algorithm was written to distribute the posts into four groups, corresponding to the four subsessions per competition session. Each post was then assigned a random time value based on the competition session's start and end times. Although this initial approach provided a suitable method to test the competition infrastructure, it lacked complexity in methodology and did not fare well in competition as shown in Figure 2. below. However, as this was the first iteration of the competition, this submission was not detected by one bot, "seunghyundetector2", which could signify the additional improvements necessary for accurate detection of bot accounts early into the research and competition process. This bot detector is shown in light green in Figure 2. below.

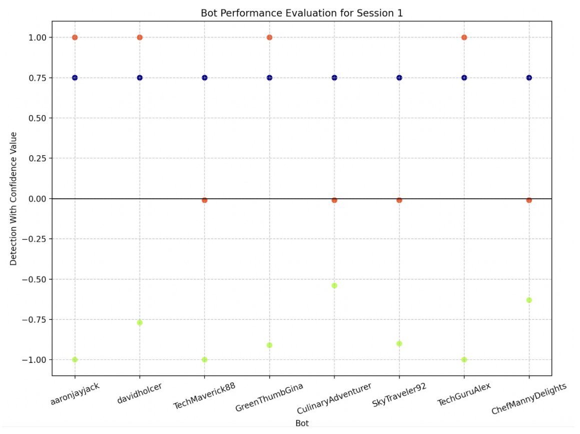

Figure 2: Session 1. Results. The author's generated users are 'aaronjayjack' and 'davidholcer'. Positive scores correspond to predictions of bots whereas negative scores correspond to predictions of real users.

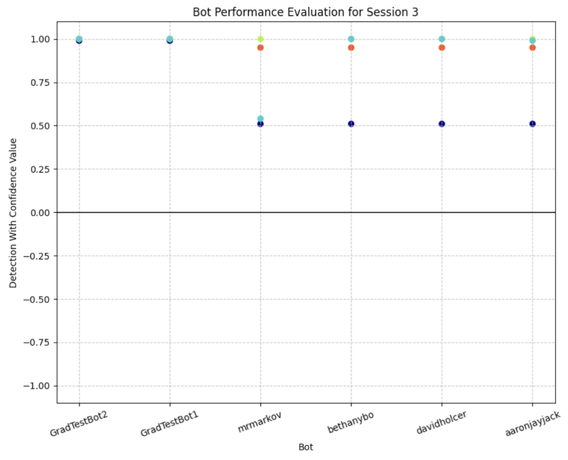

Figure 3: Session 3. Results. The author's generated users are 'mrmarkov', 'bethanybo', 'davidholcer', and 'aaronjayjack'. Positive scores correspond to predictions of bots whereas negative scores correspond to predictions of real users.

Markov Chain Models

A second method was introduced to generate a user, whose posts were created using Markov chain models for text generation. [1]

The sources for the Markov chain model were publicly accessible articles from *Wikipedia* and *The New York Times*. The generated posts were randomly distributed across the four subsessions and were assigned timestamps stochastically within the session's time bounds. Despite incorporating this user-post generation approach, all bot detectors successfully identified the created users as bots during this iteration, Session 3, of the competition, as shown in Figure 3 right. This outcome is likely due to two primary factors:

1. The lack of semantic meaning in the generated text, which often consisted of incomplete or incoherent sentences produced by the Markov chain model.
2. The randomized timing distribution of posts, which remained a potentially detectable feature for the bot detectors.

Results Parser

To enhance the understanding of the tweet dataset and to better quantify the performance of bots and detectors within the competition, a results parser was developed. The parser extracts and organizes key insights from session results. The extracted information includes:

Top n-grams for the posts in the dataset. [2]

Bot vs. real user statistics for the following metrics:

- Total Posts
- Total Links in Posts
- Average Links per Post
- Total Hashtags in Posts
- Average Hashtags per Post
- Total Emojis in Posts
- Average Emojis per Post
- Total Word Count in Posts
- Average Words per Post
- Total Character Count in Posts
- Average Characters per Post

A heatmap of the time distribution of posts divided by real vs bot accounts. An example is shown in Figure 4. below for session 6.

For each bot detector, confidence scores were plotted against the proportion of posts, visually distinguishing correct and incorrect predictions of bot vs. non-bot accounts for each bot detector. An example of this plot is shown in Figure 5. and Figure 6. below, corresponding to competition session 6.

Additionally, the Matthews Correlation Coefficient (MCC) was shown for each bot detector. [3] An example figure for session 6 is shown in Figure 7. below.

[1]Markov chain models generate text by predicting the next word in a sequence based on the probability of it following the previous word or words. The model uses a predefined dataset to calculate probabilities, outputing generated text that mimics the patterns of the source material.

[2]N-grams are contiguous sequences of n words from a given sample text. For example, 2-grams (bigrams) are continuous sequences of two words within the sample text.

[3]The MCC was calculated for each bot detector to provide a balanced measure of prediction quality. The MCC is a performance metric used in binary classification that accounts for true positives, true negatives, false positives, and false negatives. It provides a single score ranging from -1 to +1, where +1 indicates perfect predictions, 0 indicates random prediction, and -1 indicates total disagreement between predictions and actual outcomes.

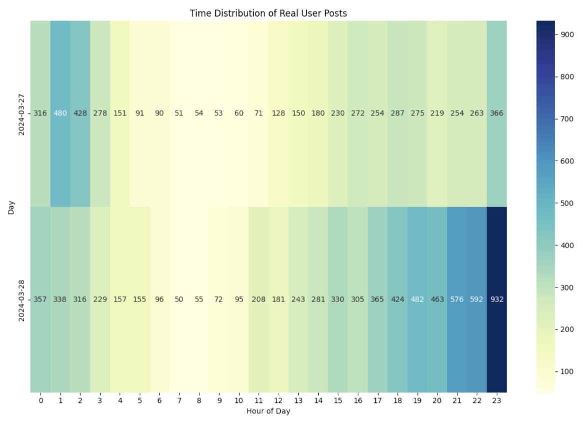

Figure 4: Heatmap representing the time distribution of posts obtained from the Results Parser corresponding to the results obtained from Session 6.

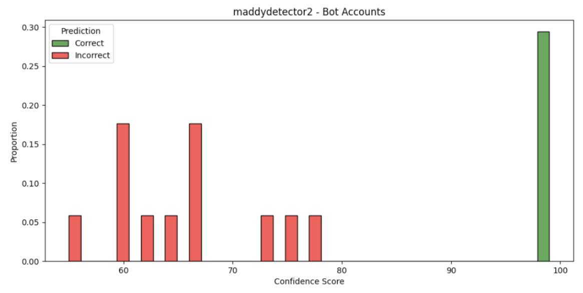

Figure 5: Plot representing the confidence scores obtained from the Results Parser for bot detector 'maddydetector2' corresponding to the Bot accounts. Results obtained from Session 6.

Finetuning Large Language Models (LLMs) using Text Messages

The next approach was to train an LLM using a dataset of the author's text messages. [4] In theory, since text messages conversations are often casual conversation, including the use of abbreviation and typos, by simulating such conversations with a bot, it would be difficult for bot detectors to discern the bot accounts from the real accounts. This approach relied on finetuning a Llama 3.1 LLM model [5] obtained via the HuggingFace API [6] and finetuning using a text database

[4]Large Language Models (LLMs) are advanced machine learning models designed to process and generate human-like text by understanding natural language patterns. In the context of Natural Language Processing (NLP), LLMs analyze large amounts of text data to predict word sequences, generate coherent sentences, and perform tasks including translation, summarization, and enhancement of text. By fine-tuning an LLM with a specific dataset, the model can adapt its output to reflect the nuances, vocabulary, and tone of the given data.

[5]Llama 3.1 is an LLM created by Meta Inc.

[6]HuggingFace is a public repository of artificial intelligence and large language models.

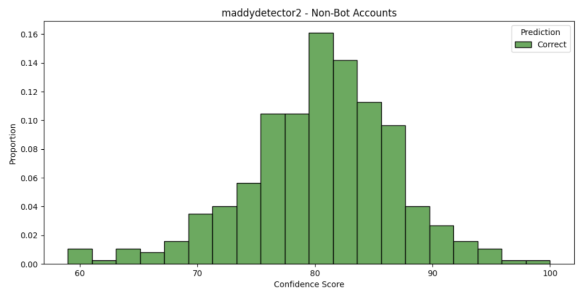

Figure 6: Plot representing the confidence scores obtained from the Results Parser for bot detector 'maddydetector2' corresponding to the Non-Bot accounts. Results obtained from Session 6.

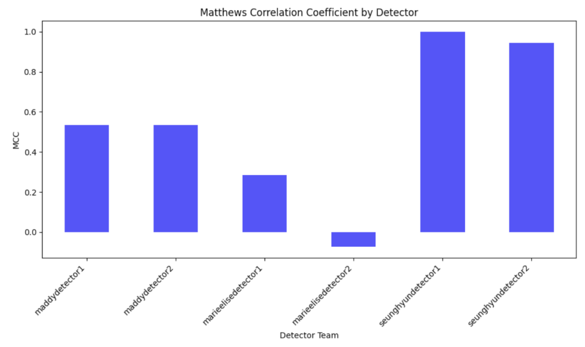

Figure 7: Plot representing the MCC values obtained from the Results Parser for all bot detectors in Session 6.

obtained from iMessages. [7] However, in practice, the results obtained from this LLM training did not show sufficient success to perform well in the real competition. Thus it was decided to change approaches and use another LLM for user and post generation, Open AI's ChatGPT model gpt-4o-mini.[8]

ChatGPT Prompt Completion

ChatGPT Prompt Completion generated user information using a random subset of existing real "users." The ChatGPT Completion API was used for this purpose. This process was completed in 3 steps:

1. Gathering a random subset of 20 users from the results files from Sessions 3-5, and extracting user information in the following text format:

[7]iMessages is a proprietary messaging application from Apple Inc. Messages were collected from the "chat.db" database and extracted as text information associated with a given contact, and time sent.

[8]Open AI's ChatGPT is an LLM trained on a large, diverse corpus of internet text to generate human-like text and perform conversational tasks. The model used in this research, gpt-4o-mini, is a smaller-scale version designed for efficient experimentation. It uses contextual understanding and probabilistic text generation to produce realistic outputs, making it an excellent choice for simulating synthetic social media posts.

Name – ‘{Name of user}’
Username – ‘{Username of user}’
Description – ‘{Description of user}’
Location – ‘{Location of user}’
Where the information within the brackets {} indicates inserted information specified by the given description. Each user’s information was separated by an empty line.

2. ChatGPT Completion API was prompted to generate user information using the following prompts:
   a) System Prompt:
   You are a creative assistant that generates realistic twitter user metadata.
   b) User Prompt:
   Generate user metadata in a similar format to the following dataset.:
   This user prompt was then followed by the 20 user’s information gathered from Step 1.

This process returned 20 User’s information in the format specified in Step 1. From this text output, key user information: *Name, username, description, location* fields were extracted and used for user generation. For the competition Session 6, five (5) of these generated Users were selected for insertion within the competition infrastructure.

ChatGPT’s Completion API was also used to generate posts using a random subset of existing real “posts” from the results files obtained from Sessions 3-5.

1. Gathering a random subset of posts 60 posts by scrambling the results files from Sessions 3-5.
2. ChatGPT Completion API was prompted to generate posts using the following prompts:
   a) System Prompt:
   You are a creative assistant that generates realistic tweets. No links, use ‘https://t.co/twitter_link’ instead. No mentions, use ‘@mention’ instead.
   b) User Prompt:
   Generate tweets in a similar format to the following dataset.:
   This user prompt was then followed by the 60 sample posts gathered from step 1.

This process returned 60 text posts. However, not all 60 posts were used for each user. A normal distribution was used to select the total number of posts to assign to each user. The mean of the normal distribution was the ‘users_average_amount_posts’ found within the session metadata information. The standard deviation was one quarter of this value. Using a normal distribution with the given parameters, a selected number of posts were determined and assigned to a given user, adhering to the 10 post minimum value. Then, given the total output of 60 tweets, a subset of posts were obtained such that the total number of posts equaled the predetermined number assigned to each user.

The following settings were used in the parameters of ChatGPT’s Autocomplete API: *model:* “gpt-4o-mini”
temperature: 1
The temperature parameter is a float value that ranges from 0 to 2. Higher values generate more random output while lower values will produce more deterministic outputs.
max_completion_tokens: 16384
The maximum number of tokens the model can return.
presence_penalty: 2.0
The presence penalty parameter is a float value ranging from -2.0 to 2.0. Higher values decrease the output’s likelihood of repeating input lines. A value of 2.0 was used to highly discourage the repetition of sample data. (OpenAI, 2024)

Text Scramble and Enhancements

After posts were generated by ChatGPT, further enhancements were made to increase the similarity of the output to posts found within the sample dataset and to prevent exhibiting patterns archetypal of an LLM’s output. Such patterns included: lack of typographical errors, consistently capitalized sentences, infrequent use of emojis. To address each of these potential markers, typographical errors and sentence capitalization variations were introduced into the output tweets using a weighted probability distribution. Additionally, links, hashtags, and user mentions were augmented. For typographical errors, the ‘typo’ python library was used to generate the following textual modifications.

- *Character Swap:* With a 15% chance, this function swaps two consecutive random characters in a word, introducing minor typographical variations.
- *Character Skip:* With a 15% chance that this function will skip a random character in a word, simulating a typing omission.
- *Extra Character Addition:* With a 10% chance, this function adds an extra letter next to a random character in a word, choosing a key that is a keyboard neighbor to the original.
- *Nearby Character Replacement:* This function, with a 10% chance, replaces a random character in a word with one from a neighboring key on the keyboard, mimicking a common typing error.
- *Similar Character Replacement:* With a 10% chance, this function substitutes a random character in a word with another visually similar character (e.g., ‘O’ with ‘0’).
- *Space Skipping:* This function has a 10% chance to remove a random space from the string, simulating accidental concatenation of words.
- *Random Space Addition:* With a 10% chance that this function will add a space at a random position within the text, splitting a word unexpectedly.
- *Repeated Character Addition:* With a 10% chance, this function repeats a random character in a word, mimicking a typing error.
- *Single Character Simplification:* This function has a 10% chance to replace a set of repeated consecutive characters with a single instance of the letter.

(Kumar, 2024)

Each of the errors above count as 1 typo. For each post, the number of typographical errors to generate was decided using a normal distribution dependent on the number of characters in the tweet. The normal distribution was modelled using a mean of 0.9 typos per 100 characters with a standard deviation of 0.42 typos per 100 characters. These values were chosen manually by hypothesizing typo occurrences within tweets. The total typo count was then determined by rounding the output of this normal distribution to the nearest whole number, using the total number of characters in the original tweet and requiring a minimum of 0 typos.

Given the analysis gleaned from the session results parser, specifically the statistical frequency of hashtags, mentions, and links, it was decided that the generated tweets required further modification to better conform to discovered statistical insights.

Thus, the following additional functions were created and called on the generated tweet after introducing typographical errors.

- *Hashtag Addition:* With a 60% chance, this function randomly selects between 1 and 5 words from the text and converts them into hashtags by adding the '#' character before each selected word.
- *Link Duplication:* This function, with a 30% chance, duplicates every occurrence of a link within the text, effectively doubling the number of hyperlinks.
- *Mention Addition:* With a 40% chance, this function appends between 1 and 3 mentions (e.g., '@mention') to the end of the tweet.
- *Word Exaggeration:* This function exaggerates specific words with certain rules. If a word ends in an exclamation mark ('!') or question mark ('?'), the punctuation is repeated between 1 to 5 times inclusive, chosen randomly. For other words longer than one character, there is a 5% chance that the last character is repeated between 2 and 6 times inclusive, chosen randomly.
- *Case Transformation:* For each word, there is a 10% chance it becomes fully capitalized. Additionally, there is a 5% chance that the word is transformed into an alternating case format, randomly (with 50% chance) transforming each character into uppercase or lowercase.
- *Randomized Sentence Case:* With an 80% chance, this function changes the first letter of the text to lowercase, mimicking the tendency of humans to neglect proper sentence capitalization.

An example of a modified tweet follows, after calling all above functions on the input tweet.

Original Tweet: "Nominees are in, and OMG-I can't even choose the best one this week! :)"

Augmented Tweet: "nOMINEES ARE in, and OMG-I #can't even choose the #6esT one this week!!!! :)"

Probabilistic Time Distribution

For the competition iteration using ChatGPT's autocomplete, posts were distributed using a weighted probability distribution. Given the percentage distribution of posts in a session, the posting frequency was reproduced to closely match the dataset's posts' time distribution. To do so, each time interval found in the session metadata was given an associated weighted probability depending on the fraction of total posts that were found within. Then, for each post per user, a time interval was randomly chosen using the preselected frequency weights. Within each time interval, a random time was sampled and attributed to the given post.

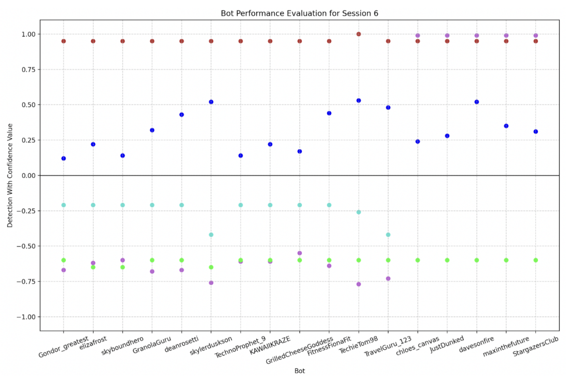

Figure 8: Session 6. Results. The author's generated users are 'chloes_canvas', 'JustDunked', 'davesonfire', 'maxinthefuture', and 'StargazersClub'. Positive scores correspond to predictions of bots whereas negative scores correspond to predictions of real users.

Discussion

The objective of the research was to develop a bot generation approach that would remain undetected within the bounds of the competition platform, and to better understand and detail the efficacy of the employed strategies. The implemented methodologies showed varying success within the competition infrastructure. The initial attempts of manual tweet generation were largely unsuccessful given their simplistic, naïve approach and randomly distributed time attribute. Sessions 1 and 3 show that despite some initial success (undetected bot) in Session 1, as the bot detectors developed their methods, the author's users and posts generated in Session 3 were all detected as shown in Figure 2. and Figure 3. After an unsuccessful attempt to train the Llama LLM on a dataset of text messages, OpenAI's ChatGPT Completion API was used on the LLM gpt-4o-mini. After synthetic generation of users and posts using pre-existing user and post datasets run from prior competition sessions, generated posts were systematically enhanced via a combination of introducing typographical errors within generated tweets and calling functions to further modify words and statistically introduce human-like textual patterns. Results improved as a bot, "marieelisedetector2" failed to detect the created users as shown in Figure 8. above. This can likely be attributed to the introduction of typographical errors as well as the probabilistic time distribution of generated posts.

These successful results indicate the efficacy and potential benefit of continuing to use LLMs in the context of synthetic user and post generation. Further research and development of the employed methodologies could prove useful to decrease the detection rate of generated bot accounts.

Conclusions and Future Work

In this study, a systematic approach to bot development was undertaken, exploring methods such as handcrafted generation, Markov Chain models, and LLMs such as ChatGPT gpt-4o-mini. While manual approaches lacked complexity and Markov chains struggled to perform well in competition, LLMs combined with text augmentation proved more effective in mimicking human behavior and generating more sophistication, supported by improved competition results.

Despite these advancements, results indicate that more research would be beneficial in addressing the limitations of current methodologies and further refining human-like text generation techniques.

Future work may include:

Fine-tuning LLM Outputs: Further optimization of ChatGPT/LLM models to improve coherence and variety in generated post topics.

Group Bot Dynamics: Developing algorithms that simulate coordinated behaviors among bot accounts, similar to group interaction or swarm intelligence algorithms.

Incorporating Profanity and Sentiment: Adding sentiment-based generation and replicating profanity trends observed in the original dataset could further enhance realism and fool bot detectors.

User Categorization: Separating users into different categories such as ‘professional’, ‘casual’, and ‘conspiracy’ categories to tailor content generation according to the posting and semantic patterns of each group.

These topics aim to refine synthetic generation techniques and contribute to the broader understanding of existing digital bot detection challenges.

References

David Holcer. 2024. Bot or Not Github Repository. https://github.com/davidholcer/BattleBotsTemplate. Accessed: 2024-12-27.

Hayawi, K.; Saha, S.; Masud, M. M.; Mathew, S. S.; and Kaosar, M. 2023. Social media bot detection with deep learning methods: a systematic review. *Neural Computing and Applications*, 35(12): 8903–8918.

OpenAI. 2024. ChatGPT Create Chat Completion API Documentation. https://platform.openai.com/docs/api-reference/chat/create. Accessed: 2024-12-27.

Ranvijay Kumar. 2023. Typo Python Library. https://pypi.org/project/typo/. Accessed: 2024-12-27.

Acknowledgements

The author would like to thank Professor Derek Ruths for his supervision and guidance throughout the research.

Additionally, the author is thankful towards Émile Ducharme and Aviv Shlomi for their dedication and support with the explanation and maintenance of the competition architecture, and for troubleshooting technical problems when they arose.