To Bot or not to Bot?

Seyedehsara Mirsepassi

seyedehsara.mirsepassi@studenti.unimi.it

Department of Computer Science, Università degli studi di Milano

Abstract: This study explores the effectiveness of multimodal transformer-based architectures for detecting AI-generated fake personas on social media platforms by using information from multiple sources, including text, profile images, and behavioral data. The model combines RoBERTa for understanding written content, user metadata like follower/following counts, and image features extracted using the CLIP vision encoder. The experiments were conducted using the TwiBot-22 dataset, which contains information on 1 million Twitter users. After testing different model setups and training strategies, the final version reached about 63% accuracy on a balanced test-set of 15,000 users. These results show that using multiple types of information together can improve bot detection over just relying on text.

1. Introduction

This project focuses on detecting AI-generated fake personas on Twitter platform. Such accounts, often referred to as "bots," use generated or stolen content to simulate real users. These bots can spread misinformation, manipulate trends or engage in spam activities, making their detection a key problem in social media integrity and security.

The goal of this project is to classify each user as either a human (label 0) or a bot (label 1) by using multimodal data—which is information from multiple sources—rather than relying on text alone. The model integrates the following three types of inputs:

- **Profile and tweet text** This includes the user's profile description (bio), username and tweet content. Analyzing the style, coherence, and language patterns of this text helps detect unnatural phrasing, unusual repetition or linguistic signals that indicate automated behavior.
- **Behavioral metadata:** Quantitative indicators such as the number of followers, followings, tweets and their ratios. Bots often exhibit abnormal activity patterns, like following many users without gaining followers or having large tweet volumes in short periods. These features help capture those patterns statistically.
- **Profile image content:** The user's profile picture is processed using CLIP, a model trained to understand image semantics. This allows detection of AI-generated or mismatched images that don't align with the user's stated identity.

By combining these diverse types of information, the model aims to learn a richer representation of each user and improve the detection of AI-generated fake accounts compared to what can be achieved by analyzing text alone.

2. Dataset and Preprocessing

The TwiBot-22 dataset contains over 1 million Twitter user accounts and their associated metadata. It includes:

- **user.json:** user metadata (username, bio, followers, followings, profile image URLs, etc.)
- **tweet_0.json to tweet_8.json:** tweet content
- **label.csv:** bot/human labels
- **split.csv:** Train/Validation/Test labels

This dataset includes 1,000,000 Twitter user accounts, with 860,057 labeled as human and 139,943 as bots. The dataset was split into 700,000 users for training, 200,000 for validation, and 100,000 for testing. On average, each user had 9.7 tweets, with a maximum of 10 and a minimum of 1.

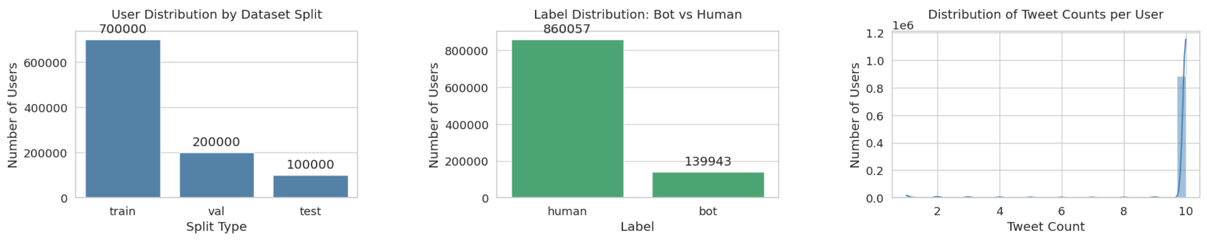

Fig. 1: Dataset Overview

The data preparation process involved multiple steps:

2.1. *Initial Data Loading and Preprocessing Challenges*

Working with TwiBot-22 posed several technical and computational challenges due to its size and structure:

- **Memory limitations while processing tweet data:** The tweet files (tweet_0.json to tweet_8.json) collectively contain around 100 GB of data, with each individual file being approximately 11–13 GB. Initially, an attempt was made to load the entire content using json.load(), which immediately caused RAM crashes in standard environments like Google Colab. Furthermore, the files are structured as a single large JSON array (not line-delimited), making it incompatible with memory-efficient parsers such as json.load().
  - **Solution:** A custom streaming generator was implemented using ijson to parse the JSON files incrementally. Each tweet was read one-by-one and matched against a predefined list of valid user IDs. If a match was found, the tweet text was saved into memory (limited to the first 10 tweets per user). This selective loading approach significantly reduced memory consumption and enabled full preprocessing on tens of thousands of users.
- **Efficient and reliable image handling:** TwiBot-22 provides paths to profile images but does not include precomputed image features. Downloading images on-demand during model training would have led to frequent failures due to missing URLs, timeouts, or internet instability.
  - **Solution:** A dedicated preprocessing phase was added to download all valid profile images in advance. To improve reliability and performance:
    - ∗ Download progress was saved using a last_idx.txt checkpoint to resume after failures.
    - ∗ Images were stored in subfolders of 10,000 each, e.g., 000000–009999, 010000–019999.
    - ∗ Corrupted or broken images were skipped automatically.

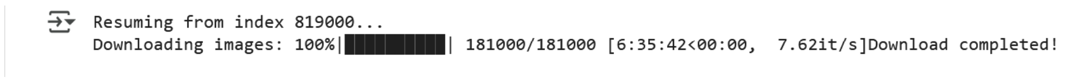
```
Resuming from index 819000...
Downloading images: 100%|██████████| 181000/181000 [6:35:42<00:00,  7.62it/s]Download completed!
```

Fig. 2: Valid profile images were downloaded in advance

2.2. *Filtering Strategy and Sampling*

- Users with fewer than 10 tweets were excluded to ensure sufficient textual data for RoBERTa to generate meaningful embeddings.
- Incomplete or corrupted user records were dropped during preprocessing by checking if a user was present in both the user dataframe ”user_df” and label dictionary ”label_dict”, and ensuring tweets were available.
- Users were categorized into bot and human using labels from label.csv. From these, a balanced subset of 50,000 bots and 50,000 humans was randomly selected.
- Experiments were conducted with larger sets (e.g., 100k bots + 100k humans), but such configurations exceeded available RAM and were therefore abandoned in favor of the 100k balanced sample.

3. Model Architecture

This section explains the model design, showing how data is transformed from raw inputs into final predictions.

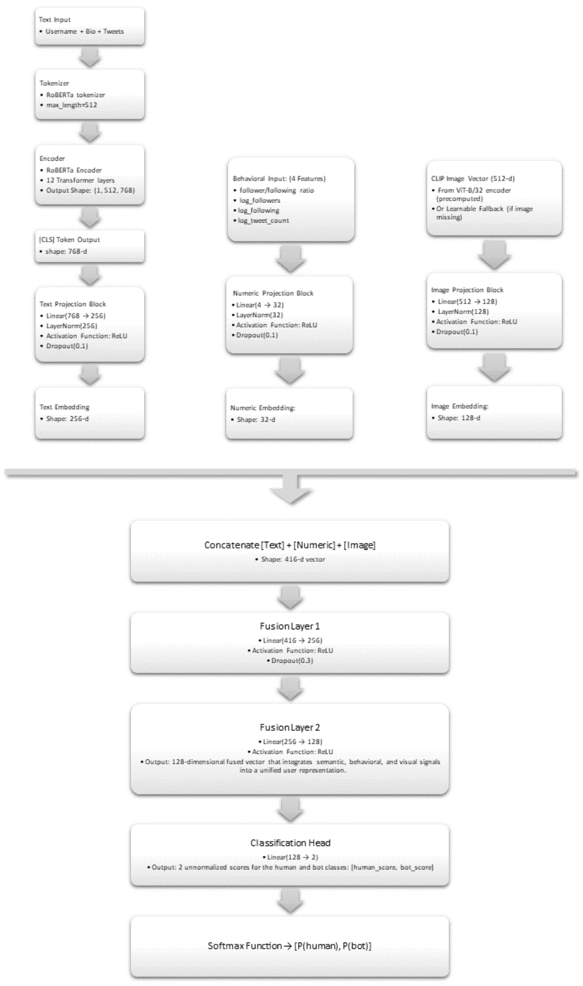
Text Input
• Username + Bio + Tweets
Tokenizer
• RoBERTa tokenizer
• max_length=512
Encoder
• RoBERTa Encoder
• 12 Transformer layers
• Output Shape: (1, 512, 768)
[CLS] Token Output
• shape: 768-d
Text Projection Block
• Linear(768 → 256)
• LayerNorm(256)
• Activation Function: ReLU
• Dropout(0.1)
Text Embedding
• Shape: 256-d
Behavioral Input: (4 Features)
• follower/following ratio
• log_followers
• log_following
• log_tweet_count
Numeric Projection Block
• Linear(4 → 32)
• LayerNorm(32)
• Activation Function: ReLU
• Dropout(0.1)
Numeric Embedding:
• Shape: 32-d
CLIP Image Vector (512-d)
• From ViT-B/32 encoder (precomputed)
• Or Learnable Fallback (if image missing)
Image Projection Block
• Linear(512 → 128)
• LayerNorm(128)
• Activation Function: ReLU
• Dropout(0.1)
Image Embedding:
• Shape: 128-d
Concatenate [Text] + [Numeric] + [Image]
• Shape: 416-d vector
Fusion Layer 1
• Linear(416 → 256)
• Activation Function: ReLU
• Dropout(0.3)
Fusion Layer 2
• Linear(256 → 128)
• Activation Function: ReLU
• Output: 128-dimensional fused vector that integrates semantic, behavioral, and visual signals into a unified user representation.
Classification Head
• Linear(128 → 2)
• Output: 2 unnormalized scores for the human and bot classes: [human_score, bot_score]
Softmax Function → [P(human), P(bot)]

Fig. 3: Model Architecture

3.1. *Textual Input Processing*

- The user's profile data — including username, profile description, and tweet history — is concatenated into a single long text block.
- This input is first passed to the **RoBERTa-base tokenizer**, which is based on Byte-Pair Encoding (BPE). It splits the text into subword units and assigns each a unique integer ID from a fixed vocabulary size of 50,265 tokens. Special tokens such as `<s>` (start of sequence) and `</s>` (end of sequence) are added. The output includes a list of token IDs representing the input sentence (max 512 tokens) and a binary list marking real tokens (1) vs padding (0):

```
input_ids = [0, 1332, 435, 9871, 204, 4, 2, 1, 1, ..., 1]  # Length 512
attention_mask = [1, 1, 1, 1, 1, 1, 1, 0, 0, ..., 0]
```

- Before entering the first transformer layer, RoBERTa adds learned **positional embeddings** to each token embedding to retain word order information, since transformers are not inherently sequential.
- The tokenized output is then passed to the **RoBERTa-base transformer encoder**, a pretrained model from Hugging Face consisting of 12 identical transformer layers. Each layer is composed of two key subcomponents: a multi-head self-attention block and a feedforward neural network (FFN):

1. **Multi-head Self-Attention Block**
   The self-attention mechanism allows each token in the input sequence to dynamically attend to all other tokens, enabling the model to understand contextual relationships regardless of their position.

   Each self-attention block contains 12 parallel attention heads (a single self-attention mechanism) that capture different types of relationships in the text.

   After the attention output is computed, a residual connection is added (to preserve the original input) and passed through a layer normalization step to stabilize training.
2. **Feedforward Neural Network (FFN)**
   Following the self-attention mechanism, the 768-dimensional vector output is passed through the FFN, which is a two-layer fully connected network with a hidden size of 3072 and GELU activation. The FFN processes each token representation independently, helping it refine its meaning using the context gathered from attention block.

   - Input: 768-dimensional vector
   - Hidden layer: 3072 dimensions (using GELU activation)
   - Output layer: back to 768 dimensions

   A second residual connection and layer normalization follow the FFN block.

This sequence of operations is repeated **12 times**, once per transformer layer. All layers share the same structure but with different learned parameters. The input to the model is a sequence of up to 512 tokens. Each token is represented as a 768-dimensional embedding. Stacking multiple such layers allows the model to build deep and rich semantic representations of text, which are critical for understanding user behavior in this task. After passing through the 12 layers, the output has shape:

$$\text{Output Shape: } (\text{batch\_size=}1, 512, 768)$$

- Among these outputs, the first vector ([CLS] token - represented as `<s>` in RoBERTa) is extracted. This vector summarizes the entire input's semantic content.
- This [CLS] vector is passed through a projection block (Neural Network) to reduce its dimensionality and enhance training stability:

  Linear(768 → 256) → LayerNorm → ReLU → Dropout
- The result is a 256-dimensional text embedding that captures the semantic and stylistic features of the user's profile and tweet content.

3.2. Behavioral Input Processing

- From each user's metadata, four numerical features are computed:
  - Follower/following ratio
  - Log of follower count
  - Log of following count
  - Log of tweet count
- Logarithmic transformation is used to compress large ranges and reduce skewness in distributions.
- The resulting 4D feature vector is passed through a dedicated feed-forward projection block before fusion with other modalities:

  Linear(4 → 32) → LayerNorm → ReLU → Dropout
- This results in a 32-dimensional numeric embedding that encodes user activity and behavioral patterns in a normalized feature space.

3.3. *Visual Input Processing (CLIP)*

- After all valid images were downloaded during preprocessing, each profile image is passed through the CLIP ViT-B/32 image encoder, a pretrained transformer model from OpenAI designed to generate semantic embeddings.
- CLIP generates a 512-dimensional vector for each image, capturing abstract concepts like gender cues, image quality, realism, and more using model_clip.encode_image(...).
- Initially, users without a valid profile image were simply assigned a fixed zero vector of shape ”512” like [0, 0, ..., 0], to maintain input consistency. However, during experimentation, it became clear that this approach introduced a hard distinction between users with and without images, which could negatively impact the model’s generalization:

```
clip_feature = torch.zeros(512)
```

  To address this, the zero vector strategy was replaced with a learnable fallback embedding — a randomly initialized trainable vector of the same size ”512”. This embedding is updated during model training alongside other weights. It adapts based on how users without images behave in combination with text and numeric features. As a result, even users without images contribute meaningful information, allowing the model to better generalize and reducing the performance gap caused by missing visual features.

```
self.missing_clip_embedding = nn.Parameter(torch.zeros(512))
nn.init.normal_(self.missing_clip_embedding, mean=0.0, std=0.02)
```

- The 512-dimensional image feature is passed through another projection block:

  Linear(512 → 128) → LayerNorm → ReLU → Dropout
- This results in a 128-dimensional image embedding, providing semantic visual context for the user.
- These features were stored in a dictionary by user ID and serialized to a file (clip_features.pt) for quick loading during model training.

3.4. *Fusion Layer*

- The three embeddings — text (256), numeric (32), and image (128) — are concatenated into a single 416-dimensional feature vector.
- This combined vector is passed through two dense layers to fuse modalities:

  Linear(416 → 256) → ReLU → Dropout(0.3)

  Linear(256 → 128) → ReLU
- The output is a 128-dimensional fused vector that integrates semantic, behavioral, and visual signals into a unified user representation.

3.5. *Classification Layer*

- The fused vector is passed to a final dense layer:

  Linear(128 → 2)
- This produces two output values (logits), representing the unnormalized scores for the human and bot classes.
- A softmax function is applied to convert these scores into class probabilities:

  softmax(logits) → [P(human), P(bot)]
- The model selects the class with the highest probability as the final prediction.

This multimodal design enables the model to make more informed decisions by leveraging complementary cues from language, behavior, and visual appearance.

3.6. Training Configuration

During model training, several strategies were explored to improve performance, especially on harder-to-classify samples.

- **Cross-entropy loss with label smoothing** was used as model's primary loss function. Cross-entropy compares the predicted class probabilities to the true labels and penalizes incorrect predictions more heavily the more confident they are. **Label smoothing** slightly softens the target labels (e.g., changing 1 to 0.95 and 0 to 0.05), which prevents the model from becoming too confident and helps it generalize better to unseen data. This loss is applied to the final logits output by the classifier head (one for "human", one for "bot") before softmax is used to calculate class probabilities.

$$loss\_fn = nn.CrossEntropyLoss(weight=weight\_tensor, label\_smoothing=0.05)$$

- **Focal Loss** was also implemented and tested. It modifies cross-entropy to focus more on difficult examples (e.g., users that are frequently misclassified), by down-weighting already well-classified samples. This was intended to improve recall, especially for more diverse human accounts. However, focal loss sometimes caused unstable training and was ultimately not used in the final model due to better and more consistent results from the regular cross-entropy with label smoothing.
- **RoBERTa was frozen** for the first epoch of training. This prevents the pretrained text encoder from being disrupted by random gradients from the newly initialized fusion and projection layers, allowing those newer components to stabilize first.
- **Evaluation metrics** included:
  - **Macro F1 score:** Calculates the F1 score (harmonic mean of precision and recall) independently for each class and then averages them.
  - **Weighted F1 score:** Similar to macro F1, but each class's F1 score is weighted by its number of true instances (support). This ensures that performance on more frequent classes has a greater impact on the final score. It is useful when classes are imbalanced.

3.7. Experimental Results and Threshold Tuning

After training the final multimodal model on a balanced dataset of 100,000 users (50K bots and 50K humans), evaluation was conducted on a 15,000-user test set. Using the default classification threshold of 0.5 (where the model predicts "bot" if its softmax probability is greater than 0.5), the model achieved a test accuracy of 63%, with a macro F1 score of 0.6287. Precision and recall scores were well-balanced for both classes: 0.63 for human and 0.63–0.64 for bots, suggesting the model could detect both classes with reasonable symmetry.

Table 1: Classification Report (Default Threshold = 0.5)

Class	**Precision**	**Recall**	**F1-score**	**Support**
Human	0.63	0.62	0.63	7500
Bot	0.63	0.64	0.63	7500
Accuracy			0.63	15000
Macro Avg	0.63	0.63	0.63	15000
Weighted Avg	0.63	0.63	0.63	15000

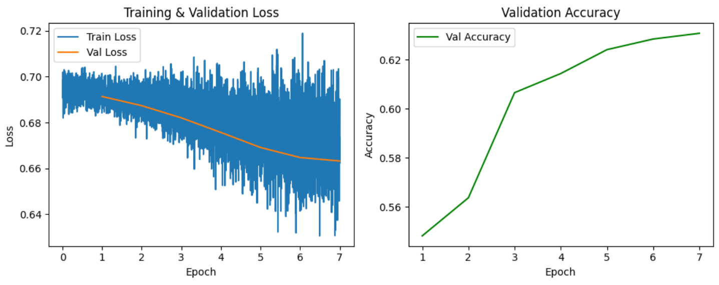

Fig. 4: Confusion Matrix

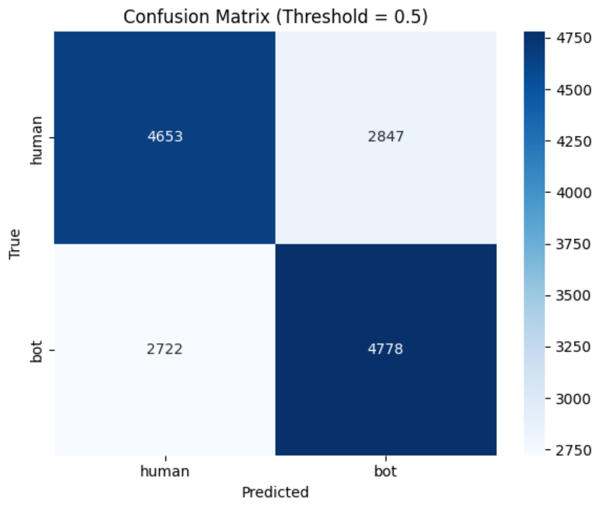

Fig. 5: Confusion Matrix

However, in real-world applications, one might want to favor higher recall for bots (to catch more suspicious accounts) or higher recall for humans (to avoid false positives). To explore this trade-off, threshold tuning was applied: instead of relying on the default threshold of 0.5, predictions were re-evaluated using multiple thresholds ranging from 0.3 to 0.6. This allowed for a more detailed view of the model's behavior across different operating points.

For example, when the threshold was lowered to 0.3, the model predicted nearly every user as a bot, achieving 100% recall for bots but 0% recall for humans. Conversely, increasing the threshold to 0.6 reversed the situation—97% recall for humans but only 10% for bots. These results demonstrate the importance of threshold selection depending on deployment goals—whether prioritizing precision, recall, or balanced performance.

Table 2: Classification Report at Various Thresholds

Threshold	**Human**			**Bot**			**Accuracy**	**Macro Avg**			**Weighted Avg**	
	Prec.	Rec.	F1	Prec.	Rec.	F1		Prec.	Rec.	F1	Prec.	F1
0.30	0.00	0.00	0.00	0.50	1.00	0.67	0.50	0.25	0.50	0.33	0.25	0.33
0.40	0.78	0.11	0.20	0.52	0.97	0.68	0.54	0.65	0.54	0.44	0.65	0.44
0.45	0.72	0.28	0.40	0.55	0.89	0.68	0.59	0.64	0.59	0.54	0.64	0.54
0.50	0.63	0.62	0.63	0.63	0.64	0.63	0.63	0.63	0.63	0.63	0.63	0.63
0.55	0.56	0.88	0.68	0.72	0.30	0.42	0.59	0.64	0.59	0.55	0.64	0.55
0.60	0.52	0.97	0.68	0.79	0.10	0.17	0.54	0.66	0.54	0.43	0.66	0.43

4. Conclusion

This project aimed to develop a robust system for detecting AI-generated fake personas (bots) on Twitter using a multimodal transformer-based model. The model combines three sources of information: text (from profiles and tweets), behavioral metadata and profile image features.

Development began with a text-only RoBERTa baseline, which reached 73% F1 score but struggled to consistently identify real users due to high linguistic variability. To address this, behavioral features—such as follower-following ratio and tweet count—were incorporated, helping the model better recognize inactive or suspicious account patterns.

Visual information was added next using CLIP to encode profile images into semantic vectors. This proved especially helpful for spotting misleading or AI-generated visuals. A learnable fallback embedding handled missing images, improving robustness. The fusion layer was then redesigned to apply per-modality projections before combining the features. This approach improved both performance and training stability by letting each modality specialize before integration.

The final model achieved a balanced F1 score around 63% and demonstrated flexibility when tested with different classification thresholds, highlighting trade-offs between bot precision and human recall. These evaluations show the model can be adjusted depending on whether false positives or missed detections are more critical in practice.

5. Future Work

There are several promising directions for improving this project. One potential enhancement is to fine-tune the CLIP image encoder along with the rest of the model, rather than keeping it fixed. This allows the encoder to be updated during training so it can better learn visual features that help distinguish bots from humans, instead of relying on general-purpose features from a pretrained CLIP model. While this approach can improve accuracy, it also requires more computing resources and careful tuning to prevent overfitting. Overall, it enables the model to focus more effectively on visual cues relevant to bot detection.

Additionally, the model could be extended by incorporating social graph information, such as who follows or interacts with whom. These network-based features might reveal hidden patterns in bot behavior. Overall, this project has shown that combining different types of information — text, behavior, and images — leads to a more powerful system. Future work can build on this foundation to handle even more complex and realistic fake profiles.

6. AI Usage Disclaimer

Parts of this projects have been developed with the assistance of OpenAI's ChatGPT (GPT-4). The AI was used to suggest architectural improvements based on project goals, support the debugging of implementation issues, reviewing experimental results and proposing further evaluation strategies. All content produced with AI assistance has been carefully reviewed, edited, and validated by me. I take full responsibility for the final content and its accuracy, relevance, and academic integrity.

References

1. ”TwiBot-22: Towards Graph-Based Twitter Bot Detection via Heterogeneous Information Network Embedding”, Feng Cao, Yichao Wu, et al. Link: https://arxiv.org/abs/2206.04564
2. ”RoBERTa: A Robustly Optimized BERT Pretraining Approach”, Yinhan Liu et al., Facebook AI. https://arxiv.org/abs/1907.11692
3. TwiBot-22 is the largest and most comprehensive Twitter bot detection benchmark to date. https://github.com/LuoUndergradXJTU/TwiBot-22