THE GHOST IN THE MACHINE

Samarth Rao

TASKS I DID:

- Task 0
- Task 1
- Task 2
- Task 3
- Task 4

(These are clickable)

A Brief Overview

This is my submission to the Precog Recruitment Task, 2026. My submission can be found at github.com/rao-samarth/human-or-ai

This report acts as **a summary** of my methodologies, findings and reports. If you want any more details, I have elaborated significantly in (i) the README.md's, and (ii) the markdown cells of all Jupyter Notebooks.

A video explanation of this project can be found at github.com/rao-samarth/human-or-ai/video.mp4

My main contribution is MATE - Memetic Algorithm for Text Evolution. Here, I modified the baseline genetic algorithm with saliency-guided mutations, lagrangian relaxations, and elitism. I showed that this performs better than a baseline genetic algorithm and is able to evolve a text from $8*10^{-7}$ probability of human-written to a 0.77 probability in 20 generations.

I also evaluated evaluated how this jailbreaking happened in significant detail by trying to manually alter text to make its human probability increase, realising that the key attributes my model is using are the tendency of humans to over-explain, the usage of rhetorics, and the usage of simpler synonyms in the human written text, among other reasons.

DATASET GENERATION

1. Class 1: Human-Written Text

For this class, I got books from a repository which contains all books on the Gutenberg project. I then had to choose authors. I initially started by choosing 2 authors - Sir Arthur Conan Doyle and P. G. Wodehouse. I then wrote scripts to clean the files of any non-essential text, randomly choose 25 books, and then choose 20 paragraphs which have 100-200 words each. This made our final class 1.

Almost. I then decided that I should add some authors with very unique, different writing styles, to see if our classifier would be able to separate that, especially when comparing class 1 with class 3. So, paragraphs from Mark Twain and Shakespeare got added to class 1.

2. Class 2: AI-Generated Text

Here, I used Gemini 3 Pro API to generate 500 paragraphs of text. I used a 2 pass approach for this. First, on the basis of the books selected in class 1, I had Gemini generate 5 key *"themes"* of the book. Then, two times for each theme, I told Gemini to generate a paragraph of 100-200 words, on the basis of that theme. My first time doing this, the paragraphs were too similar to each other. Since the main attribute was **authorship** and **not topic**, I did the following to make the paragraphs dissimilar:

1. Set the temperature to 1.0 while running

2. I added **“diversity modes”** and **“settings.”** These are a way to change the variation in framing and setting of the paragraph, while keeping the theme consistent

3. Class 3: AI Mimicry

Here, I used Gemini 3 Pro again, and prompted it to mimic the specific author of the book, for the same themes as those generated in class 2.

4. Class 4: A Failed Attempt at Fine-Tuning

I attempted to fine-tune a model on Unsloth on a portion of the books selected in class 0, and then create *“mimicry”* using that. Unfortunately, I kept facing overfitting and prompt leakage. I realised that the overfitting was because my learning rate was too high (2e-4) and the prompt leakage was because my training data consisted of raw text blocks without an instruction-response template.
In the interest of time, I could not fix this issue, but would like to generate this class in the future as well.

MATHEMATICAL ANALYSIS

I mathematically evaluated all the classes against each other, on the basis of what was given in the document, Zipf’s Law and Perplexity. The results of my findings are here:

Test	Class 1 (Human)	Class 2 (AI)	Class 3 (AI-Mimicry)	Key Finding
Type-Token Ratio (TTR)	Lower variance, mid-range	Higher, narrow band (0.6-0.8)	Higher, narrow band	AI has higher TTR but lower variance; humans show more dynamic range
Hapax Legomena	Lower	Moderately higher	Moderately higher	High temperature (1.0) leads to more rare words in AI

Zipf's Law (α exponent)	α ≈ 2.7 (steeper)	α ≈ 2.5 (flatter)	α ≈ 2.5 (flatter)	AI has flatter distribution (heavier tail) due to temperature
Zipf's Law (MAPE)	Higher error	Lower error	Lower error	AI follows Zipf more consistently
Dependency Tree Depth	Higher avg, wider range	Lower avg (2.56)	Highest avg (over-corrected)	Humans use more center-embedding; AI prefers right-branching
POS: Adj/Noun Ratio	Lower	Higher	Higher	AI over-describes compared to humans
POS: Noun/Verb Ratio	Higher (2.32)	Lower (1.72)	Mid-range	Humans prefer nominalizations; AI prefers direct verbs
POS: Adverb/Verb Ratio	Lower	Higher	Higher	AI over-modifies verbs with adverbs
Em-dashes	Lowest	Mid	Highest	AI mimicry over-uses em-dashes
Semicolons	Highest	Lower	Lower	Humans use semicolons more for nuanced connections
Colons	Highest	Lower	Lower	Humans introduce explanations more formally
Exclamation Points	Highest	Lower	Lower	Humans express emotion more freely
Double Quotes	Highest	Lower	Lower	Humans use dialogue/quotes more naturally
Flesch-Kincaid Grade	9th grade	7th grade	11th grade	AI writes simpler; mimicry over-corrects to harder
Flesch-Kincaid (variance)	High (σ ≈ 7)	Very low (σ ≈ 2)	Very low (σ ≈ 2)	AI maintains constant difficulty; humans modulate
Perplexity (GPT-2)	Higher	Lower	Mid	AI-generated text is more predictable to language models

BUILDING 3 DETECTORS

1. The Statistician

The statistician classifies the text purely on the basis of the mathematical findings above. I implemented both an XGBoost model and a Random Forest

model.

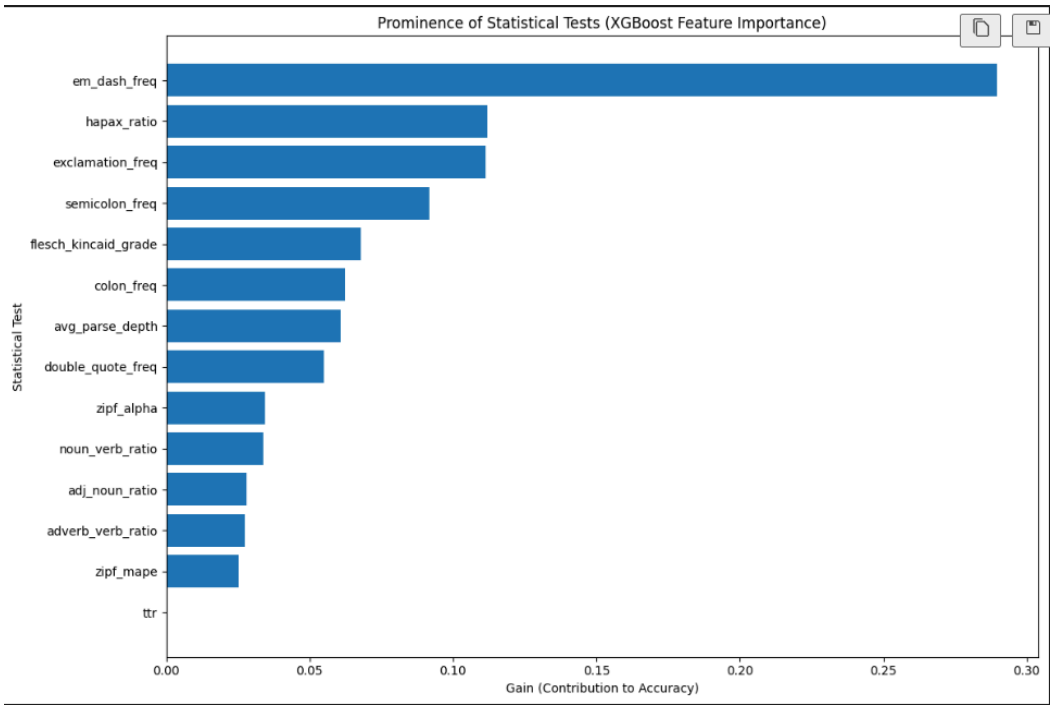
XGBoost Performance:

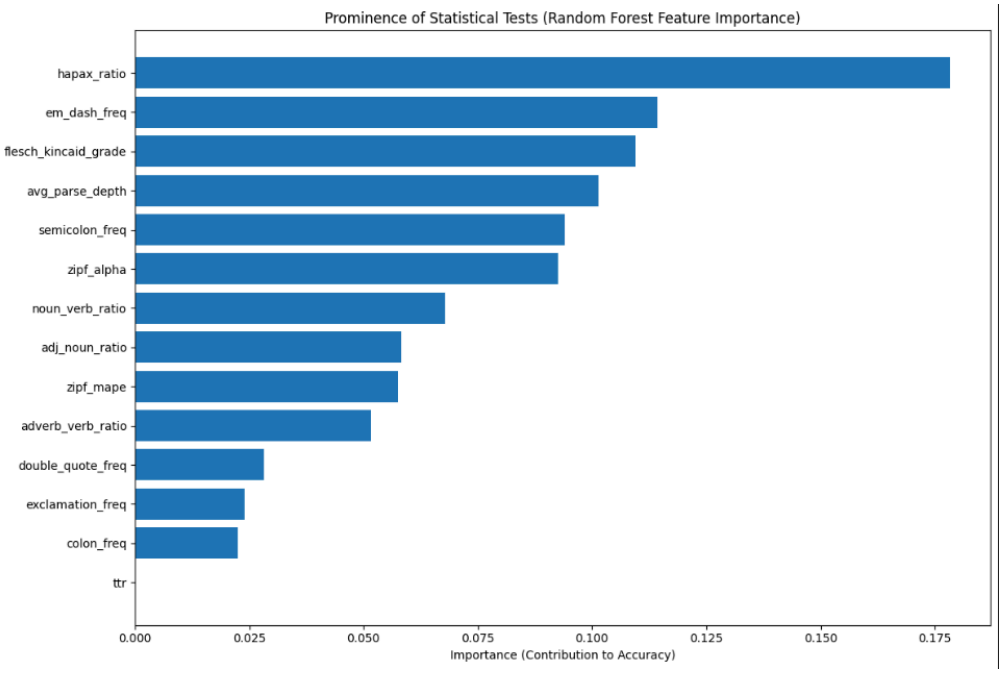
- Class 1 vs Class 2: 88.98% accuracy
- Class 1 vs Class 3: 93.00% accuracy
- Class 2 vs Class 3: 87.74% accuracy

Random Forest Performance:

- Class 1 vs Class 2: 87.00% accuracy
- Class 1 vs Class 3: 93.00% accuracy
- Class 2 vs Class 3: 85.32% accuracy

I also checked to see what the main classifying features were to both the XGBoost and Random Forest models.





Both models performed remarkably well, but as you can see: **em-dash frequency** was by far the most important feature for XGBoost (importance score >0.28), while it only scored 0.11 for Random Forest. I evaluated why. Here are my inferences:

- XGBoost is greedy by nature. Once it discovered that em-dashes were a high-signal feature, it exploited this finding aggressively.
- Random Forest builds trees in parallel using random subsets of features. In many trees, em-dashes weren't even available as a choice, forcing the model to find other reliable predictors like hapax ratio.

2. The Semanticist

This approach uses **Word2Vec** embeddings with a Multi-Layer Perceptron (MLP) to classify text based purely on semantic vector

embeddings.

I use Google's pre-trained 300-dimensional Word2Vec embeddings word2vec-google-news-300, applying a *"Bag of Means"* approach where individual word vectors are averaged to create a single paragraph vector. Stopwords ('the', 'and', 'in', etc.) are removed to reduce noise, ensuring the averaged vector retains sufficient unique signal for classification.

Semanticist Performance:

- Class 1 vs Class 2: 97.46% accuracy
- Class 1 vs Class 3: 95.91% accuracy
- Class 2 vs Class 3: 96.95% accuracy

Key Finding: The semanticist significantly outperformed the statistician. Even though the AI can count commas and mimic sentence lengths, it fails to fake the semantics which humans are able to uniquely author.

3. The Transformer

Here we use DistilBERT, a lighter transformer model. Transformers use self-attention. This is a mechanism that lets models understand which words matter most in context. For example, in the phrase *"The girl and her brother"*, the transformer associates "*her*" with "*The girl*" rather than treating each word independently. This context-awareness could help it pick up on the subtle stylistic patterns we couldn't identify in Task 1.

The transformer gave us **>99% accuracy** across classes. I found that to

be … *too* high. I decided to perform 3 tests to see if the model was indeed that good, or if there was an issue:

1. **LoRA Adapter on vs off:** With LoRA adapter enabled, the model achieved 99.64% confidence on AI text. Without it, confidence dropped to random guessing (~33% per class), proving the intelligence is contained in the fine-tuned adapter, not memorization.
2. **Checkpoint Evaluation:** While fine-tuning the model, 3 *checkpoint* models were created after each epoch. I ran the model on a clearly AI generated text, and I saw that its confidence grew slowly: 89.8% → 95.6% → 96.4%. This shows a learning curve which means that it did not just directly go from 33% → 99% by overfitting.
3. **Model Weights:** I inspected the raw adapter_model.safetensors file to ensure the training actually wrote complex patterns.

   Here, I found that the standard deviation in weights was 0.0019. This is normal. I read that typically the standard deviation in weights should be between 0.01 and 0.1 (towards the lower end of that is better), so this works.

   Key Finding: The file contains distinct, varied weights, proving the model successfully learned a complex mathematical representation of the author's style.

Before all of this, I also ran a quick sanity check by getting the works of Agatha Christie to check if it would work for a completely different author, and it did with the same confidence. This was expected though since I used a test-train split.

In conclusion

- The ~99% accuracy was achieved on a held-out test set (20% of data) that the model never saw during training, proving it generalizes to new examples
- We used LoRA, which froze 99% of the model's weights. By restricting the model to only 1.3% trainable parameters, we physically prevented it from having the capacity to memorize the training dataset, forcing it to learn stylistic patterns instead.

SALIENCY MAPPING

I used SHAP for this to calculate the contribution of specific tokens to the final classification. Given the high accuracy of the model, I did the analysis on those texts which have the *lowest* confidence. Here are my findings:

Worst Performing in Class	**Key Words (Top Features)**	**Key Findings**
1. Human-Written	their, waste, worlds, that, countless, turmoil, beyond	The model got confused because the author used very dramatic and descriptive language, which is exactly how AI often writes. It penalized the human author for sounding too much like a creative computer.

2. AI-Generated	handsome, a, probably, now, undoubtedly, plump, from, his	The model correctly identified this as AI, but its confidence dropped because the AI used specific, grounded details ("handsome fee," "plump from his strategic divestments") instead of just vague fluff. These small, concrete descriptions felt "human" enough to make the detector hesitate.
3. AI Mimicry	and, fine, noble, liked, concern, civic, water, complaint	The AI did too good of a job mimicking the author's voice. The model saw so many "classic" human-sounding words (like "fine and noble," "civic beautification") that it got fooled into thinking a real person wrote it, rather than an AI trying to copy a style.

THE MATE FRAMEWORK

Here I present MATE. This is the algorithm that I developed for my task-4, wherein I was able to successfully jailbreak my model into reporting AI-generated text as human

MATE treats adversarial text generation as a constrained optimization problem. It wants to maximise P(human | x) while maintaining semantic similarity fluency in speech (ie low perplexity).

Below, I explain the steps of MATE:

1. **Reducing the search space on the basis of saliency:** Instead of optimizing the entire sequence (search space size $\text{vocabsize}^{\text{wordsinfile}}$), MATE calculates the gradient of the detector's loss with respect to the input tokens. It identifies the top 20% of tokens contributing to the "AI" classification (high saliency) and marks them as mutable. All other tokens are fixed. By this, I reduce the search space by 90%.
2. **Initial Population:** I create the initial population using Gemini 3 Pro at 1.0 temperature to ensure the content remains same but there is significant variation
3. **Global Search:**
    a. Select top candidates from the population on the basis of fitness.
    b. Create an offspring from the selected parents. I use Gemini 3 Pro API for this as well. I ensure while prompting that while the styles are merged, the content remains the same.

    c. Regenerate the high saliency tokens in the new offspring text to explore more local structures (this is called mutation)
4. **Local Search:** After mutation, individual offsprings undergo simulated annealing. This is nothing but saying that the algorithm one-by-one perturbs the mutable tokens of a candidate. Improvements to the offspring are accepted.
5. **Lagrangian Relaxation:** I have placed penalties on perplexity and semantic Similarity. If the population violates a constraint then the associated penalty weight increases dynamically causing the fitness to decrease. Thus, the next generation has to prioritize fluency.

Key Findings:

- MATE successfully evolved text from **0.00%** to **77.7%** Human probability within 19 generations, crossing the decision boundary where baseline substitution attacks failed. It crossed into the *"Human"* category for the first time at the 12th iteration. It began to stabilise at ~78%. It did not go higher than that.
- I initially ran the whole algorithm without search space reduction. This was taking very long, ~35mins/generation. Search space reduction reduced that significantly, to ~6mins/generation.
- The Simulated Annealing component allowed candidates to escape plateaus where the standard hill-climbing greedy algorithm stalled.
- I also tried using Gradient-Guided Seeding for creating the initial population. With this, the initial fitness was -13 instead of -14.

THE PERSONAL TEST

Converting human-written code to make it sound AI generated was quite easy. All I had to do was keep Miriam-Webster open and find the most complicated sensible synonymous word / phrase.

The other way around though, was not as easy. Here, I initially tried the same strategy. After 3-4 iterations, I realised the only thing that would work was rambling. A lot. Like a lot, a lot, a lot, do you know what I mean?

FUTURE WORKS

1. I would like to finetune a model on Unsloth
2. I would like to think of more heuristics to speed up MATE
3. I want to use MATE to make MATE better. Run the algo, get a jailbroken offspring. Train the model on the offspring. Run the algo again. This idea is inspired by Generative Adversarial Networks
4. See if the model can detect watermarks. If it can, modify MATE to destroy the watermarks.